EDR: A Generic Approach for the Distribution of Rule-Based Reasoning in a Cloud-Fog continuum

Nicolas Seydoux [a,b], Khalil Drira [b] Nathalie Hernandez [a,*] and Thierry Monteil [b]
[a] *MELODI, IRIT, Toulouse, France*
email: {name.surname}@irit.fr
[b] *SARA, LAAS-CNRS,*
Université de Toulouse, CNRS, INSA, Toulouse, France
email: {name.surname}@laas.fr

Editors: Federica Cena, University of Turin, Italy; Armin Haller, Autralian National University, Australia; Maxime Lefrançois, École des Mines de Saint-Étienne, France
Solicited reviews: Maxime Lefrançois, École des Mines de Saint-Étienne, France; Two anonymous reviewers

Abstract. The successful deployment of the Semantic Web of Things (SWoT) requires the adaptation of the Semantic Web principles and technologies to the constraints of the IoT domain, which is the challenging research direction we address here. In this context we promote distributed reasoning approaches in IoT systems by implementing a hybrid deployment of reasoning rules relying on the complementarity of Cloud and Fog computing. Our solution benefits from the complementarity between Cloud and Fog infrastructures. Indeed, remote powerful Cloud computation resources are essential to the deployment of scalable IoT applications, and locally distributed constrained Fog resources, close to data producers, enable low-latency decision making. Moreover, as IoT networks are open and evolutive, the computation should be dynamically distributed across Fog nodes according to the transformation of the network topology. For this purpose, we propose the Emergent Distributed Reasoning (EDR) approach, implementing a dynamic distributed deployment of reasoning rules in a Cloud-Fog IoT architecture. We elaborated mechanisms enabling the genericity and the dynamicity of EDR. We evaluated its scalability and applicability in a simulated smart factory use-case. The complementarity between Fog and Cloud in this context is assessed based on the experimentation conducted.

Keywords: Distributed reasoning, SWoT, Semantic Fog computing, SHACL rules

1. Introduction

The maturity of Internet of Things (IoT) communication technologies is fostering a wide variety of industrial and societal applications, including home automation and industry 4.0 scenarios. However, the heterogeneity of IoT data and use cases raises interoperability issues constituting hurdles for the development of cross-domain IoT service platforms, leading to isolated application silos. The Semantic Web (SW) technologies and principles constitute an interoperability enabler providing expressive vocabularies to describe data and manipulate information. The domain at the interface between the SW and the IoT is called the Semantic Web Of Things (SWoT), and its emergence is not trivial. Even though the SWoT was envisioned as soon as the fundamental article of the SW [**?**] was published, where smart agents interact with devices in the user's environment, practical SWoT achievements were proposed in recent years only [**?**]. In particular, a

*Corresponding author. E-mail: {surname.name}@zwifi.eu.

core challenge the SWoT is facing is the deployment of SW technologies, which are resource-consuming, into IoT networks, characterized by constrained devices.

The integration of the SW stack into an IoT architecture is often centered on remote and powerful machines as in [**?**] or [**?**]. IoT data is centralized on such machines before being processed using SW technologies, in a Cloud computing approach [**?**]. SWoT deployment architectures consider pervasively distributed devices, with potentially limited computation and communication capabilities. Transporting data from these local devices to remote Cloud servers relies on multiple middle nodes. It introduces a delay in data processing, and can degrade applications' responsivity.

Distributing the SW stack among the multiple middle nodes between the Cloud servers and the IoT devices allows the SWoT architecture to avoid the drawbacks of a Cloud-centered processing. By doing so, the architectures evolve towards the Fog computing paradigm [**?**] that promotes data storage and processing **at the edge of the network** [**?**]. However, Fog computing is not introduced as a paradigm meant to replace Cloud computing: its limited computing capabilities, as well as the locality of the scale of its deployments, are not suited to support Cloud computing use cases. **Cloud and Fog computing are two complementary approaches** that, when associated, enable the deployment of complex SWoT applications [**?**].

In the scope on this paper, the purpose of semantic processing is, thanks to knowledge captured in ontologies, to **process data in order to produce meaningful business information**. One can suppose that knowledge about the deployed IoT system and its environment is modelled beforehand by the system administrators. However, business-specific knowledge needs may not have been identified when the IoT system is designed and might need to be injected into to reasoning system at runtine. Business-specific knowledge must therefore be modeled as self-contained bundles, and inserted into the system at runtime when needed. Moreover, when considering a distributed approach, all of the business knowledge might not be relevant in the context of all the nodes. If packaged into bundles that can be moved from node to node, business knowledge may be opportunistically distributed in the network. Inspired by the application bursting approach introduced in [**?**], we propose to consider modular applications to enable the distribution of some of their modules. Rules are a common way to capture business-level logic: a rule is a self-contained representation of a logical process.

Following these considerations, we consider in this paper rule-based reasoning: rules are used as representation of business logic, applied in a Knowledge base (KB) capturing the environment of the node. The proposed contribution is a **generic approach to the dynamic distribution of rule-based reasoning into a Cloud-Fog IoT architecture**, called Emergent Distributed Reasoning (EDR). EDR aims at harnessing scalability and latency issues by distributing reasoning rules among Fog nodes, while benefiting from the Cloud stability and permanent availability. Strategies for rule distribution are often application-dependent, with a wide variety of requirements due to the heterogeneity of IoT application domains. That is why EDR is a generic approach, that can be specialized depending on the desired rule distribution strategy. The work presented in this paper completes and extends two conference articles, [**?**] and [**?**], where some aspects of EDR and its refinements have been introduced. Novel work includes a more extensive presentation of related work, the detailed presentation of the vocabulary enabling the genericity of EDR and the description of the usage of the Linked Rules [**?**] principles. Complementary evaluations regarding the impact of distribution, and the impact of the execution of EDR on a constrained hardware are included, leading to a discussion analyzing the light shed by the obtained results on Cloud-Fog complementarity. The scientific challenge we faced considers three characteristics of the distributed reasoning system: scalability, responsiveness, and dynamicity. These characteristics are presented in detail in Section §2. In Section §3, existing work is introduced, to identify the added value of the present contribution. The core contribution is detailed in Section §4 and Section §5, and it is evaluated in Section §6. This paper is concluded in Section §7.

2. Desirable characteristics for the proposed solution

In order to capture the main characteristics of the contribution presented in §4 and §5, an illustrative industry 4.0 use case is introduced, that will drive the evaluations in Section §6. Elements considered in the use case are then generalized into the main desirable characteristics for the proposed approach.

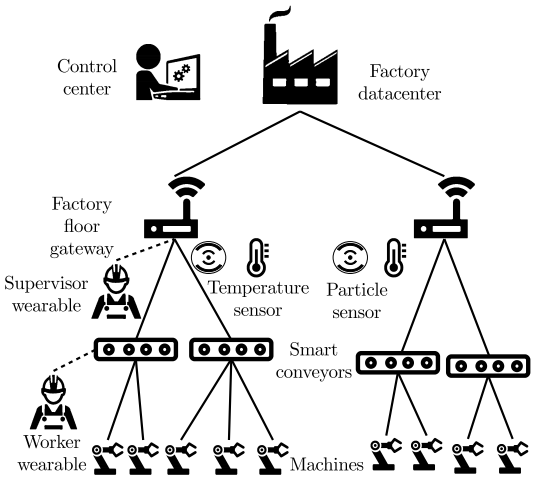


Fig. 1. Fog-enabled smart factory

2.1. Illustrative smart factory use case

Let us consider a production plant divided into two floors, processing different kinds of products. These floors are modular: the plant structure is subject to change in order to adapt to new productions. Each floor is equipped with conveyor belts carrying products from machine to machine for transformation. Devices are organized hierarchically: machines are connected to conveyors that are connected to the floor gateway that itself collects and delivers data to the factory datacenter. The factory is equipped with sensors in order to ensure the safety of workers: each floor is equipped with presence, luminosity, particle and temperature sensors, and the workers are equipped with wearables communicating with nearby conveyors. Observations from the different sensors are used in order to identify potentially harmful situations, and then notify the control center, where actions can be taken remotely. Unsafe situations are described with deduction rules, based on the semantic description of observations and of the environment. For instance, “The presence of a worker near an operating conveyor in a low luminosity environment is a personal security hazard” is a potential rule. Some rules are also dedicated to quality insurance: sensors available in the factory, such as temperature sensors, or sensors integrated to machines and to the conveyor, enable the continuous control of production quality. Some operations are temperature-sensitive, and a quality insurance rule is “The detection of a temperature above a certain threshold while machines are operating is a break in the cold chain”. As safety and quality insurance are time-sensitive applications, rule processing should be as fast as possible. Moreover, the mobility of some sensors (*e.g.,* worker wearables), combined with the modularity of the factory floors, create a dynamic network topology that evolves over time.

2.2. Scalability

Due to the modularity of the factory, the number of devices in the environment is not bounded a priori. In the specific industry 4.0 use case, this device count is unlikely to increase by multiple orders of magnitude, contrary to application domains of the IoT such as smart cities or connected vehicles, where large volumes of devices are involved.

Therefore, **scalability** is an important characteristic for a SWoT system, and the decentralization of reasoning is an enabler of such scalability [**?**]. However, the difference of computing power between Cloud and Fog nodes should not be neglected: the intrinsic capabilities of Cloud architectures enable a resource upscale impossible for Fog architectures. Moreover, Cloud infrastructures provide a stability that is complementary to the dynamic nature of Fog architectures. We propose therefore to leverage both the distributed nature of Fog computing and the permanent, powerful nature of Cloud computing by adopting a mixed approach.

2.3. Responsivity

In the proposed use case, the rules deployed in the system are used to detect potentially harmful situations, requiring the inferred notifications to be received by the control center as soon as possible. The proposed system should be able to reduce as much as possible the time from the appearance of an undesirable situation, and the moment where the control center is notified of such situation. **Responsivity** therefore is another desirable characteristic for our contribution.

Fog-enabled architectures trade computational power for proximity with data sources, which reduces the number of hops between data production and data processing. This reduces delays due to message delivery to distant Cloud nodes, and it is also interesting for situations where increasing the proximity with data sources decreases the complexity of reasoning. When decentralizing processing, the individual computational load is reduced for each node compared to a centralized approach, which can yield better performances [**?**]. Instead of funneling all the data towards

the Cloud before inferring higher level information, combining Fog computing and direct communication between Fog nodes and applications should enable a faster notification delivery.

2.4. Dynamicity

IoT systems are dynamic by nature: they are open systems, where devices can appear and disappear, as well as move from one point to the other. In the smart factory use case we introduced, the modularity of the factory floors might lead to changes in the network. More frequently, failures might happen, disconnecting a device. For energy saving purposes, not all the machines might also be powered permanently. Moreover, some devices are attached to workers that are mobile: they will be connected to different machines over time, leading to a dynamic network topology.

As the placement of rules in the network should adapt to the evolution of the topology, the last characteristic that we want for EDR is **dynamicity**. Depending on the devices available at a given moment on a given node of the network, not all the applicative rules will necessarily be relevant to this node. If a rule requires observations from a sensor that disconnects, carrying on applying this rule is a waste of resources. Applications consuming IoT data are also subject to change, and adapting the rule distribution strategy depending on the applications is also an aspect of dynamicity we consider.

3. Related work for rule deployment in SWoT architectures

As the proposed approach sets out to deploy reasoning rules among Fog nodes to enable deducing application-dedicated information from IoT data, state-of-the-art work dealing with logical rules for the IoT, distributed reasoning and processing on constrained nodes is presented.

3.1. Rules for the SWoT

Rules are logical twofold elements, composed of preconditions and postconditions. Preconditions represent a state of the world such that the rule should be applied in order to generate its post conditions, which represent a new state of the world. In our literature search, we identified two main types of rules associated to the SWoT [**?**]:

- **Production rules**, or deduction rules, in which preconditions are expressed as a logical expression, and postconditions are new knowledge which is the logical consequence of the preconditions.
- **Event-Condition-Action (ECA) rules**, in which preconditions are the association of a logical expression and an event triggering its evaluation, and the postconditions are actions to be executed if the preconditions are matched. Such actions are not limited to knowledge inference: they can be instantiated by running a piece of code.

As production rules are explicit deduction representations, they have been considered in IoT networks to express and share the correlation between sensor observations and high-level symptoms since early work on the SWoT [**?**]. [**?**] lists numerous works using rules for context-awareness in the IoT.

With the goal of facilitating rule reuse, Linked Rules principles have been proposed [**?**]. They apply the basic principles of Linked Open Data and Linked Open Vocabularies to rules: rules are designated by dereferencable International Resource Identifier (IRI)s, expressed in W3C-compliant standards, and they can be linked to each other. Inspired from the Linked Rules, the Sensor-based Linked Open Rules (S-LOR) [**?**] is dedicated to rule re-usability for deductions based on sensor observations. Production rules are a mechanism similar to Complex Event Processing (CEP) approaches, used for instance in [**?**], but the rule representation shifts from an ad-hoc rule format in CEP to a unified format in the SWoT.

[**?**] proposes a classification of production rules for the IoT, in order to identify recurring patterns. The authors distinguish rules enabling deductions from relations between nodes, and from relations between events (*i.e.* changes of the environment). In our contribution, we go further than this distinction by manipulating hybrid rules: their preconditions may rely both on conditions expressed on the nodes of the network, or on their environment.

3.2. Centralizing rule processing on Cloud nodes

In most existing approaches, *i.e.* [**?**], [**?**] or [**?**], production rules are handled by Cloud nodes. An example of Industrial IoT (IIoT) use case enabled by Cloud-based semantic rules processing is presented in [**?**]. This paper proposes a self-configuring smart factory: conveyors and machines produce data which is pro-

cessed on a Cloud node where user rules are used to make reconfiguration decisions. Rules are expressed in SWRL. The same formalism is used in [**?**], where production rules are computed in a central Cloud node in order to dynamically reconfigure the communication network topology between devices and the Cloud node. The inferred deductions are converted into network reconfiguration actions by ad-hoc agents. A similar hybrid approach is used in [**?**]: rules are expressed as production rules, but their postconditions may include ad-hoc properties dedicated to the triggering of actions.

In [**?**], a multi-agent blackboard approach is chosen to dynamically manage rules in a smart home. Observations are published to a central node, the Domotic Status Board (DSB), where they are checked against rules in order to trigger inferences and reactions: the rules considered combine properties of production rules and ECA rules. Rules are expressed in the Jena formalism[1], and an interface also allows users to control the system based on controlled grammar sentences. In this system, rules may be injected or deactivated at runtime. ECA rules are also used in a smart home use case in [**?**]: the authors propose an autonomic-like approach, where collected data is used to trigger actions of the system based on rules. A distinction is made between two types of actions stored in the KB: high-level actions, which are policies chosen by the user, and low-level actions, which are the actual implementations of the former, built by domain experts to hide the complexity of the system to the end-user. User preferences are expressed through a GUI, and converted from the GUI to KB individuals. During this conversion, appropriate low-level actions are selected to implement user-generated policies. The actual deployment topology is not presented, but the absence of any element indicating a distribution of the underlying platform leads to the conclusion that it is executed on a central node.

Production rules are used for context-awareness in a smart user space in [**?**]. Location information is combined to business knowledge, and to observations of the state of the user's environment, in order to make assumptions on the context. For instance, the following is a rule introduced by the authors: "IF the user is in an airport lounge with a low luminosity and the drapes closed THEN the user is sleeping". Such deduction is then used by context-aware services to adapt their behavior, materialized by ECA rules. Data required for the deductions are gathered into a central hub before being processed, and deductions are then sent to remote nodes.

Rules are deported on Cloud nodes rather than executed in Fog nodes when used to achieve context-awareness, such as in [**?**] or [**?**], in order to obtain a global execution context. However, in [**?**] for instance, some reconfiguration decisions could be taken considering only a local context. In this case, rules could be executed directly on Fog nodes.

