



Article

SOLIOT—Decentralized Data Control and Interactions for IoT

Sebastian R. Bader ^{1,*}  and Maria Maleshkova ^{2,*} ¹ Fraunhofer IAIS, Schloss Birlinghoven, 53757 Sankt Augustin, Germany² Institute for Computer Science, University of Bonn, Endenicher Allee 19a, 53115 Bonn, Germany

* Correspondence: sebastian.bader@iais.fraunhofer.de (S.R.B.); maleshkova@cs.uni-bonn.de (M.M.)

Received: 30 April 2020; Accepted: 8 June 2020; Published: 16 June 2020



Abstract: The digital revolution affects every aspect of society and economy. In particular, the manufacturing industry faces a new age of production processes and connected collaboration. The underlying ideas and concepts, often also framed as a new “Internet of Things”, transfer IT technologies to the shop floor, entailing major challenges regarding the heterogeneity of the domain. On the other hand, web technologies have already proven their value in distributed settings. SOLID (derived from “social linked data”) is a recent approach to decentralize data control and standardize interactions for social applications in the web. Extending this approach towards industrial applications has the potential to bridge the gap between the World Wide Web and local manufacturing environments. This paper proposes SOLIOT—a combination of lightweight industrial protocols with the integration and data control provided by SOLID. An in-depth requirement analysis examines the potential but also current limitations of the approach. The conceptual capabilities are outlined, compared and extended for the IoT protocols CoAP and MQTT. The feasibility of the approach is illustrated through an open-source implementation, which is evaluated in a virtual test bed and a detailed analysis of the proposed components.

Keywords: internet of things; digital twin; linked data; cloud systems

1. Introduction

Technologies and practices from the Web increasingly find their way into industrial manufacturing processes. The originally distinctly separated Operation Technology merges with the decentralized technology stack of the Web. One reason for this is the assimilation of requirements: large-scale distributed systems must provide interoperability but also implement protection of data and systems by design. As such, the digitizing of the industrial manufacturing domain can benefit from the latest developments of the Web community.

The ongoing digitization promises a new age of interconnection and interoperability of systems, devices and actors. Motivated by the potential, uncountable procedures and practices have been developed to connect computers and devices, to share data and to enable composed processes. Since several years, the paradigm of the Internet of Things (IoT) has gained more and more popularity, aiming for integrating nearly any component or ‘thing’ into Internet-based networks. The concept of Digital Twins follows the paradigm to merge the physical and virtual representation of such things into one indispensable entity and thereby tracks more and more attention.

However, the created heterogeneity of protocols, interfaces, interaction patterns, data formats, identifiers, etc. have proven themselves as challenging and reoccurring problems, especially in distributed systems and architectures. Therefore, several standardization groups and committees started efforts to establish a common ground. These initiatives originate from the Operations Technology domain but also take place in communities around Web and Cloud technologies. The target

of SOLID [1], for instance, is the human user and its data in the Web. The Digital Twin appears hereby as the combination of the human user with its digital data collection. The overlaps in requirements and faced concerns with Industrial IoT settings are significant and indicate a largely shared problem understanding. However, distinct requirements (normative processes, control assurance, sustainability, resource constraints) prohibit a direct one-to-one transfer from the SOLID specification to an Industrial IoT solution. The industrial scenario does not require the same usability degree but has similar requirements regarding interoperability, system federation, security and data protection (privacy). This paper presents an analysis of the overlapping features and missing requirements, outlines a mapping approach supported with early proof of concepts called SOLIOT (**SOLID** for **IoT**, see Figure 1) and evaluates the prototype through a qualitative and quantitative examination.

Regarding the distributed character of IoT scenarios is crucial for any implementation. This requires clear interaction patterns, interfaces and a shared understanding of the meaning and implications of entities, states, and interactions. While most approaches only target the communication layer and therefore enable syntactic interoperability, the federated interaction between the entities themselves requires a mature and transparent concept of their behavior, interactions, capabilities and authorities. REST is maybe the most successful and well-known pattern in this regard, allowing clean APIs, transparent interactions, and clean expectations on consequences. Extensions such as Linked Data Platform (LDP) for data further develop these ideas. SOLID transforms the patterns to personal profiles with access rules.