3.3. Distributing rule processing on Fog nodes

The centralized architecture of the previously described papers raises issues such as the cost of semantic reasoning that increases rapidly with the size of the KB [**?**]. Fog computing offers a low-latency, resilient alternative for rule processing, even though the constrained nature of Fog nodes (compared to Cloud nodes) must be taken into account: processing power or bandwidth are critical resources. Centralization also requires all the content collected by IoT devices to be processed in the same place, while Fog computing makes computing power available closer to IoT devices. Fog computing enables contentto be processed with rules **where it is produced**, rather than requiring it to be transported to a remote node to be processed by Cloud computing. Rule placement in Fog architectures is thus a topic of interest for the SWoT

Most approaches for processing on constrained nodes focus on optimizations enabling such processing for a single node without considering the others. When considering a distributed execution composed of several Fog nodes, processing placement is not dynamic: all nodes execute the same rules, or each a predefined rule set statically assigned. For instance, even though it is not directly targeted at SWoT applications, the RETE algorithm proposed in [**?**] is dedicated to constrained nodes. RETE aims at reducing the memory requirements for production rule processing. This is a very interesting optimization, but it is dedicated to a single Fog node and does not consider distributed processing. [**?**] shows how gateways are Fog nodes capable of enriching data: observations are initially produced by legacy devices in ad-hoc formats. It is the gateway, communicating with devices using protocols adapted to constrained environments, such as CoAP, that enriches the data before forwarding it towards a Cloud node. Observations are therefore enriched on the edge of the network, and only the Fog nodes in di-

[1] https://jena.apache.org/documentation/inference/#rules

rect contact with legacy devices have to perform data enrichment. [**?**] or [**?**] propose to execute ECA in Fog architectures, used to automate the response of the system to a stimulus. However, both authors only consider one gateway executing the rules, and the ad-hoc rule format is not suited for rule exchange. The contribution introduced in [**?**] uses ECA rules associated to SW formalisms, namely SWRL and SPARQL. The authors use the Wiselib RDF provider [**?**], as well as CoAP and 6LowPan communication, in order to enable semantic processing directly on constrained nodes. How rules are distributed in the network is not discussed.

Regarding processing distribution in existing work, the dynamic nature of IoT networks should be considered. The topology of a network evolves as devices connect, disconnect, or move geographically. Therefore, a viable distribution of rules at a given moment is not guaranteed to remain optimal in the future, and **the distribution strategy should be adapted to the evolution of the network topology**. [**?**] does not detail the mobility strategy used for its mobile nodes, and each node applies all the rules regardless of their relevance to the content it aggregates. In [**?**], rule placement is static, in either Cloud or Fog nodes. [**?**] focuses on resource placement in a Fog-enabled IoT. The authors compute optimal deployment of application modules based on the representation of available resources in the Fog architecture compared to requirements expressed by applications. Module positions are static, and computed at the time of deployment. Rules are deployed on gateways in an IIoT context in [**?**]. The rules themselves are not expressed using SW formalisms, but they are combined to a semantic engine proposed in [**?**] in order to consume enriched data. The placement of rules in the Fog architecture is not dynamic, however ad-hoc mechanisms enable rule update at runtime.

EDR differs from previous proposals by several aspects in order to comply with the requirements described in Section §2:

- The locality of the knowledge involved in the rule deployment: each node only considers its own KB when propagating a rule.
- The **dynamicity** of rule deployment in the SWoT system at runtime, constantly adapting to the state of the topology in an event-driven behavior.
- The **genericity** of the approach, enabling its adaptation to various application-level strategies.

4. EDR, a generic approach to dynamically distributed rule-based reasoning

In this section, EDR, a generic approach to dynamically distributed rule-based reasoning supported by semantic Fog computing, is introduced. EDR is based on architectural assumptions that are presented in Section §4.1. EDR's functional overview is depicted in Section §4.2, before presenting the vocabulary used to describe EDR core functionalities in Section §4.3. Modular rules are at the core of EDR, the formalisms used to represent them and the roles of their modules is described in Section §4.4.

4.1. Assumptions on the underlying architecture

EDR is based on the hypothesis of a **hierarchical network topology**: nodes are organized in a tree-like structure, and only communicate with neighboring nodes, *i.e.* Cloud node and semantic-computing-enabled Fog nodes. The neighbours of a node are either its (unique) parent, or its children nodes. This assumption is made because such topologies are frequent in IoT networks, represented in studies such as [**?**], [**?**], [**?**] (based on the oneM2M standard), [**?**], or [**?**]. Based on this hypothesis, it can be assumed that there only is one path from any node to any of its ancestors, which simplifies our approach.

Applications are not deployed on a Cloud node belonging to the IoT topology: they are executed remotely on personal devices such as smartphones or laptops. **Rules represent applicative needs**: when deductions from sensor observations are required by an application, it injects the rule in the network in order to be provided directly with the deductions, instead of being forwarded raw data by the network and applying the rules itself.

It is therefore assumed that **Fog nodes can communicate with applications directly**. Rules are initially submitted by applications to the Cloud node, so it is the only node they know *a priori*. The Cloud infrastructure provides a unique permanent interface to the network, the dynamic Fog topology underneath is therefore transparent for applications.

We qualified the EDR as "dynamic", because nodes constantly re-evaluate their past decisions (*e.g* rule management or data propagation). Whenever an event occurs that may impact the current distribution of the rules in the network, each node locally recomputes the decision algorithm shown on Fig. 4, and introduced in the remainder of this section. However, the proposed

approach will adapt to the evolutions of the underlying topology based on the assumption that all events impacting the topology are considered by the appropriate nodes. In particular, the failure of a node must be captured by its parent, which must then propagate the consequences of this event on its own behavior to the rest of the network. Therefore, we consider failure management to be handled in the middleware layer, and it does not need to be explicitly handled in the scope of the applicative layer proposed in this contribution.

4.2. Overview of the EDR approach

In order to ensure decentralization, the algorithm of the EDR approach is executed in parallel on each node able to perform reasoning in the topology. EDR considers a neighbor-to-neighbor rule and data propagation, enabling a reduction the nodes' knowledge of the topology to a limited subset of the complete deployment. Thus, consistency of the knowledge only has to be maintained with neighbors, which limits required knowledge-related exchanges between nodes, and improves scalability. Due to the potential mobility and variable availability of Fog nodes, **EDR is meant to foster decision making in a local context for each node, leading at a large scale to the dynamic emergence of a desirable behavior**.

A parent node propagates a rule to its child if the parent considers that the child is empowered to apply the rule. This decision is made by the parent based on a **deployment strategy** embedded in the rule, as well as on the knowledge it has of said child. The deployment strategy captures the **criteria required for a node to process a rule**, and therefore characterizes if a child node is suitable to be forwarded said rule. In order to enable rule deployment, nodes exchange messages describing their characteristics, *e.g.,* their location, the type of data they observe, or the type of data they are interested in. To ensure the **dynamicity** of the rule deployment with respect to the **evolving network topology**, these messages are exchanged constantly, whenever nodes characteristics are modified. When a node makes a new deduction based on a rule, it sends the result to all the nodes it knows to be interested, including the application that submitted the rule.

The EDR approach itself is agnostic to the deployment strategy, which is defined by the rule implementer: that is why we qualify EDR as **generic**. The present section §4 is dedicated to the EDR approach, which defines the characteristics of a deployment strategy without implementing them. Such implementation is described with a refinement of EDR, EDR$_{\mathcal{T}}$, introduced in Section §5.

A functional representation of an EDR node is provided in Fig. 2: each node has a local KB, where knowledge necessary to the execution of EDR is stored. This knowledge is used to drive the basic functionalities of the node, and rules are used by the inference engine to update the KB.

Featured knowledge includes:

- the knowledge the node has of its own characteristics,
- the knowledge it has about its neighbors,
- the knowledge it has about the static organization of the environment such as the geographic or indoor location, or the relationship between the surrounding elements,
- the value of the last observations depicting the current state of the dynamic features of the environment,
- the rules that it has received from either applications or other nodes.

This knowledge is used to control the behavior of the node, composed of simple functionalities. A node is able to:

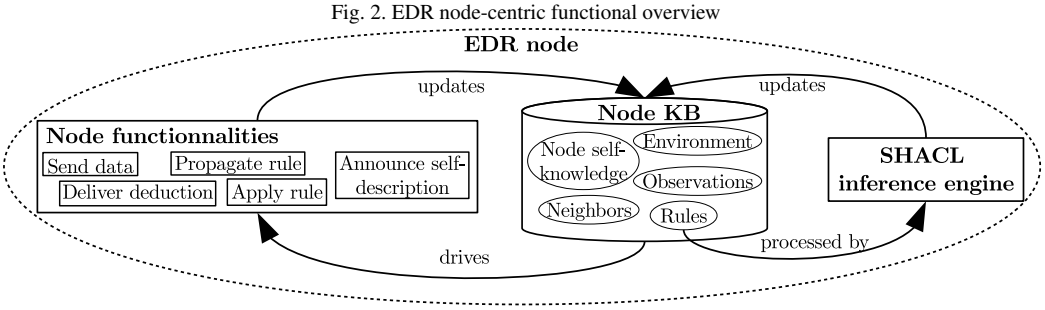
- Send a piece of data, typically a sensor observation, to a remote node,
- Propagate a rule to a remote node,
- Apply a rule on its knowledge base,
- Announce a description of its own characteristics to a remote node,
- Deliver a deduction obtained by processing a rule to a remote node,

How these node functionalities are related to the KB in the core EDR mechanism to enable the propagation of observations and rules is described in Section §4.3. The modular rule representation embedding the deployment strategy, and the updates of the KB they trigger, are detailed in Section §4.4. In this paper, the focus is on the propagation of rules, and on their execution, which leads to the production of new information. How this information may be used by the nodes to trigger real-world actions is not in the scope of this contribution. Using high-level deduction to trigger actions in an autonomic loop has been the topic of previous work [**?**].

4.3. A vocabulary driving the deployment mechanism

Node behavior is made quite simple on purpose, in order to decorrelate the rule-specific deployment strat-

Fig. 2. EDR node-centric functional overview



egy from the core algorithm on which EDR is based. Rule deployment strategies are dedicated to a particular purpose, *e.g.,* response time reduction or privacy enforcement, while EDR is generic. In order to support the genericity of EDR with a knowledge-driven method, node functionalities are based on a dedicated vocabulary, used to describe knowledge in the node's KB.

For instance, this vocabulary captures the hierarchical nature of the topology. Let the set of children of node n be referred to as $Children(n)$, and the singleton containing n's parent be noted $Parent(n)$. The relation between a node n and any $n_c \in Children(n)$ is expressed with the triple <*n*,*lmu*:*hasDownstreamNode*,n_c>[2,3], based on a nomenclature presented in [**?**]. The inverse relation exists, to express the connection between a node n and its parent $n_p \in Parent(n)$: <*n*,*lmu*:*hasUpstreamNode*,n_p>.

A description of all the functionalities of the nodes, and of the vocabulary that drives them, is provided in Section §4.3.1. Further details about the announcement functionality are provided in Section §4.3.2, especially with regard to the consumption of data. Finally, the scope of the announcements is studied in Section §4.3.3.

4.3.1. Basic node functionalities

Each functionality relies on dedicated triples, and a node implements its behavior based on the description held in its KB. How these triples are inferred from the deployment strategy is described in the next section §4.4. Before detailing how the strategy triggers nodes functionalities, let us examine the vocabulary describing said node functionalities.

[2]Namespaces are available in Appendix A

[3]Individuals such as n and n_c are identified with an IRI in the triples

Announce self-description: When a node connects, disconnects or changes characteristics, it notifies its neighbors of its self-representation. Since a notification is sent at each update of the node's state, the perception of a node by its neighbors remains consistent with its evolution over time. Two mechanisms support this announcement:

- a partial update, in which a node adds statements to its description already held by the target
- a complete update, in which the representation of the node is completely erased by the target before being updated.

These mechanisms add information about a node by exchanging light messages containing partial representations, while removing outdated statements with the complete update. A particular node characteristic that is declared in the announcement functionality is the type of data in which a node is interested, captured with the predicate *edrt*:*isInterestedIn*, which is used in the data sending functionality. The announcement functionality is extended by the mechanisms described in Section §4.3.2 to control which characteristics of the node are propagated, and the scope of this propagation in Section §4.3.3.

Apply rules: When a node n receives a new observation, either from its own sensors or children, n executes the rules r stored in its KB if the description of r contains <*r*,*edr*:*isRuleActive*,true>.

Deliver deduction: If the processing of an observation with rule r by node n leads to a deduction δ, δ is sent to each node belonging to $\bigcup n_{consumer}$ where <$n_{consumer}$,*edr*:*consumesResult*,*r*> is in the KB of n. The application that submitted the rule r to the network is known as the rule originator o, and is represented by the triple <*r*,*edr*:*ruleOriginatedFrom*,*o*>. The originator of a rule is considered as a consumer of rule results, in order to enable deduction delivery to applications. The deduction delivery functionality is separated

from the interest notification part of the announcement functionality for flexibility.

Send data: When node n receives an observation of type ρ_t, if $n_p \in Parent(n)$ has declared its interest for this type, the observation is forwarded toward n_p. Observations are exchanged lazily: if a node n receives an observation of type ρ_t, and knows no other node interest in such type, the observation is not forwarded. Such interest is represented in node n KB with the triple <n_p,*edr*:*isInterestedIn*,ρ_t>. The notification of the interest is considered as a characteristic of the node, managed in the announcement functionality.

Propagate rule: A node sends a rule to one of its neighbors if it considers that its target is capable of applying the rule, such a consideration being part of the rule deployment strategy. In the case where rule r should be propagated towards node n_{target} by n, the triple <r,*edr*:*transferableTo*,n_{target}> is present in n's KB.

4.3.2. Controlling node characteristics propagation

The EDR algorithm depends on the exchanges between neighboring nodes of their mutual descriptions, enabled by the announcement functionality. However, presupposing node characteristics relevant to any deployment strategy that will be implemented to refine EDR is not possible. In order to remain agnostic to the deployment strategy, EDR relies on a dedicated vocabulary used to describe which of each node's characteristics should be announced to its neighbors. A node has two types of neighbors: its parent, and its children, and since the parent is unique (according to our assumptions) while the children are potentially many, two approaches are devised.

Announcing characteristics to a node's parent: Let us consider a node n, with a characteristic represented by a property *hasCharacteristic* and captured in its knowledge base such that <n,*hasCharacteristic*,v>, with v either a literal or an individual. When announcing its characteristics to its parent, n searches in its KB for all the triples where it is the subject, and the predicate is typed as *edr*:*ParentAnnouncedProperty*. If the property *hasCharacteristic* is such that <*hasCharacteristic*,*rdf*:*type*,*edr*:*ParentAnnouncedProperty*>, then the triple <n,*hasCharacteristic*,v> is part of the self description sent by the node n to its parent because *hasCharacteristic* is considered a relevant characteristic of n.

Announcing characteristics to a node's children: The announcement mechanism from parent to children is quite similar to the one from children to parent, with the difference that children may be many. Therefore, the class *edr*:*ChildrenAnnouncedProperty* has two subclasses to distinguish two possible cases:

- *edr*:*AllChildrenAnnouncedProperty* denotes a characteristic that is systematically announced to all the node's children.
- *edr*:*SomeChildrenAnnouncedProperty* denotes a characteristic that should only be announced to a subset of the node's children.

This distinction is made to give flexibility to the deployment strategy designers.

In the case of a characteristic captured by a predicate of type *edr*:*SomeChildrenAnnouncedProperty*, each child eligible to be proxied the new characteristic must be represented explicitly with the predicate *edr*:*announceTo*, which requires the reification of the announced characteristic. In order to be announced towards child node $n_c \in Children(n)$, the triple <n,*hasCharac*,v> is transformed into the following reified statement: *statement* rdf:$subject$ n_c; rdf:*predicate* *hasCharac*; rdf:$object$ v; *edr*:*announceTo* n_c. The choice of the children to which the characteristic should be announced is application-specific, and is therefore part of the deployment strategy. As the rest of the deployment strategy, it is embedded in rules as described in Section §4.4.

The interest of a node for a type of data, denoted by the predicate *edr*:*isInterestedIn*, is managed as a node characteristic. Therefore, depending on the deployment strategy, the interest of nodes is classified as one of the subclasses of *edr*:*ChildrenAnnouncedProperty*. More details about this particular predicate are provided in Section §5, with the instantiation of a concrete deployment strategy.

4.3.3. Propagating knowledge beyond neighbors

EDR is designed for neighbor-to-neighbor rule and data propagation: a node n only communicates with $n' \in Parent(n) \cup Children(n)$ (with the exception of deduction delivery). However, such design may hamper the propagation of rules, by preventing the diffusion of knowledge required by the deployment strategy to make decisions as to where the rules should be placed. We want to avoid the situation in which the characteristics of a node $n_c \in Children(n)$ makes it adequate to apply a rule which is held by $n_p \in Parent(n)$, but n cannot apply the rule, and therefore n_p does not

propagate the rule to n, preventing its eventual propagation to n_c. A complementary functionality is thus described by the EDR vocabulary to enable such diffusion of knowledge describing node characteristics: **proxying**.