In the near future, autonomous devices will more and more coordinate processes independently and therefore need to interact with each other. Web-based approaches might cover a significant number of scenarios but certainly not all (bandwidth, resource constraints, latency requirements). Several tailored IoT protocols can fulfill such needs but lack mature, well-known interaction concepts and scalability across organizations, systems, or in the open Internet.

As for instance Pfrommer et al. [2] state, current approaches are tailored to their target scenario and then adapted afterwards. SOLIOT presents a Digital Twin concept primarily based on the standardized Web of Things data model, the self-descriptive interaction patterns of LDP and the data protection mechanism as developed by the SOLID approach. To the best of our knowledge, no other framework has yet regarded these aspects as the corner stones of a consistent, comprehensive solution instead of unrelated developments. SOLIOT shows how the commonly shared principles driving these technologies actually outline a broadly followed trend towards a distinct set of core principles and how they can be combined to implementable Digital Twins, and which steps are still missing. Several works have already translated the concepts and patterns for Web-based architectures and map them to IoT protocols. This work extends these activities with the following core contributions:

- Analysis and structure of the core IoT requirements
- Examining the Digital Twin characteristics of SOLID
- A mapping and extension of the SOLID stack towards IoT requirements
- Presenting a thereby specified Digital Twin representation, its potential and open gaps

The mapping of SOLID features to two different IoT protocols, the Message Queuing Telemetry Transport (MQTT, [3]) and the Constraint Application Protocol (CoAP, [4]) illustrates the potential of SOLIOT. Both protocols explicitly target different communication patterns. While CoAP follows a request/response process, MQTT is based on publish/subscribe. Both CoAP and MQTT contain only a minimal set of added requirements, which makes them optimal representatives to illustrate the SOLIOT core principles. The novelty of SOLIOT is therefore not provided by the CoAP or MQTT mapping but by the compliant merging of their underlying concepts and the restriction to self-descriptive entities.