The proxying mechanism implemented in EDR is inspired from [**?**], where reasoning nodes act as proxy for the characteristics of legacy nodes unable to process enriched data. In EDR, each reasoning-enabled node has a similar role, and proxy characteristics of its neighbors. Such proxying is bidirectional: the characteristics of a node's parent are proxied towards its children, and vice versa. Specifically, node n proxying a characteristics of $n_p \in Parent(n)$ towards any $n_c \in Children(n)$ means that n announces such characteristics to n_c as if it were its own. An example of proxied node characteristics, detailed in Section §5.2.2, is the interest of a node for a data type, briefly introduced here for the sake of illustration.

If a node n wants to be notified whenever a temperature observation is available, it notifies its children $n_c \in Children(n)$ of such interest. If any n_c collects temperature observations, it will forward such observations towards n. Moreover, each n_c will in turn notify that it is **itself** interested in temperature observations to any node $n_{cc} \in Children(n_c)$. Any node n_{cc} collecting a temperature observation will therefore send it to n_c, which will itself send such an observation to n. The characteristic of the initial node n (here, the interest in temperature) has indeed been proxied to n_{cc} by n_c: n_{cc} only has knowledge of n_c, and communication is kept strictly between direct neighbors. To support this mechanism, two classes of properties are defined in the EDR vocabulary: *edr*:*ParentProxiedProperty*, and *edr*:*ChildrenProxiedProperty*.

Characteristics proxied from children to parent: Let us assume that $n_c \in Children(n)$, and that n_c has a characteristic expressed by the triple <n_c,*hasCharacteristic*,v>, that should be proxied towards $n_p \in Parent(n)$. Such information about the predicate v is materialized by the triple <*hasCharacteristic*,*rdf*:*type*,*edr*:*ParentProxiedProperty*>. When receiving the description of n_c, n checks for the presence of properties classified as *edr*:*ParentProxiedProperty*. Since *hasCharacteristic* is such a property, the node n updates its own representation towards n_p by sending the triple <n,*hasCharacteristic*,v>, therefore proxying the capacity of n_c.

Characteristics proxied from parent to children: The proxying mechanism from parent to children is similar to the one from children to parent. Contrary to the announcement functionality, the multiplicity of children is not considered: all the children are proxied any received parent characteristic. Such policy is made necessary by the locality of decision-making enforced by EDR. On the one hand, a node n receiving a characteristic to proxy $n_p \in Parent(n)$ does not have the contextual knowledge that leads n_p to announce this particular characteristic to n. On the other hand, the node n_p does not have a detailed knowledge of the topology below n, and therefore cannot make any assumptions about to which children in particular n should proxy the characteristic of n_p.

It is possible that the proxying mechanism and the announcement mechanism lead to conflicting behaviors. In particular, a node may have chosen not to announce a characteristic of its own to some of its children, but be required to proxy the same characteristic instead of one of its ancestors. In this case, the proxying mechanism supersedes the announcement mechanism, and any proxied characteristic is processed as a *edr*:*AllChildrenAnnouncedProperty*. For instance, if a node n did not announce its interest for a data type ρ_t to $n_c \in Children(n)$, n will nonetheless announce such interest to n_c if $n_p \in Parent(n)$ notifies n of its own interest for ρ_t.

4.4. Rule representation and deployment

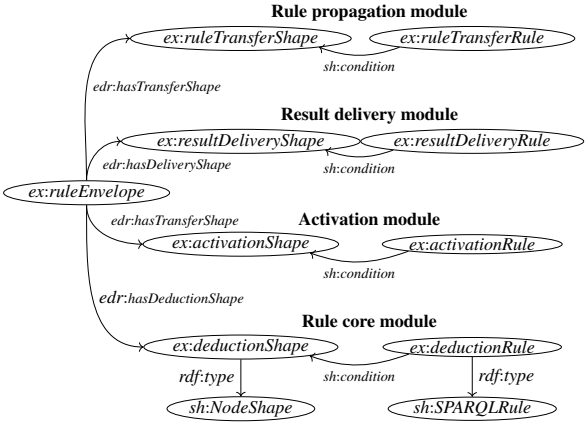
4.4.1. Rule modular structure

EDR rules are composed of several modules, as represented on Fig. 3. Each of these modules enables some node functionalities:

- The Rule propagation module triggers the rule forwarding functionality
- The Result delivery module triggers the result delivery functionality
- The Activation module triggers the rule application, the data consumption and the result delivery functionalities.
- The Rule core module contains the actual business logic of the rule

The intelligence regarding rule deployment is located in the rules, and not hard-coded into EDR or statically attached to nodes. The behavior of the algorithm at a global scale can thus be parameterized at a fine granularity, for each rule. Rules are represented in SHACL, and the modules are based on the SHACL

Fig. 3. Rule modules



advanced functionality named "SHACL rules". Each module is composed of two parts: a SHACL rule, that inserts deductions into the KB, and a SHACL shape that determines whether the rule is applied or not. An example rule, named $r1$, is provided online [4]. In the remainder of this section, a generic description of these rule modules and their roles is given, each illustrated in $r1$. An implementation is proposed in Section §5, where specific behaviors dedicated to a particular strategy are described.

In order to associate all the modules to a rule represented as a single individual in a node's KB, we introduce the notion of **rule envelope** as a reification mechanism. The envelope of an EDR rule is an individual subject of triples whose predicates are *edr*:*hasTransferShape*, *edr*:*hasApplyShape*, *edr*:*hasDeliveryShape* and *edr*:*hasDeductionShape*. The rule envelope is especially useful in the rule deployment process, when all the modules of a given rule must be collected for the rule to be propagated to a remote node.

4.4.2. Rule modules

Core module The operational part of the rule, containing the application-dedicated inference, is referred to as the **rule core** module. The core module is based on a predicate logic rule used to deduce high-level information, similar to the rules introduced in the use case in Section §2.1. Let r^{core} be such a rule core module, noted as $r^{core} : \Gamma_1 \wedge ... \wedge \Gamma_n \rightarrow \Delta_1 \wedge ... \wedge \Delta_m$, where $\Gamma_1 \wedge ... \wedge \Gamma_n$, designated as the **body** of r^{core}, is a conjunction of conditions and $\Delta_1 \wedge ... \wedge \Delta_m$, designated as the **head** of r^{core}, is a conjunction of deductions. The rule core module only encompasses applicative deduction logic: it is unrelated to the deployment of the rule. r^{core} is only evaluated when the whole rule r has been declared active on a node in the deployment process, *i.e.* if the triple <r,*edr:isRuleActive*,true> is in the node's KB.

Rule transfer module The **rule transfer module** determines on which remote nodes the rule may be deployed, according to a rule-specific deployment strategy. This condition is expressed as a SPARQL query embedded in the SHACL rule being the conditional part of the rule transfer module. The deduction part of the module infers the triple <r,*edr:transferableTo*,n'>, enabling the rule forwarding mechanism of the node (*c.f.* Section §4.3.1). The transfer module of a rule r is denoted $r^{transfer}$.

Rule activation module The **activation module** detects if the current node is suitable to apply the rule itself. If the conditional part of rule r activation module determines that the current node is suitable to apply r, the activation of rule r is made explicit by the triple <r,*edr:isRuleActive*,true>. In the case where some node characteristics are conditionally proxied towards children (*edr*:*SomeChildrenProxiedProperty*), the rule activation module may infer reified statements as described in Section §4.3.3. This case is illustrated in more detail in Section §5.3. The activation module of a rule r is denoted $r^{activation}$.

Result delivery module The **result transfer module** enables the forwarding of deductions to other nodes that are not the originator of the rule, such as the parent n' of a node n if n' applies a rule r' that consumes the deductions made by a rule r applied by n. By default, the originator o of a rule r is assumed to be interested in the results of r, denoted with <o,*edr*:*consumesResult*,r>. If a remote node n' is interested in the deductions made by rule r, the result transfer module infers that <n',*edr:consumesResult*,r>.

4.4.3. Dynamically managing modules activation

The rule core must be computed each time a new observation is received by the node, in order to check if new deductions may be inferred. However, it is worth noting that the other rule modules only need to be evaluated when the rule is received, or when the topology evolves, *e.g.,* with new productions by children, new consumptions by parent, or nodes connecting/disconnecting.

The SHACL standard is such that by default, when reasoning on a KB containing SHACL shapes and

[4] https://w3id.org/laas-iot/edr/iiot/r1.ttl

rules, all of them are considered[5]. In order to reduce the computation load, and to only process rule modules when needed, a SHACL functionality is used: the reasoner does not consider shapes or rules *r* such that <*r*,*sh:deactivated*,true>. The modules of a rule *r* are therefore only activated for a reasoning step when *r* is received, or when the topology evolves.

The appropriate modules, *i.e.* all except the core module, are classified as *edr:NodeSensitiveComponent* (as opposed to what would be a "Content sensitive component"). Therefore, a unique query activates or deactivates rule modules related to deployment, for all the rules stored in a node's KB.

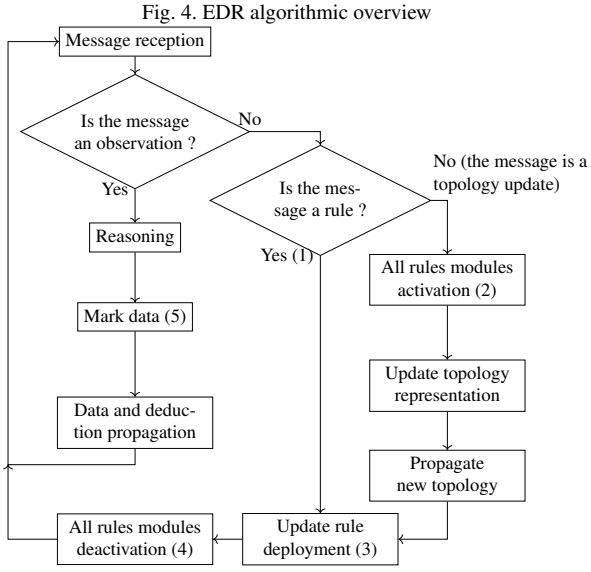
Deployment module management is represented on Fig. 4, in an overview of the algorithm. When a rule is initially received, all of its modules are active. No activation is required when receiving a new rule, marker (1) on Fig. 4. The rule deployment update, marker (3) on Fig. 4, is performed by the reasoner. Since no other rule deployment module has been activated since the new rule has been received, and by default these modules are deactivated, only the deployment of the newly received rule is computed.

In the case where the node receives information about a topology update, such as the connection or disconnection of a node or the change of characteristics of a known node, it is possible that the rule deployment should be updated accordingly. For this reason, for all the rules stored in the node's KB, the deployment modules are activated upon the reception of a topology update, as seen in marker (2) on Fig. 4. The received change is then integrated in the KB, and if necessary the new topology is propagated to parent nodes, before performing a reasoning step computing the deployment rule modules. If the placement rule needs to be updated due to the topology change, the new deployment is enforced by activating or propagating rules in compliance with the deductions and the EDR vocabulary, before deactivating the rule deployment modules, marker (4) on Fig. 4.

If the received message is an observation, no rule deployment update is required. The only active rule modules are the core modules for rules that the node should process, and they are used by the reasoner to test if new inferences are possible. The marking and propagation of deductions is discussed in Section §4.4.4.

[5]See Section §4.3 of the recommendation https://www.w3.org/TR/shacl/#validation-definition

Fig. 4. EDR algorithmic overview



4.4.4. Leveraging the unique identification of rules

EDR rules are compliant with the Linked Rules principles [**?**], and in particular they are uniquely identified by an IRI. The identification of rules being shared among all nodes, provenance can be traced for a given deduction. Two purposes have been identified for this traceability: the avoidance of redundant computation, and the update of rules at runtime.

Preventing redundant computation With the rules being uniquely identified among all nodes, it is possible to mark observations when they have been processed with a rule, successfully leading to a deduction or not. After an observation *o* has been involved in a reasoning step with rule *r*, a new triple is added to the observation description: <*o*,*edr*:*usedForDeductionBy*,*r*>. This marking prevents an observation being processed multiple times with the same rule when it is propagated from one node to another. Considering this marking or not is up to the rule implementers: for instance, the strategy presented in Section §5 takes it into account, so that each observation is at most processed once by each rule for performance issues. Depending on the propagation strategy, it may be necessary to process the same piece of data with the same rule in multiple contexts, in which case the marking may be ignored. The marking of observations with the *edr*:*usedForDeductionBy* property is shown on Fig. 4, marker (5).

If a rule is submitted by multiple applications to the topology, the uniqueness of the identifier also avoids redundant processing. In a node's KB, each rule can

be associated to several originators, indicating that the deduction should be sent to several applications. Expressed in an application-specific namespace, two identical rules would be applied twice, leading to a waste of resources.

Updating rules at runtime The use of a unique dereferencable identifier also incrementally modifies rules at runtime, so that the operation of the monitored system is not interrupted. Modifying rules allow applications to fine-tune their behavior according to a feedback loop that considers either previous responses to inputs, or external factors (*e.g.,* seasonal change, or regulation evolution). When a rule r is received by a node n, if r's IRI is already known by n, all the triples describing the rule are compared to the triples stored in the node's KB.

If the newly received version of the rule is different from the version held by the node, then the rule representation is updated in the KB, and the rule is processed as if it were a new rule. All the modules of the rule are evaluated, and changed characteristics of the node, if there are any, are propagated to its neighbors as in any topology change. However, it is possible that the new representation of the rule is no longer applicable by children of the current node (or by their descendant in the case of proxying), to which the former version of the rule had been previously propagated. In the regular EDR algorithm, the rule would not be forwarded to such children, but in this case this is an issue: two different mutually exclusive versions of the rule are executed in the topology.

To tackle this issue, an object property is used: when a node n transfers a rule r to $n_c \in Children(n)$, it adds the triple <r,*edr*:*transferredTo*,n_c> to the rule description stored in its KB. When n updates r, it transfers the new version of r towards any $n_c \in Children(n)$ if n_c received the former version of r by searching for this predicate. If it is no longer relevant, *i.e.* if the new version of r is not transferable to n_c (according to its transfer module), the triple <r,*edr*:*transferredTo*,n_c> is removed from n's KB. Even if n_c is not able to apply the new version of r (as determined by the application module of the rule), updating its KB enforces the consistency of the representation of r across the network. The same process is carried on recursively in order to ensure that all the nodes of the topology eventually have an up-to-date representation of the rule. If n had transferred r to n_c because n_c was proxying some characteristics of its descendants, two situations are possible. Either n_c directly applied r without transferring it, in which case once n_c receives the updated version of r the propagation stops, or n_c transferred r to any $n_{cc} \in Children(n_c)$. In this case, n_c's KB contains the triple <r,*edr*:*transferredTo*,n_{cc}>, and r's update is propagated towards n_{cc} thanks to this triple, and so on.

This approach however leaves a consistency issue unsolved: during the propagation of the new rule version, the two mutually exclusive versions of the same rule are both active. There is no guarantee that the latest version of the rule has been propagated successfully at any point in time after its injection in the network. A way to solve this issue is to attach a version number to the rule with the *owl*:*versioninfo* annotation property. This version information is then attached to deductions made with the rule, so that applications are aware of the version of the rule that leads to any deduction.

5. Refining EDR with EDR$_\mathcal{T}$

As has been said in Section §4, EDR is a **generic** approach to rule deployment among semantic-enabled Fog nodes, agnostic to the criteria according to which rules are propagated in the topology. In order to demonstrate the applicability of EDR, the present section is dedicated to **EDR$_\mathcal{T}$, an approach refining EDR by implementing a deployment strategy**.

After introducing the EDR$_\mathcal{T}$ core principle in Section §5.1, the knowledge required by nodes executing EDR$_\mathcal{T}$ is described in Section §5.2. How EDR$_\mathcal{T}$ is implemented in rule modules is discussed in Section §5.3. The behavior of nodes executing EDR$_\mathcal{T}$ is detailed in Section §5.4, in order to capture the complete deployment process.

5.1. Implementing a deployment strategy based on property types with EDR$_\mathcal{T}$

The purpose of EDR$_\mathcal{T}$ is to **bring rules as deep as possible in the topology, in order for them to be processed as soon as possible**, while limiting unnecessary message exchanges. EDR$_\mathcal{T}$ is meant to reduce the delay between the moment observations able to trigger a deduction by a rule are produced by devices, and the moment said deduction is received by the rule originator. Due to the assumed hierarchical nature of the network, the deeper a node is in the topology, the fewer descendants it has. A node processing a rule deeper in the hierarchy will thus apply said rule less often, on a smaller KB, since it should receive fewer updates

from its descendants. Since reasoning on a smaller KB yields better performances [**?**], propagating rules as deep as possible among reasoning nodes reduces computing complexity. Therefore, in EDR$_\mathcal{T}$, a node receiving a rule propagates said rule to any of its children able to process it.

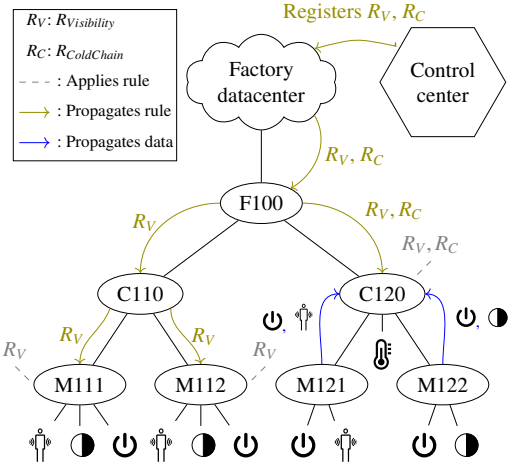
EDR$_\mathcal{T}$ implements a deployment strategy **driven by the *t*ypes of properties produced by nodes**. These properties can be either environmental properties captured by sensor observations (*e.g.,* luminosity) or higher level properties deduced by other rules (*e.g.,* comfort). Nodes characteristics capturing these productions are exchanged between neighbors in order to identify the lowest possible node able to process the rule. These characteristics are captured in the rule modules to enable the deployment process. The conditional shape of rule modules is based on both **property types consumed by the rule** and **property types produced by neighboring nodes** to infer the node behavior.

To manipulate these property types in the following sections, the *body* and *head* notations introduced in Section §4.4.2 are extended. We introduce $body_t(r_x) = \{\gamma_1, ..., \gamma_{n'}\}$ and $head_t(r_x) = \{\delta_1, ..., \delta_{m'}\}$ where γ_i designates the property type of Γ_i, and δ_j the property type of the deduction Δ_j. It should be noted that not all Γ_i or Δ_j used in the rule are relevant to the EDR$_\mathcal{T}$ approach.

Let us consider $R_{Visibility}$ and $R_{ColdChain}$, illustrative rules provided in natural language in Section §2.1. A translation of $R_{Visibility}$ in based on description logic is: $Location(?l) \wedge Presence(?l, ?o_1) \wedge ?o_1 = True \wedge Luminosity(?l, ?o_2) \wedge ?o_2 < 300L \wedge Machine(?m) \wedge Activity(?m, ?o_3) \wedge ?o_3 = True \wedge locatedIn(?m, ?l) \rightarrow LowMachineVisibility(?m)$. For this rule, the defined predicates behave as follows: for the conditions, $body_t(R_{Visibility}) = \{Presence, Luminosity, Activity\}$, and for the deductions, $head_t(R_{Visibility}) = \{LowMachineVisibility\}$. *Location* is a property type that is not considered by the deployment strategy implemented by EDR$_\mathcal{T}$. For $R_{ColdChain}$, represented in description logic in Section §6.3, $body_t(R_{ColdChain}) = \{Temperature, Activity\}$, and $head_t(R_{conveyor}) = \{ColdChainBroken\}$.

The deployment of $R_{Visibility}$ and $R_{ColdChain}$ by EDR$_\mathcal{T}$ in an extract of the simulation topology is shown on Fig. 5. Both rules are submitted by the control center application to the Cloud node, and are deployed among Fog nodes. Nodes applying the rules (*e.g.,* machines M111 and M112 for $R_{Visibility}$) directly provide the control center with deductions, which is not represented on the figure for the sake of legibility.

Fig. 5. Example of EDR$_\mathcal{T}$ deployments



5.2. *Node characteristics at stake in EDR$_\mathcal{T}$*

5.2.1. *Node knowledge on itself*

A node n has in its KB information about the property types of the data it produces, denoted by the predicate $own\_productions(n)$. Data produced by node n is either collected by sensors to which n is directly connected, or obtained as deductions when n applies a rule. When a reasoning-enabled node is connected to a sensor, it enriches the raw observation, and propagates the enriched observation on the network, which ensures that the observation is only enriched once. In the topology displayed on Fig. 5, node M111 is connected to three sensors: $own\_productions(M111) = \{Presence, Luminosity, Activity\}$. The production of observations by node n for a property type ρ_t is denoted <n,edr:producesDataOn,ρ_t>.

5.2.2. *Node knowledge on the topology*

A node n knows its parent in the network tree-like hierarchy. On Fig. 5, $Children(C110) = \{M111, M112\}$, and $Parent(C110) = \{F100\}$. The node communicates its characteristics to these neighbors to support the deployment strategy implemented by EDR$_\mathcal{T}$. Such characteristics include the types of the data produced by the node, as well as the types of data consumed.

Announcing productions: The transmission of rules among nodes organized by EDR$_\mathcal{T}$ is driven by the knowledge each node has on the network around itself. Productions are propagated from children to par-

ent, denoted by the triple <*edr*:*producesDataOn*,*rdf*:*type*,*edr*:*ParentAnnouncedProperty*>.

In order to enable the propagation of rules towards nodes that are not direct neighbors, the proxying mechanism introduced in Section §4.3.3 is implemented for property types productions: <*edr*:*producesDataOn*,*rdf*:*type*,*edr*:*ParentProxiedProperty*>.

To illustrate the proxying in more detail, let us define $productions(n) = own\_productions(n) \cup productions(Children(n))$. Node n announces itself to $n_p \in Parent(n)$ as a producer of $\rho_t^i, \forall \rho_t^i \in productions(n)$. For instance, on Fig. 5, $productions(C120) = \{Activity, Temperature\}$, with $own\_productions(C120) = \{Temperature\}$. If n_p was not a producer of the property type ρ_t, it includes a new triple in its KB <n_p,*edr*:*producesDataOn*,ρ_t>, and forwards this triple to $n_{pp} \in Parent(n_p)$. If node n_p was already a producer for ρ_t, its characteristics remain unchanged, and the information propagation stops.

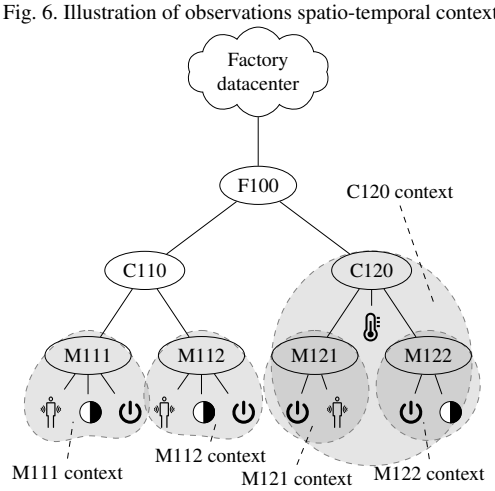
Announcing consumptions: As it has been discussed in Section §4.3.1, in order to limit unnecessary exchanges, data is exchanged lazily based on the node consumption announcement functionality. A node n has to explicitly advertise its interest for a property type ρ_t to each $n_c \in Children(n)$ in order to be notified when new observations are received or new deductions are made. In particular, a node is interested in a property type ρ_t when it applies a rule r such that $\rho_t \in body_t(r)$. The interest of a node n for a property type ρ_t is represented by the triple <n,*edr*:*isInterestedIn*,ρ_t>, and <*edr*:*isInterestedIn*,*rdf*:*type*,*edr*:*SomeChildrenAnnouncedProperty*>.

The interest of n for ρ_t is only announced to $n_c \in Children(n)$ such that <n_c,*edr*:*producesDataOn*,ρ_t>. Moreover, if some nodes $n_c^i \in Children(n)$ are able to apply the rule r themselves, node n will forward r to n_c^i, rather than notifying n_c^i of its interest for ρ_t. The details of the rule deployment strategy are provided in Section §5.3. In Fig. 5, M121 announced to C120 that it produced *Activity*, and C120 notified M121 of its interest for *Activity* in order to receive future observations.

5.2.3. Exploiting the contextual locality of IoT data

The rule deployment strategy supported by EDR$_\mathcal{T}$ is based on the assumption that the **correlation between pieces of data is embedded in the network topology**. IoT data is strongly bound to a spatio-temporal context [**?**], and the distribution of Fog nodes reflects the distribution of features observed by sensors. From this hypothesis, it can be inferred that the context of a node is a subset of the context of its parent. To illustrate this

Fig. 6. Illustration of observations spatio-temporal context



claim with $R_{ColdChain}$ previously introduced, it means that if it is possible to apply $R_{ColdChain}$ with activity and temperature observations collected by the same gateway, it is not necessary to compare the same activity observations with temperature observations collected elsewhere. As IoT data are highly contextual, applications do not necessarily need to reason over a complete KB to obtain relevant results. EDR is therefore suitable for rules exploiting this context by correlating data sharing an identical context, *e.g.,* the correlation of temperature and luminosity in the context of a single room for $R_{ColdChain}$.

The relation between the spatio-temporal context and the topology is represented in Fig. 6, where each gray area represents the context of a Fog node. Our assumption is that, since both M111 and M112 contexts contain enough information to process rule $R_{Visibility}$, the luminosity from M111 context and the temperature from M112 context will never be processed together by $R_{Visibility}$.

In the case of the C120 context, since neither M121 nor M122 produce the information necessary to process $R_{ColdChain}$ or $R_{Visibility}$, both nodes send their observations to C120. The fact that C120 is the parent of both M121 and M122 is considered a hint that the context of M122 is closer to the context of M121 than, for instance, to that of M112. The proximity of context is associated to the distance of the closest common ancestor: M121 and M122 share a parent, while the closest common ancestor to M121 and M112 is F100, at a distance of 2 hops from both nodes. Since M121 and M122 are closer to each other than M122 and M112, there is a higher chance for the luminosity observation from M122 to lead to a deduction based on $R_{Visibility}$

when processed with presence from M121 rather than M112.

As for context proximity, context inclusion is impacted by the hierarchy. A context A is considered included in a context B if the elements of context A are also available in context B. On Fig. 6, the C120 context includes the M121 and M122 contexts, since activity, presence and luminosity values are propagated to C120. Since C120 applies $R_{ColdChain}$, M121 and M122 provide it with activity observations, which it processes with its own temperature value observations.

If, as in our case, the scope of rules is not broader than the context in which they are applied, applying rules deeper in the hierarchy does not impact the completeness of the result. However, if the rules are not adapted to the topology in which they are deployed with EDR$_\mathcal{T}$, some deductions will be inferred in a centralized approach that would be missed when data is processed in a decentralized manner. For instance, let us consider two sensors producing respectively observations of types ρ_1 and ρ_2, connected to the same node n, and a rule r consuming ρ_1 and ρ_2. EDR$_\mathcal{T}$ will eventually deploy r on n, and none of the observations of type ρ_1 and ρ_2 produced by n will be processed by r outside of the context of n. This is the intended behavior of EDR$_\mathcal{T}$, but it limits its applications to some types of rules, such as rules performing the aggregation of several values of the same type. For instance, a rule that sums electrical consumptions and compares the total to a fixed value cannot be executed successfully by EDR$_\mathcal{T}$, because its scope will be larger than the contexts in which it will be distributed, that is any node producing electrical consumption observations.

This behavior is adapted to rules supporting deductions for time-sensitive applications, which is the focus of the present contribution, and cannot be applied to aggregation rules, where time series or multiple instances of the same property types are considered. This choice is motivated by the assumption that aggregation rules are more likely to be used in applications supporting long-term reporting and decision support, where the time constraint is not strong, and thus outside the scope of this contribution. The EDR approach and its refinements (such as EDR$_\mathcal{T}$) do not aim at replacing semantic Cloud computing, but seek to complement its capabilities with semantic Fog computing. This is a second reason not to support aggregation rules.

To ensure decidability, only DL-safe rules are considered, and EDR is only suitable for stratified rule sets. Cyclic dependencies between rules are not resolved. When a node applies rule r, it is considered as producer of all $\delta_i \in head_t(r)$, and this production information is used for the deployment of any rule r' such as $body_t(r') \cap head_t(r) \neq \emptyset$. However, a non stratified rule set where rules r and r' coexist such that $body_t(r') \subseteq head_t(r)$ and $body_t(r) \subseteq head_t(r')$ cannot be processed successfully by EDR, and neither r nor r' will be propagated or applied.

Listing 1: $r^{transfer}_{ColdChain}$ shape

```
SELECT $this WHERE {
  FILTER NOT EXISTS {
    $this a lmu:Node ;
    edr:producesDataOn adr:Temperature,
      adr:MachineState ;
    lmu:hasUpstreamNode [a lmu:HostNode;].
    FILTER NOT EXISTS {
      {ex:coldChainRule
        edr:transferredTo $this.}
      UNION
      {ex:coldChainRule
        edr:transferableTo $this.}}}}
```

5.3. Implementation of EDR$_\mathcal{T}$ in rule modules

The behavior of a node implementing EDR$_\mathcal{T}$ is embedded in the modules of EDR$_\mathcal{T}$-compliant rule. For now, these rules are built manually: the property types feature in the rule body and head are identified when the rule is written, and the modules are built accordingly. The knowledge required for the processing of each module is local to the node performing the reasoning process. For the sake of legibility, the SHACL representation of the rules is not reproduced in the present paper, but it is available online[6].

5.3.1. Rule Transfer module

The purpose of EDR$_\mathcal{T}$ is to **transfer each rule to the lowest possible node in the architecture**, to be applied as early as possible. The propagation of a rule r_x from node n to node n' is considered relevant if $n' \in Children(n) \wedge body_t(r_x) \subset productions(n')$, which brings it closer to sensors.

This condition is expressed in Lst. 1, an extract of the SHACL shape constituting $r^{transfer}_{ColdChain}$.

Since it is assumed that rules are initially submitted to the Cloud node, the neighbor-to-neighbor propagation is only considered downwards in the topology. Each node that handles the rule in the deployment process keeps its representation in its KB. It is not neces-

[6] https://w3id.org/laas-iot/edr/iiot/visibility.ttl, https://w3id.org/laas-iot/edr/iiot/coldchain.ttl

Listing 2: $r^{activation}_{ColdChain}$ shape

```
SELECT $this WHERE {
  FILTER NOT EXISTS {
    $this a lmu:HostNode.
    $this lmu:hasDownstreamNode ?tempProvider,
    ?activityProvider.
    ?tempProvider edr:producesDataOn
      adr:Temperature.
    ?activityProvider edr:producesDataOn
      adr:MachineState.
    FILTER EXISTS {
      $this lmu:hasDownstreamNode ?lowerNode.
      FILTER(
        ?lowerNode = ?activityProvider
        || ?lowerNode = ?tempProvider)
      FILTER NOT EXISTS {
        ?lowerNode edr:producesDataOn
          adr:Temperature, adr:MachineState.}}}}}
```

sary to re-propagate a rule upwards: if a node ceases to be able to apply a rule, the change should be considered by the activation module of the rule held by its ancestors, as it is detailed in Section §5.4.

Incrementally, the rule r will converge toward nodes such that, for any node n of them:

- n can no longer **propagate** r, *i.e.* $\forall n' \in Lower(n), body_t(r_x) \not\subset productions(n')$,
- n is able to **apply** the rule r, *i.e.* $body_t(r_x) \subset productions(n)$.

These are the nodes able to apply the rule that are the closest to the original data producing: propagating the rule deeper in the hierarchy is not necessary. Such a node is represented on Fig. 5 with gray dashes connected to $R_{Visibility}$ and $R_{ColdChain}$.

5.3.2. Activation module

In order to apply a rule r, a node n must be the lowest common ancestor to the producers of property types in the rule body. Such a node has a set $\mathcal{P}$ of children (either sensors or other Fog nodes) **partially producing the rule head**. Individually, none of the children produce all the elements of the rule head, but combined, their productions enable the processing of the rule. It is characterized as such: $\exists\mathcal{P}$, such as $\forall n_c \in \mathcal{P}$, <$n$,*lmu:hasDownstreamNode*,n_c> and $\exists\{\rho_t,\rho'_t\} \subseteq body(r)$, <$n_c$,*edr:producesDataOn*,$\rho_t$> and $\neg\exists$ <n_c,*edr:producesDataOn*,ρ'_t>, and $\forall\rho_t \in body(r), \exists n_c \in \mathcal{P}$,<$n_c$,*edr:producesDataOn*,$\rho_t$>. Lst. 2 gives a SPARQL implementation of these conditions applied to $r^{activation}_{ColdChain}$.