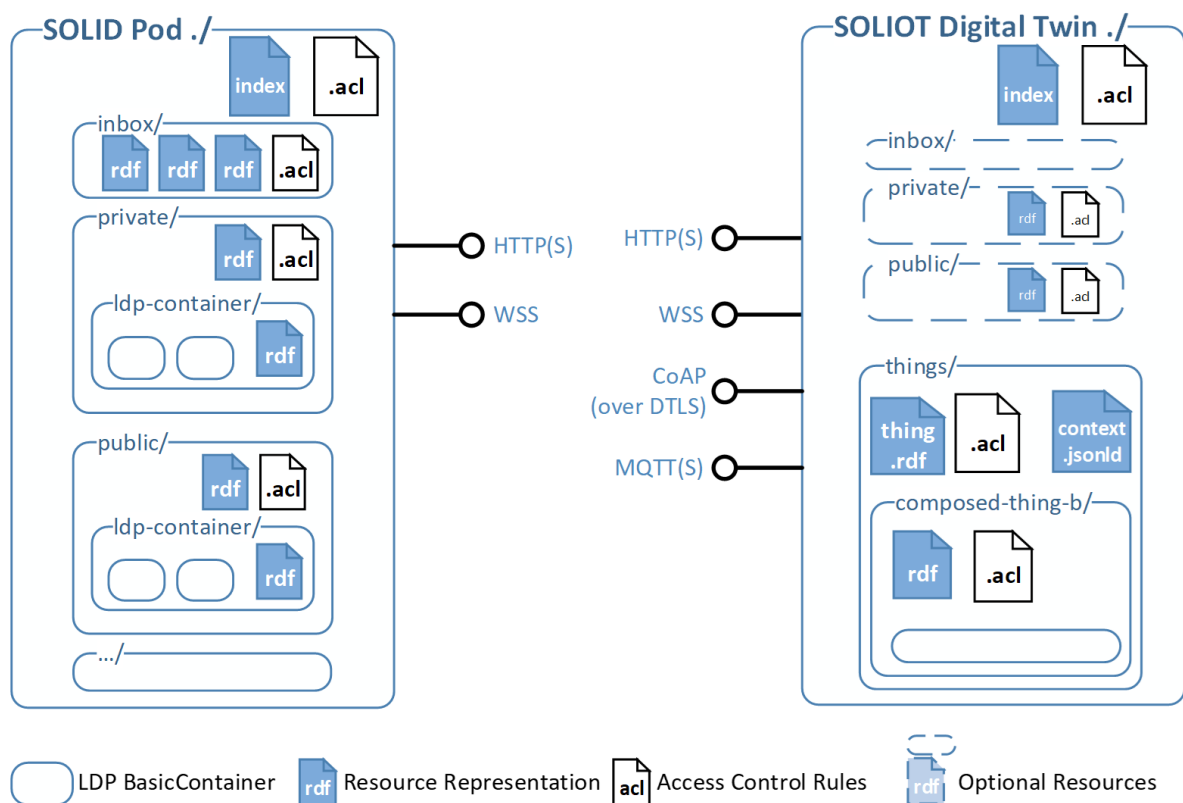


Figure 1. The SOLIOT concept extends the SOLID approach with IoT endpoints and reserved resource representations of WoT Things.

The SOLID reference project in NodeJS is extended with support for both protocols to provide two proof-of-concept developments and general showcases. Up to ten instances of the servers have been started in a federated test bed. Therefore, the scalability of the two approaches is compared to the plain SOLID on HTTP solution. Furthermore, an analysis is applied investigating which of the identified requirements can be fulfilled by the SOLIOT approaches.

The rest of the paper is organized as follows. Section 2 gives an overview of the state of the art and outlines the foundations. Section 3 illustrates the regarded scenario and supplies a running example, followed by an examination of requirements of IoT systems in Section 4. Section 5 introduces the current state of the SOLID specification with respect to these requirements. SOLIOT is presented in Section 6 together with a prototypical implementation (Section 7) and the evaluation of the proposed ideas (Section 8). Section 9 concludes the work by summarizing the contributions, discussing the advantages, limitations and opportunities for future work.

2. State of the Art

This chapter explains the background of the presented concepts and outlines relevant works of the domain. A definition of Digital Twins as the core concept of SOLIOT is applied, followed by discussion of related publications.

2.1. Preliminaries of IoT

In the following, the terminology of a Digital Twin is used to refer to the digital representation of a physical asset. Usual characteristics of a Digital Twin involve visual representations, simulation models, machine-readable descriptions and defined interfaces, virtual representation of physical attributes, bidirectional relations between the physical and the virtual entity, and more. Close or even identically used are terms such as cyber-physical systems, IoT devices, avatars, virtual representations,

asset administration shells etc. While nearly as many definitions of Digital Twins exist as terms used (for instance [5–7]), in this paper mainly the one by Glaessgen and Stargel [8] is followed:

“The Digital Twin is an integrated multiphysics, multiscale, probabilistic simulation of an [as-built vehicle or] system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its corresponding [flying] twin”. Tao et al. extend this understanding stating that “Digital Twin consists of three parts: physical product, virtual product, and connected data that tie the physical and virtual product”. This work especially focuses on the second and third parts—the virtual product and the connected data, and less on the previously strengthened simulation aspects.

2.2. Related Work

The ongoing trend towards digitizing the industrial domain and the thereby created need for standardization has led to the formation of a set of industry consortia. Most relevant among them are the International Internet Consortia with their reference architecture [9] and the German Plattform Industrie 4.0. While the first covers a broader picture across the related domains, the later one focuses on the IoT entity called an Industry 4.0 components and their digital representation, the Administration Asset Shell [10]. Further similar initiatives have been formed, among them the Italian Piano Industria 4.0 and the French Industrie du Futur are closely related to the Asset Shell concept [11]. The International Data Space on the other hand concentrates on the data exchange level and promotes workflows and patterns for usage control even after the sensitive data has left the own network [12].

The embedding of network and Internet-enabled devices into nearly any production-related asset, requiring digital communication between devices from different manufacturers, is definitely not a new development. Machine-to-machine protocols intend to solve this challenge, with the OPC UA stack [13] as the most widely adopted solution. OPC UA defines the interactions, data structures and information models for entities on the shop floor, also often referred to as Operational Technology (OT). AutomationML is further often used to describe the specific logic in OPC UA-driven production workflows as e.g., described by Volz et al. [14].

Especially in the automation domain, communication and control networks relying on OPC UA provide vendor-independent machine-to-machine architectures. The OPC UA protocol stack specifies the data representation, remote interactions with resources and functionalities and its own information model. OPC UA-driven architectures have a strong focus on the operational technology domain, for instance intended for shop floor devices. Malakuti et al. [15] present a layered architecture for Digital Twins based on OPC UA. The interactions are enabled through either OPC UA sockets or REST APIs, similar to the SOLIOT approach. Generally, the interactions close to the asset or devices are executed through OPC UA binary communication and the ones between the Digital Twin and higher-level applications rely on REST over HTTP.

The complete potential of Digital Twins however is the seamless connection throughout the islands and silos, and cross networks, domains and companies. OPC UA networks usually require gateways and proxies, thereby forming bottlenecks and enforcing deep understanding of the protocol’s abilities and restrictions. In addition, as both Perzylo et al. [16] and Schiekofer et al. [17] have mentioned, the lack of formal concept definitions hampers the further integration with higher-level applications.

Earlier approaches to transfer established web technologies to the industrial setting combine service-oriented architectures [18,19]. The method-driven semantic allows the integration through rich service functions using SOAP and the standardized WS* stack. However, the complex requirements and side effects led to redesign of the CBS, less viewed as a collection of method calls and more as a virtual resource with distinct state transitions.

Web agents and representations of avatars have been suggested to by Mrissa et al. [6] as such a resource-oriented extension of CBS with Web capabilities. The one-to-one relation of an Avatar and its cyber-physical object are intended to lift CBS to a homogeneous, Web-driven integration layer. Their interfaces appear through REST APIs and make use of semantic descriptions, both of their

capabilities and the offered and consumed data. The main focus of the Avatar approach is the interoperability challenge of IoT scenarios, with less detailed specifications on data protection and security. Virtual Representations [20] go into a similar direction, with read/write capabilities also on functions and services.

Smart Web Services [21] approach this challenge coming from a Cloud-oriented perspective. The ability to perceive, model, and react dynamically on different context environments allows this model to behave autonomously and reevaluate their own AI-powered decision model. While the flexibility of Smart Web Services may be a key to today's and tomorrow's heterogeneity challenges, the still missing understanding on consequences of such automated decision processes, in particular regarding safety and liability, still prevents their actual implementation.

Several attempts have been made to cover Web technologies in industrial settings. One of them is the W3C Web of Things Working Group proposing an Architecture and Vocabulary [22] for the intersection of the Semantic Web and IoT. One of them, the W3C recommendation for a Web of Things, focuses on the promotion of Web technologies with elaborate semantic meaning. The 'Thing' is closely integrated with conventions and specifications used in the Web. In particular, the discovery of capabilities and resources at runtime through previously uninformed clients is at the heart of the recommendation. That enables a loose coupling of server and clients, thereby allowing a real decentralized and scalable network. Through regarding an IoT network as an open Web environment, the proven security mechanisms from Web applications (encryption, authorization, identities) are introduced into operational communication. The WoT recommendation also proposes the according Thing Description ontology (TD) as the core vocabulary to denote the WoT attributes and properties.

First mappings from the Web of Things community towards IoT protocols have already been created [23]. The CoAP and MQTT bindings in form of RDF vocabularies are however still on the level of examples, and do not represent the full characteristics of these protocols. For instance, the interaction and discovery mechanisms are still missing.

Other vocabularies for IoT applications and devices are for instance the IoT-Lite ontology [24] or FIESTA-IoT [25]. Their focus on formal descriptions and automated inferencing support integrates so-called top-level ontologies and thereby eases the relation cross domains. Well-known and commonly used vocabularies such as RDF, DCTERMS or QUDT are aligned, mapped and extended with IoT-specific concepts. The Lov4IoT catalog [26] collects a broad range of such ontologies and formally described IoT vocabularies.