If the conditional part of module $r^{activation}$ determines that the current node is suitable to apply r, some deductions are inferred. The activity of rule r is made explicit by the triple <r,*edr:isRuleActive*,true>, and the nodes $n' \in \mathcal{P}$ are identified as providers of the data type which r now consumes. The interest of n for the consumption of the nodes $n' \in \mathcal{P}$ is announced, as it is captured by the <?interest,*edr:announceTo*,?partialDataProvider> triple in the SHACL rule. The object of the interest, represented as a reified statement, will be bound to any partial production of the rule head by a child of n. The interest of the rule originator o is also denoted with <o,*edr:consumesResult*,r>. These inferences enable both the **rule application** and the **rule result forwarding mechanisms** as described in Section §4.3. The SPARQL CONSTRUCT embedded in the SHACL rule for the $r^{activation}_{ColdChain}$ module is provided in Lst. 3. The focus of the SHACL shape, materialized by the `$this` variable, captures the IRI of the node applying the rule in its own KB. It is defined in the SHACL documentation as the only element shared natively between the SHACL conditional shape and the SHACL rule said shape conditions: the `$this` captures the node violating the shape defined in the condition. This is why some elements characterizing the child nodes of the current node need to be recaptured in the WHERE clause of the $r^{activation}_{ColdChain}$ rule, while the `$this` is already bound to the current node.

5.3.3. Result delivery module

In EDR$_{\mathcal{T}}$, the condition of the result delivery module checks if a node expressed interest for the type of deductions yielded by the rule. If there exists a triple <n',*edr:interestedIn*,ρ_t>, with n' a remote node and ρ_t an element of the rule r's head $head(r)$, then the result transfer module infers that <n',*edr:consumesResult*,r>.

5.4. Unraveling the main steps of EDR$_{\mathcal{T}}$

Nodes executing the EDR algorithm maintain a coherent view of their neighborhood, and deploy rules with respect to this perception of their environment according to the strategy implemented by EDR$_{\mathcal{T}}$. The neighborhood of a node is modified when a new node connects or a known node disconnects, and when the productions or consumptions of a node are modified. The main events impacting the exchanges of a node with its neighbors are therefore: when its characteristics are changed (which includes startup and disconnection), when receiving a new rule, and when receiving a new piece of data. In the following, the behavior of EDR$_{\mathcal{T}}$ for each of these events is described to refine the high-level description given on Fig. 4.

Listing 3: $r^{activation}_{ColdChain}$ rule

```
CONSTRUCT {
  $this edr:isInterestedIn adr:MachineState,
    adr:Temperature.
  $this edr:producesDataOn ex:ColdChainBroken.
  ?interest a rdf:Statement;
  rdf:subject $this;
  rdf:predicate edr:isInterestedIn;
  rdf:object ?partialProduction;
  edr:announceTo ?partialDataProvider.
  ex:coldChainRule edr:isRuleActive
    "true"^^xsd:boolean.
  ?originator edr:consumesResult ex:coldChainRule.
} WHERE {
  $this a lmu:HostNode.
  {
    $this lmu:hasDownstreamNode
      ?partialDataProvider.
    ?partialDataProvider edr:producesDataOn
      ?partialProduction.
    FILTER NOT EXISTS {
      ?partialDataProvider edr:producesDataOn
        adr:MachineState, adr:Temperature.
    }
  } UNION {
    ex:coldChainRule edr:isRuleActivable
      "true"^^xsd:boolean.
  }
  ex:R1 edr:ruleOriginatedFrom ?originator.
  BIND(STRAFTER(str(?partialProduction), "#")
    AS ?productionName)
  BIND(URI(CONCAT(str($this), ?productionName,
    "Interest")) AS ?interest)}
```

When changing characteristics Sensors are the primary source of data for the network. The data they produce is collected by their reasoning-enabled parent. When semantic computing-enabled nodes start, they try to connect to their sensors children of which they have *a priori* knowledge. How nodes discover and gather information about sensors can be a process tightly related to the underlying technology, or hard-coded in the node KB.

Nodes connected to sensors announce the property types they (and potentially any $n_c \in Children(n)$) produce to their parent node, according to the announcement functionality captured in the triple <*edr*:*producesDataOn*,*rdf*:*type*,*edr*:*ParentAnnouncedProperty*>. Similarly, when a sensor or a lower node providing data of type ρ_i to node n disconnects, n announces its updated characteristics if they have been transformed, *i.e.* if the disconnected node was the sole producer of ρ_i.

In the case when the node already held some rules, their placement might need to be updated according to the new topology denoted by the received message. In order to adjust the rule deployment accordingly, rule modules dedicated to such deployment, namely application, transfer and delivery modules, are activated, processed in a reasoning step, before being deactivated again as detailed in Fig. 4. The deductions yielded by this reasoning step, based on the *edr* vocabulary, are used to control the node behavior as described previously. The use of these modules is similar when a new rule is received, as it is described in the next section. A part of the propagation of $r_{Visibility}$ in the illustrative deployment provided in Fig. 5 is represented as a sequence diagram on Fig. 7.

When receiving a rule When node n receives a new rule r, n evaluates whether it can apply r directly, and/or if it should propagate r to some of its children by performing a reasoning step with all modules of r activated. Based on the deductions produced by this reasoning step, some node functionalities are activated if necessary:

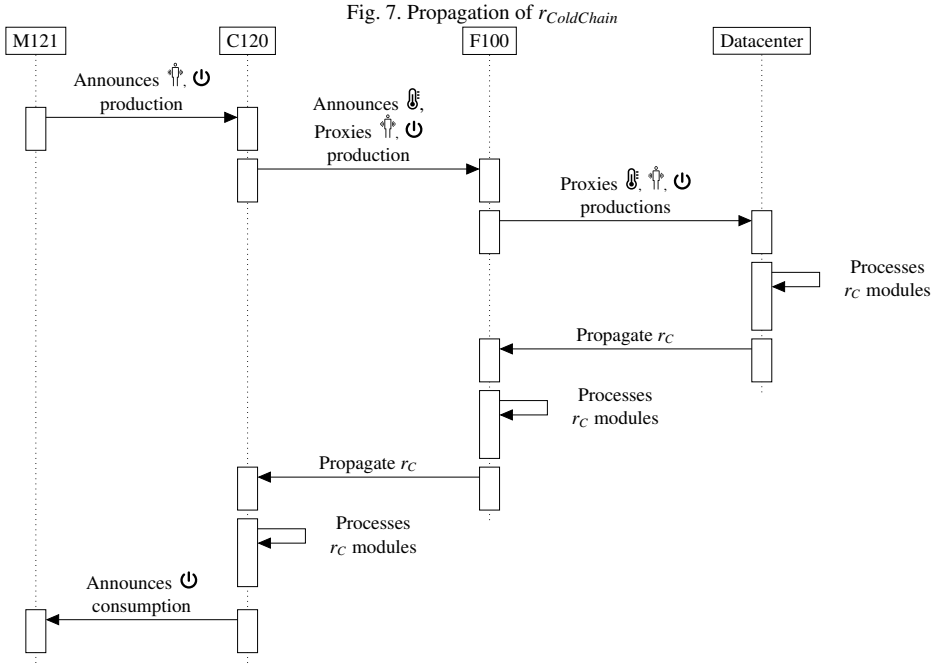
- If the rule r is applicable by the current node, the productions of n are updated by $r^{activation}$. n notifies its parent of its new productions, *i.e.* the head of r. Being able to produce the deductions of a rule is processed like a characteristics change, described in the previous section. If the applicability of rule r is enabled by the productions of some children of node n, the interest of n for their productions has been added in the KB, as well as the necessity for their notification of such interest. Node n thus notifies these children of its interest for these properties.
- The rule r is propagated to child nodes marked suitable by the rule transfer module. Local metadata is added to rule r in order to keep track of the lower nodes to which it has been transmitted with the predicate *edr:ruleTransmittedTo*. Such metadata is not added by the rule transfer module, but by the node after the completion of the propagation to the target.

When receiving new data Different kinds of data can be received by node n:

- raw observations directly produced by a sensor connected to n
- enriched observation or deduction sent to n by node $n_c \in Children(n)$

If the received observation is raw, node n enriches it by annotating it with an ontology before its processing as a new enriched observation. If the piece of data is either an enriched observation or a deduction, it is directly integrated to its KB and processed.

Fig. 7. Propagation of $r_{ColdChain}$



The data, of property type ρ_i, is first sent to $n_p \in Parent(n)$ if it is a consumer of ρ_i. Then, node n checks if new deductions can be obtained by applying the rules it has marked up as active. When receiving new data, a node does not need to activate the rule modules for activation, transfer or delivery: only the core of the rule is relevant. If the rule body matches the KB of node n, and postconditions of type δ_j are deduced, these deductions are propagated to n_p if it is consumer of δ_j. Since rules are applied on the local KB of node n, there is no impact of data distribution on reasoning complexity. A new reasoning loop is simply applied each time new data is received. The deductions yielded by rule r are also **directly** sent to r's originator(s). Therefore, applications are notified continuously by the nodes as those nodes apply the rules, instead of being notified by a restricted set of central nodes.

6. Experimentation

As EDR is a generic approach, it cannot be subjected to a quantitative evaluation by itself: it must be refined by a concrete approach implementing a deployment strategy. The evaluations presented in this section are dedicated to EDR$_\mathcal{T}$, refining EDR with a deployment strategy aiming at **reducing the deduction delivery delay**.

In order to compare the proposed contribution to a panel of baselines, different delivery mechanisms are introduced in Section §6.1. By default, EDR$_\mathcal{T}$ delivers deductions directly to applications. The proposed alternative delivery mechanisms implement variations of this approach, by propagating deliveries differently across the network. A centralized deduction baseline is also introduced.

The setup in which the evaluations were performed is described in Section §6.2, along with the references to the code used for running the experiments. Two characteristics of EDR$_\mathcal{T}$ are then assessed: its scalability in Section §6.4, and its responsivity in Section §6.5.

6.1. Deduction delivery mechanisms

The purpose of the evaluations presented in this section is to compare the performances of centralized Cloud-based and decentralized Fog-based approaches to reasoning. It aims at distributing reasoning among Fog nodes in order to perform computation as close as possible to the sensors producing observations. The baseline to which EDR$_\mathcal{T}$ should be compared is a centralized approach, where raw data is sent up to a Cloud node to be processed by rules. Since the propagation of rules for semantic Fog computing is performed neighbor-to-neighbor, it seems logical that raw

data is propagated in the same way back to the Cloud node. However, such comparison would be biased by the necessity for each piece of data to transit through multiple hops from Fog to Cloud nodes. In order to limit the impact of transfer time, and focus on processing time, new hypotheses are considered: in some configurations, Fog nodes will deliver deductions to Cloud nodes, instead of communicating directly with applications. Similarly, for centralized processing, Fog nodes should be able to deliver raw data to Cloud nodes, instead of an indirect propagation. These different configurations are referred to as "Deduction delivery mechanisms".

Unlike rule deployment strategies, deduction delivery mechanisms are **decorrelated from the rules**: they are variations of the "Deduction delivery" functionality described in Section §4.3.1. Therefore, the propagation of rules, the deductions they yielded and data is described as intended according to ad-hoc strategies (here, $\text{EDR}_{\mathcal{T}}$) through the EDR vocabulary, but for experimental purpose this propagation can be altered at the node level, preventing rule deployment or rerouting deduction delivery. Five deduction delivery mechanisms are compared in our experiments:

- **Cloud-Indirect-Raw (CIR)** is the baseline approach: the rules are only kept in the top Cloud node, and raw observations are forwarded neighbor-to-neighbor from the nodes that collect them toward the central node. The Cloud then delivers deductions to applications. Applications are notified by the Cloud node, and not by Fog nodes, in all delivery mechanisms except the last one.
- **Cloud-Direct-Raw (CDR)** is also an approach where rules are not deployed, and only processed in the central Cloud node. In this configuration, the observation producers directly send raw observations to the Cloud node, where they are used for rule-based deductions. Such a delivery mechanism enables to measure the impact of transfer time on deduction delay when centralizing raw data for processing. To implement this configuration, the interest proxying mechanism presented in Section §5.2.2 is altered. Nodes that are not the upper node in the hierarchy propagate the interests they receive without proxying them.
- **Cloud-Indirect-Processed (CIP)** is a hybrid delivery mechanism: rules are deployed among Fog nodes according to $\text{EDR}_{\mathcal{T}}$, and deductions are propagated neighbor-to-neighbor towards the Cloud node before being delivered to applications. CIP mirrors the delivery mechanism of CIR, with a decentralized reasoning. The purpose of CIP is to measure the performance gain when distributing reasoning even when communication is only possible neighbor-to-neighbor in the Fog infrastructure. To modify the result delivery behavior, whenever a node propagates a rule, it declares itself as the originator of said rule instead of the previously registered originator. Processing rules based on semantic Fog computing means that the propagation of observations is limited to the Fog nodes applying rules consuming such observations, instead of going all the way up the Cloud node.
- **Cloud-Direct-Processed (CDP)** is another hybrid mechanism where rules are processed by Fog nodes, but deductions are delivered directly to the Cloud node instead of applications. It is the Cloud node that performs the delivery to applications. In this case, the purpose is to measure the impact of centralized delivery in a decentralized reasoning context. To implement CDP, when forwarding a rule it has received, the Cloud node declares itself as the originator instead of the application. Deductions can also be propagated among Fog nodes if a node explicitly expressed its interest.
- **Application-Direct-Processed (ADP)** is the purely decentralized strategy that we propose for $\text{EDR}_{\mathcal{T}}$, where rules are processed based on semantic Fog computing and deductions are delivered directly to applications. In this case only, a deduction that has been inferred in the network will not be hosted by the Cloud node before being delivered.

The characteristics of the different delivery mechanisms are summarized in Tab. 1, where their important features are highlighted:

- whether rules are propagated among Fog nodes or not,
- whether deductions are propagated neighbor-to-neighbor or directly delivered,
- whether Fog nodes communicate with the Cloud node or directly with applications.

All these characteristics are illustrated in an example and illustrated on Fig. 8, where the propagation of raw data and deductions according to the different delivery mechanisms is represented. In the case of deduction delivery, it is assumed for the sake of clarity that deductions are made in the lowest Fog nodes. The manipulation of the EDR behavior by implementing dif-

Table 1
Delivery mechanisms summary

Approach	Rules propagation	Neighbor-to-Neighbor content delivery	Fog-App communication
CIR	✗	For data ✓	✗
CDR	✗	For data ✗	✗
CIP	✓	For deductions ✓	✗
CDP	✓	For deductions ✗	✗
ADP	✓	For deductions ✗	✓

Fig. 8. Delivery mechanisms

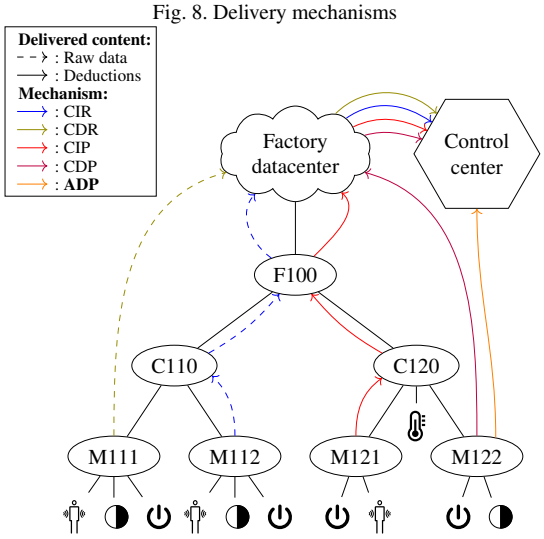


Table 2
Experimental setup

	RAM	Cores	CPU
Server	32GB	32	3.0GHz
Laptop	16GB	8	2.6GHz
RPi 3	1GB	4	1.4GHz
RPi 2	1GB	4	900MHz

ferent delivery mechanisms enables the comparison of centralized (CIR and CDR) and distributed approaches (CIP, CDP, ADP), and the comparison of approaches based on direct (CDP, CDR) and indirect (CIR, CIP) communication with the Cloud node.

6.2. *Experimental setup and implementation*

6.2.1. *Hardware setup*

In order to assess the distributed nature of the approach, and its suitability for constrained Fog nodes, the experimental setup includes a Raspberry Pi 2 and a Raspberry Pi 3, a laptop and a server, described in Tab. 2.

In order to measure the tradeoff between decentralization and the loss of computing power when reasoning on Fog nodes, experiments are run twice, in two different environments:

- In the first case, the complete topology is emulated on the same server, each node being run as an individual process. This environment is referred to as "**single-host execution**". Such an execution environment makes testing more practical.
- In the second case, the topology is distributed across different machines listed on Tab. 2. This environment is referred to as "**multi-host execution**". Such execution environment is more realistic than single-host execution, since it includes constrained nodes. However, large scale experimentation on such decentralized environments is harder to achieve, since it requires multiple machines. Ideally, each node should be executed on a separate computing system, but this would require too many resources. We compromised by executing multiple nodes on a single constrained machine. The necessity to run the experiments on multiple machines at the same time also creates technical issues making the testing process more complex.

6.2.2. *Software setup*

The use case topology is simulated for the experiments. Simulated nodes are organized in a tree-like hierarchy, with a Cloud node at the root, sensors at the leaves, and Fog nodes in between. Each physical machine running the simulation hosts multiple virtual nodes, composed of an HTTP server, a KB, a SPARQL engine, and a code base[7].

Experiments are run by simulating a building setup with sensors generating raw data. To enable the deployment on multiple machines, each node is implemented as a standalone Java process, and inter-process communication is performed over HTTP. To enable scalable experiments, sensors are implemented as multiple threads of one process, otherwise the RAM overhead for having an HTTP stack deployed for each sensor prevents deploying large topologies. To enable replaying exactly the same sequence of observations, it would have been necessary to synchronize more than 400 threads since the order in which observations are received impacts the obtained result. We were not able

[7]The code is available at https://framagit.org/nseydoux/edr

to ensure such synchronization without reducing the rate at which observations are produced by sensors. All the results are therefore simulated by generating data. Each sensor pushes a random observation to its parent every two seconds, and each simulation is run for five minutes.

6.2.3. Measured results

Two aspects of EDR have been evaluated:

- the validity of our hypothesis, namely that the distribution of rules increases responsiveness,
- the scalability of the proposed approach

To measure the responsiveness of applications enabled by EDR, the **delay between the moment observations are captured** by sensors **and the delivery of the deduction** these observation triggered is measured. Precisely, the delay for the processing of a rule is characterized as the time difference between the moment when the most recent data used in the body of the rule is produced, and the moment when the rule head is received by the application. A dedicated timestamp is associated to each observation once it has been enriched, in order to avoid any impact of the enrichment process on the measure. For instance, if a luminosity observation observed at t_1 and a temperature observation observed at t_2 match $r_{comfort}$ and trigger a deduction that is delivered to the application at t_3, the delivery delay for this particular deduction will be $t_3 - max(t_1, t_2)$. The clocks of all the machines used for the experiment are synchronized to a local server using Network Time Protocol (NTP)[8], in order to ensure a minimal time difference between the different distributed nodes.

Experimental measures showed that, for each simulation, the number of deductions is consistent between centralized and distributed approaches: **there is no knowledge loss when applying EDR$_\mathcal{T}$ under our assumptions** that the Fog topology embeds correlation between data.

In order to analyze closely the cause for the increased delay, the journey of a message has been broken down in discrete timestamped events. The first event related to a message is its construction, either by enrichment of an observation or by achieving a deduction. In order to be propagated in the network, a message might be sent from a node n to another node n', which is identified as two events: the sending from node n, and the reception by node n'.

[8] http://www.ntp.org/

Multiple hops are registered, from the first node responsible for the message creation toward any node that is interested in the message content for deduction. When a message is received by a node n, n starts a reasoning step where it tries to make new deductions based on the rules in its knowledge base. Events are logged at the beginning and at the end of reasoning. In order to detail the delay for each deduction, the journey of the most recent observation leading to the deduction is reconstructed. This journey is built by identifying all consecutive events related to the piece of data leading to the deduction, from its initial enrichment to its processing leading to the deduction, and the delivery of said deduction to the application.

Three components of delay have been identified:

- **Transfer delays**, measured between the emission and the reception of a message. This delay is both impacted by the quality of the network link between two nodes, but also by the processing speed of the recipient: the transfer is considered completed when the recipient declares the reception at the software level, and it is not measured at the network layer. When the message is transferred through multiple hops, the delays are summed.
- **Reasoning delays**, measured between the beginning and the end of a reasoning step. Reasoning delays are summed if the same message is processed with different rules across the topology.
- **Idle delays**, measured between the reception of a message and its processing, or between the reasoning step and the propagation of deductions.

6.3. Use case details

The use case considered for the evaluation is the industry 4.0 scenario introduced in Section §2.1. Table 3 summarizes the rules driving the scenario. All the rules' SHACL representations are available online[9].

As stated in Section §4.3.2, the EDR approach, and by extension EDR$_\mathcal{T}$, is agnostic to the vocabulary used to describe node characteristics. For this use case, only a few properties were needed, as shown in Lst. 4. The description of the node mainly encompasses its neighbors (with `lmu:` properties), its API (with *iotl*:*exposes*), and its characteristics (*edr*:*producesDataOn*). As shown in a dump available online[10], a node's knowledge base includes such a self-

[9] https://w3id.org/laas-iot/edr/iiot/iiot.tar.gz
[10] https://w3id.org/laas-iot/edrt/node_kb_dump.ttl

Rule ID	Rule core
R1: Low Machine Visibility	$Location(?l) \wedge presence(?l,?o_1) \wedge ?o_1 = True \wedge luminosity(?l,?o_2) \wedge ?o_2 < 300L \wedge Machine(?m) \wedge activity(?m,?o_3) \wedge ?o_3 = True \wedge locatedIn(?m,?l) \rightarrow LowMachineVisibility(?m)$
R2: Low Conveyor Visibility	$Location(?l) \wedge presence(?l,?o_1) \wedge ?o_1 = True \wedge luminosity(?l,?o_2) \wedge ?o_2 < 300L \wedge Conveyor(?c) \wedge activity(?c,?o_3) \wedge ?o_3 = True \wedge locatedIn(?c,?l) \rightarrow LowConveyorVisibility(?c)$
R3: No supervision	$Location(?l) \wedge presence(?l,?o_1) \wedge ?o_1 = False \wedge Conveyor(?c) \wedge activity(?c,?o_3) \wedge ?o_3 = True \wedge locatedIn(?c,?l) \wedge SupervisorPost(?s) \wedge supervises(?s,?c) \rightarrow NoSupervision(?c)$
R4: Fire hazard	$Location(?l) \wedge particleLevel(?l,?o_1) \wedge ?o_1 > 25\% \wedge SparkMachine(?m) \wedge activity(?m,?o_3) \wedge ?o_3 = True \wedge locatedIn(?m,?l) \rightarrow Firehazard(?m)$
R5: Cold chain broken	$Location(?l) \wedge temperature(?l,?o_1) \wedge ?o_1 > 6^oC \wedge TemperatureSensitiveMachine(?m) \wedge activity(?m,?o_3) \wedge ?o_3 = True \wedge locatedIn(?l,?m) \rightarrow ColdChainBroken(?m)$
R6: Conveyor too fast	$Conveyor(?c) \wedge Machine(?m) \wedge onConveyor(?m,?c) \wedge machineSpeed(?m,?s_m) \wedge conveyorSpeed(?c,?s_c) \wedge ?s_c > ?s_m \rightarrow ConveyorTooFast(?c)$
R7: Low quality product	$Machine(?m) \wedge productQuality(?m,?o_1) \wedge ?o_1 < 98.5 \rightarrow LowQualityProduct(?m)$

Table 3
Safety and quality rules

Listing 4: Description of a node

```
ex:floor0002 a lmu:Node;
        iotl:exposes ex:floor0002Service ;
        edr:producesDataOn adr:Presence;
        lmu:hasDownstreamNode ex:gallery0006,
                              ex:sensor0003 ;
        lmu:hasUpstreamNode ex:building0001 ;
        lmu:reasoningNode true ;
        ioto:hasId "floor0002" .
```

Listing 5: Exchanged sensor observations

```
ex:sensor0003e3dff9e3_obs
        a ssn:Observation ;
        ssn:observationResult ex:sensor0003e3_out ;
        ssn:observedBy ex:sensor0003 ;
        ssn:observedProperty ex:floor0002presence ;
        edr:receivedAt "2019-0..T..."^^xsd:dateTime.

ex:sensor0003e3dff9e3_out
        a ssn:SensorOutput ;
        ssn:hasValue ex:sensor0003e3dff9e3_val

ex:sensor0003e3dff9e3_val
        a ssn:ObservationValue ;
        dul:hasDataValue "1.0"^^xsd:float .
```

description along with ontologies, similar description for its neighbors, and observed data. Lst. 5 shows a snippet of observed data, mainly described with the legacy version of the SSN ontology, and extensions of the IoT-O [**?**] ontology for elements specific to our experiments. The proposed approach depends on the observation representation, hard-coded in the current implementation, which is why standard vocabularies have been used as much as possible. In future versions, EDR will be updated to be compliant with the updated version of SSN, SSN/-SOSA[11] [**?**]. In the implemented simulation, data is produced in the form of CSV records by sensors that are sent over HTTP to the subscribing node's API. The CSV schema mimics the schema observed in the actual deployment of ADREAM, a smart building producing publicly available data[12]: `chrono,name,value,quality,comment`. The last two headers are ad-hoc to ADREAM, and they are not used in this experiment, and the others are self-explanatory. This raw data is enriched thanks to a SPARQL-Generate query [**?**], a sample of which is shown in Lst. 6[13].

6.4. Scalability of the proposed approach

6.4.1. Simulation topologies

In order to assess the scalability of the proposed strategy for EDR, performances have been measured on three topologies, denoted s0, s1 and s2[14], and collectively as s*, as represented on Fig. 9. All s* topologies mimic the use case architecture presented in Fig. 1, with variations in the number of floors. A floor comprises of two conveyors, each of which supports two machines, with sensors distributed as shown on a JSON blueprint provided online[15], leading to a total

[11] http://www.w3.org/ns/sosa/
[12] https://syndream.laas.fr:8082/
[13] The complete query is available in the source code of EDR: node/queries/data/enrich_data.sparql
[14] Topology representations are available at https://w3id.org/laas-iot/edr/iiot/scala_syndream/clone_f_<0,1,2>.ttl respectively
[15] https://w3id.org/laas-iot/edr/iiot/clone_f_0_blueprint.json

Listing 6: Enrichment query snippet

```
GENERATE {
  # The observation
  ?obsURI a ssn:Observation;
    ssn:observationResult ?sensorOutputURI;
    ssn:observedBy ?sensor_uri;
    edr:receivedAt ?receivedTime.
    # The sensor output
  ?sensorOutputURI a ssn:SensorOutput;
    ssn:hasValue ?obsValURI.
  # ...
}
SOURCE <file://{{ FILE }}> AS ?source
ITERATOR iter:CSV(?source) AS ?obs
WHERE {
  BIND(fn:CSV(?obs, "name" ) AS ?sensor_id )
  # ...
}
```

Table 4
s* topologies

Topology	s0	s1	s2
Nodes	31	61	91

Fig. 9. Simulation topology s*

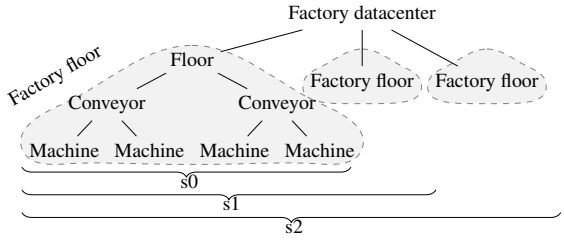


Table 5
Machine hosts for scalability experiments

Virtual node	Datacenter	Floor	Conveyor	Machine
Physical host	Server	Raspberry Pi	Server	Laptop

of 30 nodes (including both reasoning nodes and sensors). The rules described in Section §6.3 are used. The number of nodes is increased by duplicating floors: s0 has one, s1 two, and s2 three floors, for a total number of respectively 31, 61 and 91 nodes (as summarized on Tab. 4). Fig. 10 shows results for centralized approaches, and Fig. 11 for distributed reasoning, both showing single-host and multi-host execution.

6.4.2. Results

Due to scaling issues, results are separated in several figures:

- Results for centralized deduction delivery mechanisms (*i.e.* CIR and CDR) are shown on Fig. 10a for single-host execution, and on Fig. 11a for multi-host execution.
- Results for distributed deduction delivery mechanisms (*i.e.* CIP, CDP and ADP), are shown on Fig. 10b for single-host execution, and on Fig. 12a and 12b for multi-host execution.

The gain in scalability provided by the decentralized approaches appears in the results. In topology s0, the discrepancy between delivery delay for distributed and centralized reasoning approaches is reduced, especially in the single-host execution setting, with a median around 0.65s for CIR and CDR, and 0.065s for CDP, CIP and ADP.

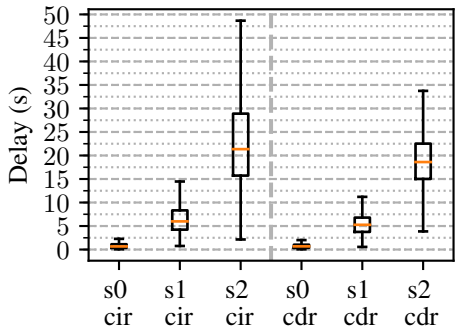
However, in topologies s1 and s2, the gap between centralized and distributed approaches increases dramatically. The deduction time is multiplied by more than 20 from s0 to s2, while the relative share of reasoning time contributing to the delay decreases, as shown on Fig. 13. The transit times are those which increase relatively the most, which denotes a network overflow over a computing saturation on the centralized reasoning node.

A delay increase is also observed for distributed delivery strategies in the single-host execution environment, but it is much smaller, as seen on Fig. 10b. In the multi-host execution environment, there is a performance difference between direct and indirect delivery mechanisms. Even though overall the increase in the number of nodes has little impact on the measured delays, the delays measured in the CIP configurations are much longer than in CDP or ADP.

An explanation for this observation is the fact that, due to their location, the Raspberry Pis are a bottleneck for communication only in this configuration. In CIP, they must both forward observations and deductions towards a Cloud node, as well as performing reasoning, while they only have to process rules with the CDP and ADP strategies. This conclusion is also strengthened by the fact that, if the Raspberry Pis 3 are replaced by Raspberry Pis 2, which have a lower computing power, that same profile is observed, with longer delays, as seen on Fig. 12c for CIP for instance. On Fig. 13, among the three decentralized delivery mechanisms, CIP has the shortest relative transfer time dedicated to reasoning. This is coherent with the fact that more deductions are forwarded by the constrained nodes rather than deduced directly by it, since it is at

Fig. 10. Scalability measures, single-host execution

(a) Centralized reasoning



(b) Distributed reasoning

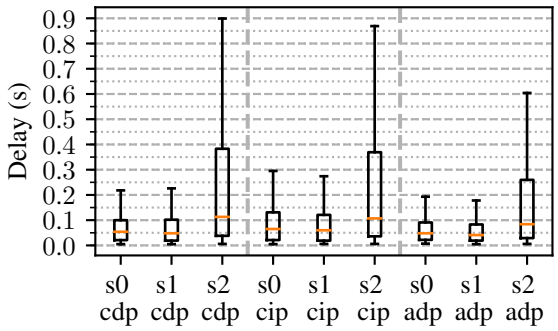
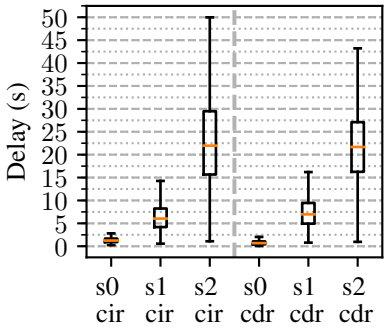


Fig. 11. Scalability measures, centralized reasoning

(a) Multi-host execution



depth 1 in the topology, and it is only connected to a few sensors compared to conveyor or machine nodes.

Approaches promoting direct communication, *i.e.* CDR and CDP, perform better than their indirect counterparts, respectively CIR, CIP. This is an expected result, as direct communication reduces the number of hops required for a message (be it an observation or a deduction) to reach its target.

A trend that can be observed in the breakout is the increase of the share of transfer time in centralized strategies compared to decentralized ones. An explanation for this phenomenon is the saturation of the network link, combined to an overhead on the central node induced by the necessity to perform all the reasoning. The central node has less CPU time available to declare reception of messages, and therefore the time between the emission event and the reception event is increased. Overall, the limited increase of delays and the balance of the delays breakdown in the distributed settings support our claim that EDR$_{\mathcal{T}}$ is a scalable approach to rule-base reasoning based on semantic Fog computing.