However, the inherent overhead of the linked-data approaches due to their strict usage of HTTP messages has led to mappings into less expressive but resource saving protocols. An LDP to CoAP translation [27] outlines patterns and solutions to bridge the gap between restricted devices and rich interactions on the application layers. The additional introduction of Edge or Fog layers through dedicated gateways [28,29] can further lift the resource-restricted data sources to the higher-level communication layers.

3. Scenario

The scenario in this section serves as a running example to outline the ideas and approaches for an integrated Usage Control of a Digital Twin in an industrial setting (cf. Figure 2). A *Manufacturer* collects static master data together with sensor observations of an operating device inside a Digital Twin. This Thing communicates through common IoT protocols and reports its observed state to a remote server. The Manufacturer mandates a *Data Analyst* to access the created data and thus to operate its failure prediction algorithm. The prediction results are sent back as a consistently updated attribute. The Manufacturer also is willing to share these insights contained in the Data Analyst's copy (Digital Twin') with its *Business Partner*, as early information on potential breakdowns directly affects their just-in-time supply chain. It is however crucial for the Manufacturer to prohibit any access of other customers of the Data Analyst, as those might be *Competitors*.

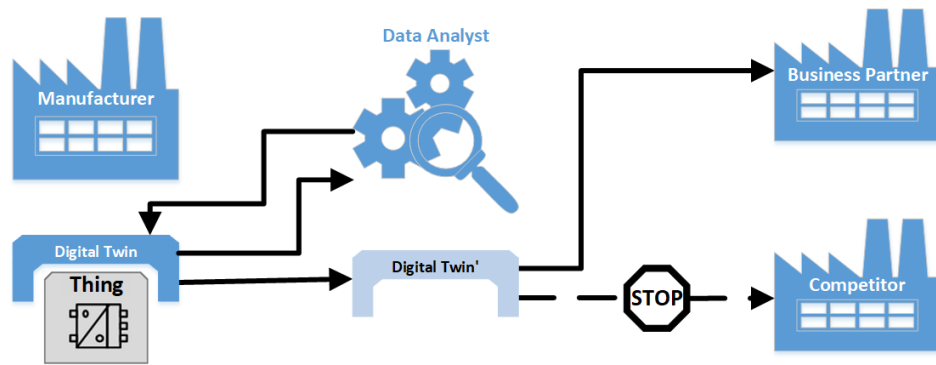


Figure 2. Example collaboration network. The Digital Twin information flows from left to right.

Figure 2 outlines the information flow in this simplified setting. The Thing, enriched with the Thing Description to a Digital Twin, provides static master data and dynamic sensor streams. This interaction is managed by an access control engine deployed directly at the SOLIOT server. The Data Analyst as an intermediary can request the data using the IoT or Web endpoints. It also may copy the complete Digital Twin and move it into its own SOLIOT instance. Please note that this step results in a copy of the Digital Twin representation (Digital Twin'), which is not directly located at the original asset anymore but in the Data Analyst's server.

However, as the Digital Twin' is now located at the Data Analyst, the contained information is not under control of the Manufacturer anymore but still contains its critical data. As such, the Data Analyst needs to evaluate the access requests from both the Manufacturer's Business Partner and the Competitor accordingly. The required instructions and descriptions must be contained in the Digital Twin, and in the Digital Twin'.

4. IoT Requirements

In IoT settings, the asset or *Thing* is at the core of all design decisions. Even though software or services are also regarded and treated as such—to gain a coherent environment—usually physical objects are in the focus. The conversion to Digital Twins is therefore determined primarily regarding the capabilities and characteristics of the asset and less the demands of the consuming applications.

Many publications propose their own requirement analysis or categorization [9,30–32]. Each of them has its justification in their regarded application environment. Still, one can extract a certain set of similar views and approaches. For instance, most frameworks distinguish between functional characteristics and information modelling aspects, promote the importance of security features, and connect a virtual representation with a Thing. However, their variances are still significant and it is usually not possible to directly align the different frameworks. For this paper, the extensive models of Bassi et al. [33] and Kovatsch et al. [22] are most suitable in order to explain the constraints in the example as both models suggest a clear classification of requirements, outline a generic data model and propose protocol bindings based on a defined list of necessary interaction features.