Table 6
Machines hosts for distribution experiments

Virtual node	Datacenter	Floor	Conveyor	Machine
Physical host	Server	Laptop	Raspberry Pi	Server

6.5. *Impact of distribution on responsiveness*

6.5.1. *Simulation topology*

To measure how distribution impacts responsiveness, four topologies were distinguished, labeled d1 to d4 and further on simply denoted d*. Each of these topologies is composed of 42 identical nodes, and processes data according to four rules, r1 to r4. The difference between the four d* topologies is the location of sensors, as depicted in Fig. 15. Sensors producing data of the type ρ_1 are directly attached to the top node in d1, while they are attached to its children in d2. Since $body_t(r1) = \{\rho_1, \rho_4\}$, $r1$ is applied at a maximum depth of 1 in d1, but is propagated to nodes of depth 2 in d2, hence a "more decentralized" execution is performed in d2 than in d1. Rule execution depths are given in Tab. 7: in d4, all sensors are connected to leaf nodes, and the distribution is maximal.

To assess the impact of distribution, the same sensors are deployed from topology d0 to d4, but they are not situated at the same level, enabling the control of the level at which rules are processed. Sensors are situated in d* topologies so that the rules are processed at the depths depicted in Tab. 7. The simulation topology is composed of 42 nodes in total (including sensors), hosted on the physical machines as detailed on Tab. 6. Fig. 16 displays results for single-host approaches, and Fig. 17 for multi-host approaches, both showing centralized and distributed reasoning.

Fig. 12. Scalability measures, decentralized reasoning

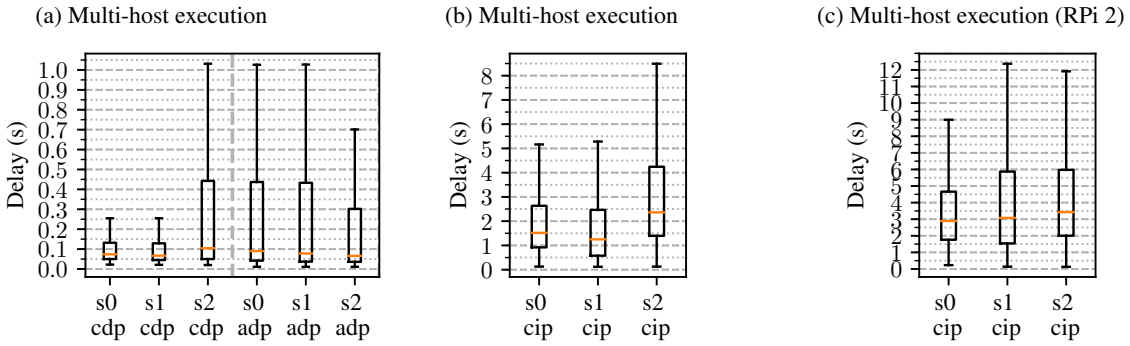


Fig. 13. Breakout of delays (normalized, multi-host execution)

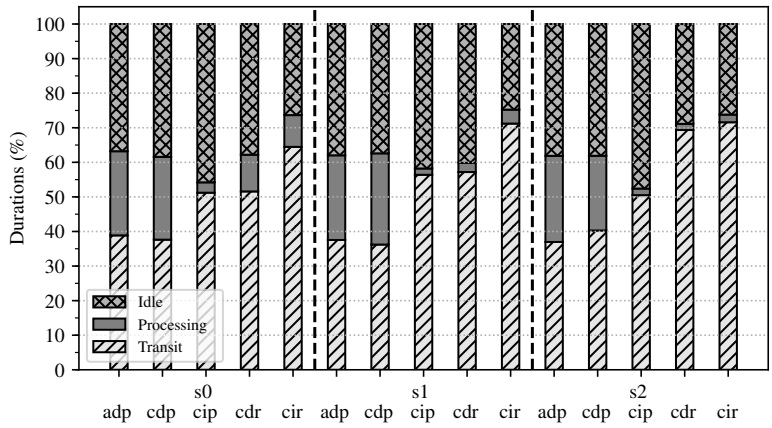


Fig. 14. Reference topology for d*

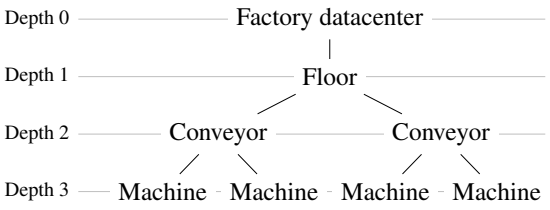
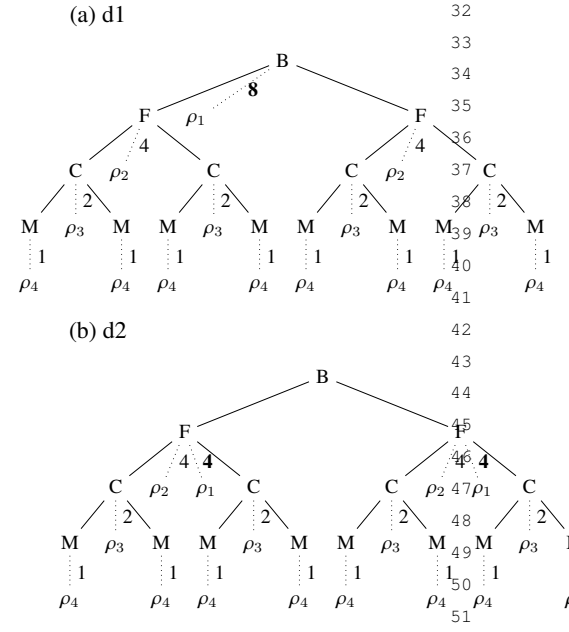


Table 7
Depth of rule processing for d*

	R1	R2	R3	R4	R5	R6	R7
d0	0	0	0	0	0	0	0
d1	0	1	0	1	1	0	0
d2	1	1	0	1	1	0	0
d3	1	1	0	3	3	1	3
d4	3	2	2	3	3	2	3

Fig. 15. d* topologies



6.5.2. *Results*

With the centralized reasoning delivery mechanisms, there is little impact of the distribution on per-

Fig. 16. Distribution experiments, single-host execution

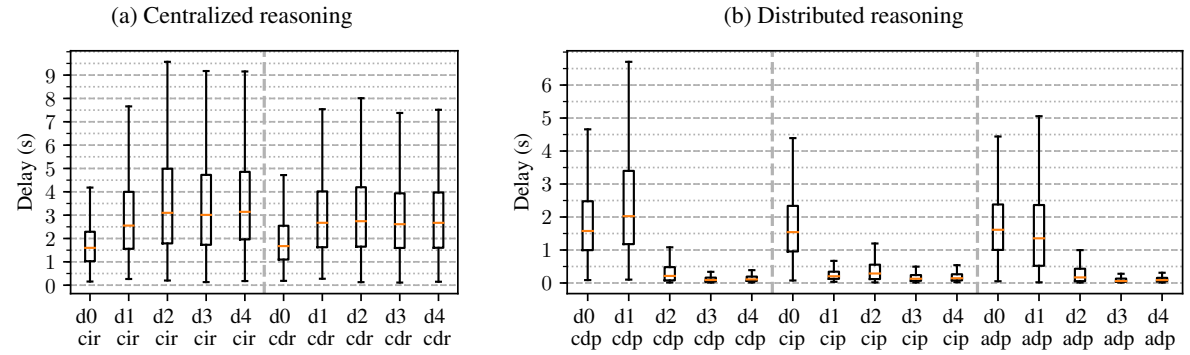
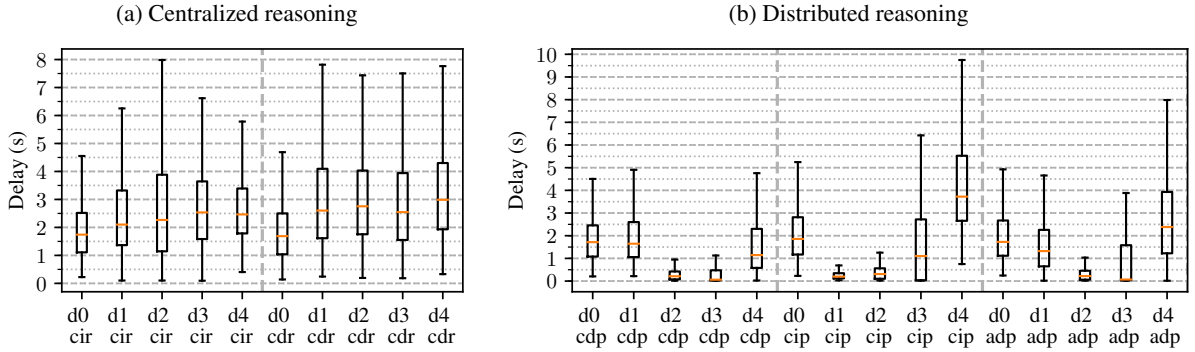


Fig. 17. Distribution experiments, multi-host execution



formances as seen on Fig. 16a. The best performances are measured in the most centralized topology, d0, when the sensors are directly connected to the reasoning node, thus minimizing the transit time, as shown on Fig. 16a and Fig. 17a. Moreover, for this completely centralized topology, the delays measured with the decentralized delivery mechanisms (CDP, CIP, ADP) are comparable to the centralized ones (CIR, CDR), which is an expected result: since all the sensors are connected to a single node, there is no difference between rule deployments. It should also be noted that there are no significant differences between the centralized and decentralized executions. Since all reasoning, which is the most computing-intense process of the simulation, is located in both cases on the most powerful node, it is also an observation consistent with our expectations.

For the decentralized delivery mechanisms, where rules are propagated into the network according to the EDR$_{\mathcal{T}}$ technique, the distribution has indeed an impact on deduction delivery delay, seen on Fig. 16b. In the single-host execution environment (Fig. 16b), where all the nodes have comparable capabilities, there is a correlation between the depth at which rules can be executed (denoting a greater distribution of processing), and the delivery delay decreases. In this case, each node takes charge of an increasing share of the reasoning, leading to a relative decrease of the idle time compared to the reasoning time as seen on Fig. 18.

However, comparing Fig. 16 and 17 shows a discrepancy between the simulation in a single-host and a multi-host-host environment, the latter actually including constrained nodes. For ADP and CIP on Fig. 17b, at the d3 topology, the third and fourth quartiles show an increase in the delays. The median delay is compliant with the expected decreasing trend for ADP, but it begins increasing for CIP. For the d4 topology on Fig. 17b, where the distribution is maximal, there is an significant increase of delays for all decentralized delivery mechanisms, exceeding the delays measured even for d0. This is discussed in details in Section §6.6

6.6. *Discussion*

When increasing the distribution of rule execution in the multi-host experimentation environment, a degradation of the performances is observed. An explana-

Fig. 18. Distribution experiments delays breakout (single-host execution)

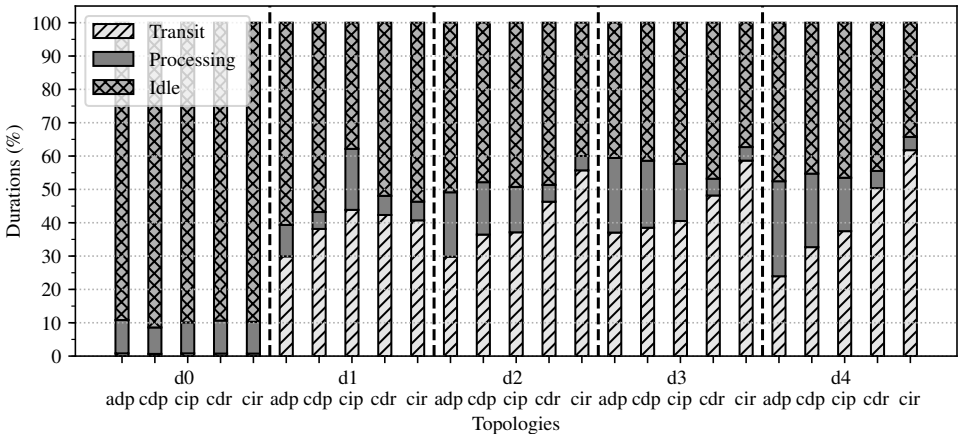
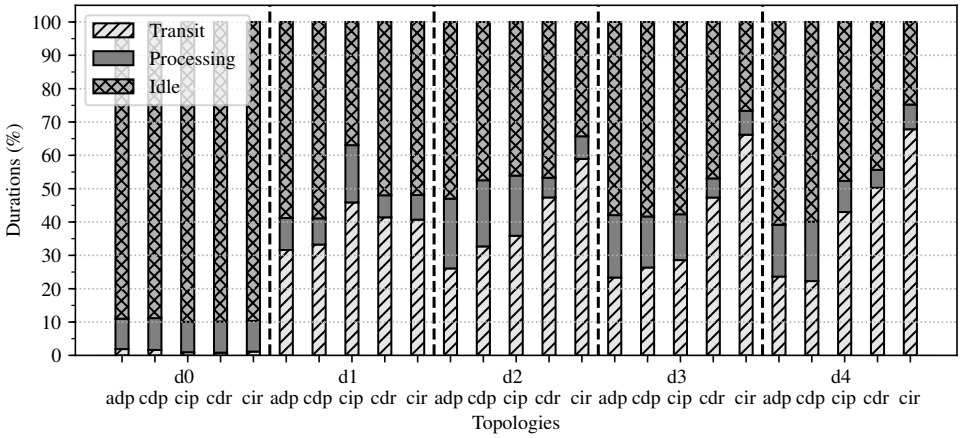


Fig. 19. Distribution experiments delays breakout (multi-host execution)



tion for this phenomenon is the saturation of the Fog node beyond a certain work load, the tipping point being crossed around d3 (see Fig. 17b). Rules executed deeper are processed by constrained Fog nodes, and beyond a certain load, the benefits of the distribution are compensated by the limitations of their processing capabilities.

The progressive relative increase of idle time when increasing distribution, seen when comparing d3 an d4 on Fig. 18 and Fig. 19, supports this hypothesis. To this regard, the EDR$_{\mathcal{T}}$ technique has a naive approach, where the capabilities of the Fog nodes are not considered in the deployment process. The results obtained are encouraging, especially in terms of scalability, and moreover the proposed experimentation aimed at creating extreme conditions, by distributing the rules as much as possible. The topology obtained is not necessarily an accurate reflection of what would be deployed in a real-world application, and it is designed to show a trend rather than to be applied directly.

The technological choices made for the implementation of EDR$_{\mathcal{T}}$ are also factors to be considered in the observed results. Overall, EDR$_{\mathcal{T}}$ is still a proof of concept, and some choices in the implementation should be reconsidered for performance:

- The HTTP framework used (Jersey[16]) has been chosen for convenience for the flexibility of development it allows, but it adds a certain overhead in the memory print and execution time which is not negligible in a constrained environment.
- The SHACL engine used in our experiments is described by its creators as "not really optimized for performance, just for correctness"[17]. It is possible that in the future, better performances will

[16] https://jersey.github.io/
[17] https://github.com/TopQuadrant/shacl

be reached by sheer improvement of the SHACL engine. This engine was chosen because, to the best of our knowledge, it was the only Jena-compatible SHACL implementation at the time of implementation.
– Knowledge is exchanged between nodes serialized in RDF Turtle. Other more compact RDF serializations exist [**?**], and switching to such a format would reduce the communication overhead when messages are exchanged.

Moreover, due to technical constraints, the experiments we conducted could not be performed at a large scale on constrained nodes. This introduces a bias in the measured results, since the simulated nodes ran on much more powerful machines than the Fog nodes should be. We are aware of this bias, and the experiments are designed in such way that it has as little impact as possible. For future experiments, we intend to set up a network of virtual machines, emulating the actual capabilities of physical nodes, rather than mere processes.

7. Conclusion and future work

In this paper, we proposed EDR, a generic approach for dynamically distributed rule-based reasoning in a Cloud-Fog IoT architecture. In existing approaches to rule-based reasoning for the SWoT, computation is often performed on Cloud nodes only, potentially leading to a centralized bottleneck, and, by design, creating network communication overhead. In order to tackle these issues, decentralized approaches are proposed in the literature, taking advantage of the Fog computing paradigm. In such cases, computation is disseminated among Fog nodes in order to be brought closer to the IoT devices producing the data. However, these distributed reasoning approaches do not discuss rule placement: it is static, either computed at design time, or all the nodes execute the same set of rules.

With EDR, the contributions described in this paper, address these shortcomings by leveraging the complementarity between Cloud and Fog computing, in order to associate remote powerful nodes providing stability, and local, limited, opportunistically available computing resources. EDR is a generic approach to dynamically distributed rule-based reasoning, based on modular SHACL rules. The execution by Fog nodes of **core EDR functionalities** is **controlled via a dedicated vocabulary** describing knowledge in each node's KB. This vocabulary is used by rule modules to **implement deployment strategies** enabling the propagation of rules neighbor-to-neighbor across the Fog tier of the Cloud-Fog-Device pattern. Rule **deployment strategies aim at optimizing rule placement for customizable criteria**, such as response time or energy consumption, based on the knowledge stored in each node's KB. Such knowledge includes a description of its neighbors, the current state of the environment based on sensor observations, and background knowledge. Overall, EDR enables, in a purely **decentralized and emergent** manner, the **deployment of rules**, the **propagation of data** and the **delivery of deductions** inferred when applying the rules once they have been deployed. In order to enforce its genericity, EDR itself is made agnostic to individual deployment strategies. It has to be refined by injecting **rules embedding their own deployment strategy**, selected according to application-level requirements. The obtained genericity enables the implementation of several policies, however it requires from the developing team a full SWoT expertise, from the IoT to the SW. We hope that future adoption of the SWoT will support the generalization of such expertise.

To show the interest of our contribution, we proposed EDR$_{\mathcal{T}}$, an EDR refinement implementing a deployment strategy dedicated to **reducing delays** for transmitting deductions to applications. EDR$_{\mathcal{T}}$ aims at deploying rules on Fog nodes as close as possible to sensors, while avoiding unnecessary computation. Rules are thus propagated toward sensors producing the **type of data** they consume, **as deep as possible** in the topology. The propagation stops when the rule is deployed on the Fog node which is the closest common ancestor to these sensors in the topology. To enforce the locality of decisions, node characteristics are announced through the network thanks to a proxying mechanism, where data productions and consumptions are propagated.

The genericity and the dynamicity of the EDR approach are achieved by design, while its scalability and the improvement brought by distribution for responsiveness have been measured through experimentation. A simulated smart factory use case has been considered, executed on a powerful server or distributed across constrained nodes. Decentralized delivery mechanisms outperform centralized ones: Quality of Service (QoS) is less degraded when the number of nodes increase in a distributed reasoning setting. Enabling a more widespread distribution of rules by modifying sensor deployment does not improve QoS with

a centralized delivery mechanism. The complementarity of Fog and Cloud paradigms is also supported by the results of our approach: there is an improvement of performances even in cases where deductions are forwarded to a Cloud node, and not directly to applications, compared to a centralized reasoning approach. Therefore, unloading the Cloud infrastructure by performing semantic Fog computing, while considering the Cloud node both as a computation resource and as a stable Web endpoint for applications enables scalable deployments for the SWoT.

However, not considering the resources available in the Fog showed limitations, and in future work we intend to develop distribution strategies able to perform load balancing between Cloud and Fog nodes based on node capabilities. The impact of the changes in the underlying network on the deployment is not evaluated in this paper. The dynamicity of EDR is shown by construction, but future work will include a detailed evaluation of the performances of this adaptation mechanism.

The genericity of the EDR approach enables such extensions to be developed without modifying the core algorithm. Likewise, future work will include the development of a privacy-aware deployment strategy for EDR. In the strategy implemented by $\text{EDR}_{\mathcal{T}}$, a complete cooperation is assumed between nodes, and there are no guarantees regarding the scope of data exchange. However, IoT data includes private elements, that should only be shared with trusted third-parties. The distributed nature of EDR fosters a paradigm shift: data producers can become data owners, and remain in control. Instead of sending their data to service providers, data owners are provided with rules, and only reveal to remote node part of their data. In the past years, multiple security breaches have been revealed, so enabling users to regain control over their data might restore the trust users need to have regarding the systems that are deployed in their environment. Distributing reasoning driven by a privacy-aware strategy would be a first step towards safer, more user-friendly IoT systems.

Appendix A. Namespaces

Prefix	Namespace
ssn:	http://purl.oclc.org/NET/ssnx/ssn
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
owl:	http://www.w3.org/2002/07/owl#
lmu:	https://w3id.org/laas-iot/lmu#
iotl:	http://iot.ee.surrey.ac.uk/fiware/ontologies/iot-lite#
ex:	http://example.com/ns#
edr:	https://w3id.org/laas-iot/edr#
edrt:	https://w3id.org/laas-iot/edrt#
dul:	http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#
adr:	https://w3id.org/laas-iot/adream#

Table 8
Namespaces referenced in this paper, and the associated prefixes

Appendix. References

[1] T. Berners-Lee, J. Hendler and O. Lasilla, The Semantic Web, *Scientific American* **284**(5) (2001), 34–43. doi:10.1038/scientificamerican0501-34.

[2] D. Pfisterer, K. Romer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, A. Kröller, M. Pagel, M. Hauswirth, M. Karnstedt, M. Leggieri, A. Passant and R. Richardson, SPITFIRE: toward a semantic web of things, *IEEE Communications Magazine* **49**(11) (2011), 40–48. doi:10.1109/MCOM.2011.6069708.

[3] A. Gyrard, M. Serrano, J.B. Jares, S.K. Datta and M.I. Ali, Sensor-based Linked Open Rules (S-LOR): An Automated Rule Discovery Approach for IoT Applications and its use in Smart Cities, in: *Proceedings of the 26th International Conference on World Wide Web Companion*, International World Wide Web Conferences Steering Committee, 2017, pp. 1153–1159. ISBN 978-1-4503-4914-7. doi:10.1145/3041021.3054716.

[4] S. Wang, J. Wan, D. Li and C. Liu, Knowledge reasoning with semantic data for real-time data processing in smart factory, *Sensors (Switzerland)* **18**(2) (2018), 1–10. doi:10.3390/s18020471.

[5] P. Mell and T. Grance, The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology, *National Institute of Standards and Technology, Information Technology Laboratory* **145** (2011), 7. ISBN 1047-6210. doi:10.1136/emj.2010.096966.

[6] F. Bonomi, R. Milito, J. Zhu and S. Addepalli, Fog Computing and Its Role in the Internet of Things, in: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, ACM Press, New York, New York, USA, 2012, pp. 13–16. ISBN 978-1-4503-1519-7. doi:10.1145/2342509.2342513.

[7] P. Patel, M. Intizar Ali and A. Sheth, On Using the Intelligent Edge for IoT Analytics, *IEEE Intelligent Systems* **32**(5) (2017), 64–69. doi:10.1109/MIS.2017.3711653.

[8] Y. Sahni, J. Cao, S. Zhang and L. Yang, Edge Mesh: A New Paradigm to Enable Distributed Intelligence in Internet of Things, *IEEE Access* **5** (2017), 16441–16458. ISBN 978-1-5090-6517-2. doi:10.1109/ACCESS.2017.2739804.

[9] F.B. Charrada and S. Tata, An Efficient Algorithm for the Bursting of Service-Based Applications in Hybrid Clouds, *IEEE Transactions on Services Computing* **9**(3) (2016), 357–367. doi:10.1109/TSC.2015.2396076.

[10] N. Seydoux, K. Drira, N. Hernandez and T. Monteil, A Distributed Scalable Approach for Rule Processing: Computing in the Fog for the SWoT, in: *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, IEEE, 2018, pp. 112–119.

[11] N. Seydoux, K. Drira, N. Hernandez and T. Monteil, Towards Cooperative Semantic Computing: a Distributed Reasoning approach for Fog-enabled SWoT, in: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, Springer, 2018, pp. 407–425.

[12] A. Khandelwal, I. Jacobi and L. Kagal, Linked rules: Principles for rule reuse on the web, in: *International Conference on Web Reasoning and Rule Systems*, Vol. 6902 LNCS, Springer, pp. 108–123.

[13] A.I. Maarala, X. Su and J. Riekki, Semantic reasoning for context-aware Internet of Things applications, *IEEE Internet of Things Journal* **4**(2) (2016), 461–473.

[14] X. Su, P. Li, J. Riekki, X. Liu, J. Kiljander, J.-P. Soininen, C. Prehofer, H. Flores and Y. Li, Distribution of Semantic Reasoning on the Edge of Internet of Things, in: *IEEE UbiComp*, 2018, p. 79. ISBN 9781450348140.

[15] H. Boley, M. Kifer, P.-L. Pătrânjan and A. Polleres, Rule interchange on the web, in: *Reasoning Web International Summer School*, Springer, 2007, pp. 269–309.

[16] A. Sheth, C. Henson and S.S. Sahoo, Semantic Sensor Web, in: *IEEE Internet Computing*, Vol. 12, 2008, pp. 78–83. ISSN 1089-7801. ISBN 1089-7801 VO - 12. doi:10.1109/MIC.2008.87.

[17] O.B. Sezer, E. Dogdu and A.M. Ozbayoglu, Context Aware Computing, Learning and Big Data in Internet of Things: A Survey, *IEEE Internet of Things Journal* **5**(1) (2018), 1–1. doi:10.1109/JIOT.2017.2773600.

[18] Z. Li, C.H. Chu, W. Yao and R.a. Behr, Ontology-driven event detection and indexing in smart spaces, in: *Proceedings - 2010 IEEE 4th International Conference on Semantic Computing, ICSC 2010*, 2010, pp. 285–292. ISBN 9780769541549. doi:10.1109/ICSC.2010.63.

[19] Y. Sun and A.J. Jara, An extensible and active semantic model of information organizing for the Internet of Things, *Personal and Ubiquitous Computing* **18**(8) (2014). ISBN 1617-4909. doi:10.1007/s00779-014-0786-z.

[20] G. Xu, Y. Cao, Y. Ren, X. Li and Z. Feng, Network Security Situation Awareness Based on Semantic Ontology and User-Defined Rules for Internet of Things, *IEEE Access* **5** (2017), 21046–21056.

[21] I.B. Rodriguez, J. Lacouture and K. Drira, Semantic Driven Self-Adaptation of Communications Applied to ERCMS, in: *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, IEEE, 2010, pp. 1292–1299. ISBN 978-1-4244-6695-5. doi:10.1109/AINA.2010.158.

[22] Y. Evchina, J. Puttonen, A. Dvoryanchikova and J.L.M. Lastra, Context-aware knowledge-based middleware for selective information delivery in data-intensive monitoring systems, *Engineering Applications of Artificial Intelligence* **43** (2015), 111–126. doi:10.1016/j.engappai.2015.04.008.

[23] P. Kasnesis, C.Z. Patrikakis and I.S. Venieris, Collective domotic intelligence through dynamic injection of semantic rules, in: *IEEE International Conference on Communications*, Vol. 2015-Septe, 2015, pp. 592–597. ISSN 15503607. ISBN 9781467364324. doi:10.1109/ICC.2015.7248386.

[24] P. Lillo, L. Mainetti, V. Mighali, L. Patrono and P. Rametta, A Novel Rule-based Semantic Architecture for IoT Building Automation Systems, in: *International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Vol. 12, IEEE, 2015, pp. 124–131. ISSN 18456421. ISBN 978-9-5329-0056-9. doi:10.1109/SOFTCOM.2015.7314063.

[25] D. Hussein, S.N. Han, G.M. Lee, N. Crespi and E. Bertin, Towards a dynamic discovery of smart services in the social internet of things, *Computers & Electrical Engineering* (2016). doi:10.1016/j.compeleceng.2016.12.008.

[26] W. Van Woensel and S.S.R. Abidi, Optimizing Semantic Reasoning on Memory-Constrained Platforms Using the RETE Algorithm, in: *ESWC*, Vol. 10843 LNCS, 2018, pp. 682–696. ISSN 16113349. ISBN 9783319934167.

[27] P. Desai, A. Sheth and P. Anantharam, Semantic gateway as a service architecture for IoT interoperability, in: *2015 IEEE International Conference on Mobile Services*, IEEE, 2015, pp. 313–319.

[28] Y.H. Lee and S. Nair, A Smart Gateway Framework for IOT Services, *Proceedings - 2016 IEEE International Conference on Internet of Things; IEEE Green Computing and Communications; IEEE Cyber, Physical, and Social Computing; IEEE Smart Data, iThings-GreenCom-CPSCom-Smart Data 2016* (2016), 107–114. ISBN 9781509058808. doi:10.1109/iThings-GreenCom-CPSCom-SmartData.2016.44.

[29] C.E. Kaed, I. Khan, A.V.D. Berg, H. Hossayni and C. Saint-Marcel, SRE: Semantic Rules Engine for the Industrial Internet-Of-Things Gateways, *IEEE Trans. Industrial Informatics* **14**(2) (2018), 715–724. doi:10.1109/TII.2017.2769001.

[30] I. Chatzigiannakis, H. Hasemann, M. Karnstedt, O. Kleine, A. Kröller, M. Leggieri, D. Pfisterer, K. Römer and C. Truong, True self-configuration for the IoT, in: *2012 3rd IEEE International Conference on the Internet of Things*, IEEE, 2012, pp. 9–15.

[31] H. Hasemann, A. Kröller and M. Pagel, RDF provisioning for the internet of things, in: *Proceedings of 2012 International Conference on the Internet of Things, IOT 2012*, IEEE, 2012, pp. 143–150. ISBN 9781467313469. doi:10.1109/IOT.2012.6402316.

[32] M. Taneja and A. Davy, Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm, in: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management*, IEEE, 2017, pp. 1222–1228. ISBN 978-3-901882-89-0. doi:10.23919/INM.2017.7987464.

[33] C.E. Kaed, I. Khan, A. Van Den Berg, H. Hossayni and C. Saint-Marcel, SRE : Semantic Rules Engine For the Industrial Internet- Of-Things Gateways, *IEEE Transactions on Industrial Informatics* **14**(2) (2018), 715–724. doi:10.1109/TII.2017.2769001.

[34] C.E. Kaed, I. Khan, H. Hossayni and P. Nappey, SQenloT: Semantic query engine for industrial Internet-of-Things gateways, *2016 IEEE 3rd World Forum on Internet of Things, WF-IoT 2016* (2016), 204–209. ISBN 9781509041305. doi:10.1109/WF-IoT.2016.7845468.

[35] A. Zanella, N. Bui, A. Castellani, L. Vangelista and M. Zorzi, Internet of Things for Smart Cities, *IEEE Internet of Things Journal* **1**(1) (2014), 22–32. doi:10.1109/JIOT.2014.2306328.

[36] M. Ben-Alaya, S. Medjiah, T. Monteil and K. Drira, Toward semantic interoperability in oneM2M architecture, *IEEE Communications Magazine* **53**(12) (2015), 35–41. doi:10.1109/MCOM.2015.7355582.

[37] I. Szilagyi and P. Wira, Ontologies and Semantic Web for the Internet of Things - a survey, in: *IECON*, IEEE, 2016.

[38] N. Seydoux, K. Drira, N. Hernandez and T. Monteil, IoT-O, a core-domain IoT ontology to represent connected devices networks, in: *European Knowledge Acquisition Workshop*, Springer, 2016, pp. 561–576.

[39] N. Seydoux, K. Drira, N. Hernandez and T. Monteil, Capturing the contributions of the semantic web to the IoT: a unifying vision (extended abstract), *Semantic Web technologies for the Internet of Things* (2017).

[40] S. Nikoli, V. Penca and Z. Konjovi, Semantic Web Based Architecture for Managing Hardware Heterogeneity in Wireless Sensor Network, in: *International Journal of Computer Science and Applications*, Vol. 8, 2011, pp. 38–58. ISBN 9781450301480.

[41] C. Perera, A. Zaslavsky, P. Christen and D. Georgakopoulos, Context aware computing for the internet of things: A survey, *IEEE Communications Surveys and Tutorials* **16**(1) (2014), 414–454. ISBN 1553-877X VO - PP. doi:10.1109/SURV.2013.042313.00197.

[42] K. Janowicz, A. Haller, S.J.D. Cox, D. Le Phuoc and M. Lefrançois, SOSA: A lightweight ontology for sensors, observations, samples, and actuators, *Journal of Web Semantics* **56** (2019), 1–10. doi:10.1016/j.websem.2018.06.003. http://www.sciencedirect.com/science/article/pii/S1570826818300295.

[43] M. Lefrançois, A. Zimmermann and N. Bakerally, A SPARQL Extension for Generating RDF from Heterogeneous Formats, in: *The Semantic Web*, E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler and O. Hartig, eds, Lecture Notes in Computer Science, Springer International Publishing, 2017, pp. 35–50. ISBN 978-3-319-58068-5.

[44] X. Su, J. Riekki, J.K. Nurminen, J. Nieminen and M. Koskimies, Adding semantics to internet of things, *Concurrency and Computation: Practice and Experience* **27**(8) (2015), 1844–1860. doi:10.1002/cpe.3203.