Still, neither the terminology, nor the structure of the underlying system architecture, or the data models are completely congruent. Nevertheless, a combination of both models is feasible and creates a reasonable structure for the further analysis. In the following, the relations of the respective requirements are outlined, using the layered model of Figures 3 and 5 and the similar view of Figure 4 as a simplified reference model.

The four main categories *Functional* and *Non-functional* requirements, *Protocol Bindings* and *Data Models* group the targeted concerns into a basic structure. To increase readability, each stated requirement is identified by its category identifier, for instance 'F' for a functional requirement, followed by a number. This notation is used throughout this paper and is intended to better follow the mapping of the respective requirements with the respective system capabilities.

4.1. Functional Requirements

The functional requirements of an IoT system collect the core functions and capabilities of a system to make it work. Following Bassi et al. [33], this category frames “the system’s runtime Functional Components, including the components’ responsibilities, their default functions, their interfaces, and their primary interactions” (page 133 [33]). In this sense, these aspects are regarded independently of the protocol implementations outlined later, although the distinctions can certainly differ.

Kovatsch et al. [22] specify a set of required principles for this category. Even though some of the mentioned aspects are actually non-functional characteristics (scalability or flexibility), a core set of necessary features can be extracted. Most relevant is certainly *interoperability* (F1) to manage the heterogeneity of devices and enable the exchange of digital data. Kovatsch et al. [22] further distinguish Thing-to-Thing Interaction (F1.1), Thing-to-Gateway (F1.2), Thing-to-Cloud (F1.3), and Web Integration (F1.4), The abilities to *read and interact with resource states* (F2), manage notifications (F3) and invoke operations (F4) are also stated (cf. Figure 3).

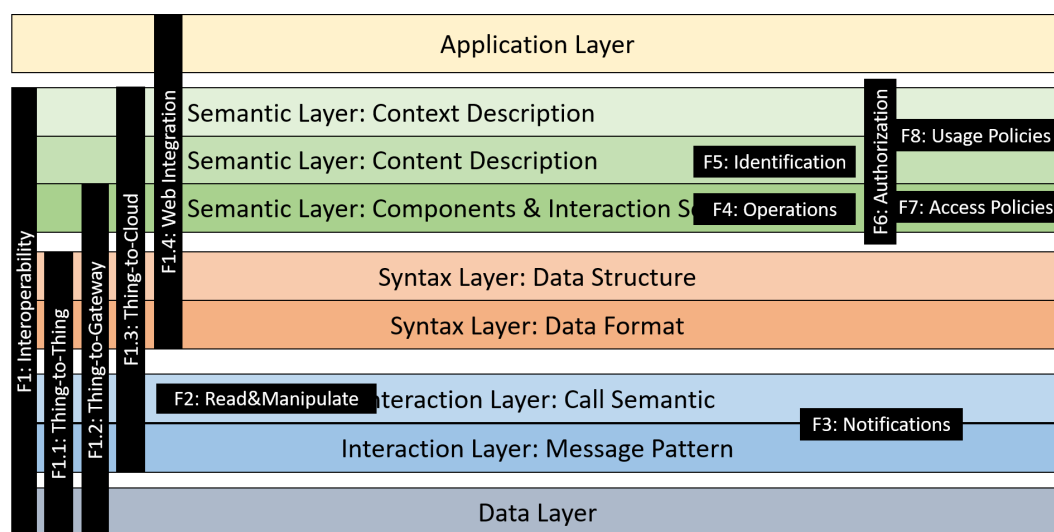


Figure 3. Overview of Functional Requirements positioned in the IoT layer model.

A particular challenge for IoT frameworks is a durable and scalable *identification* (F5) mechanism [33]. A suitable method needs to be able to cope with the different existing identification patterns but also allow the seamless integration of new approaches. Basically, following the outlined example, the Manufacturer must have the necessary freedom to find a *globally unique* character sequence, which is *understood by sufficiently many* current and future systems and combines identification with the *resolution of meta data* in federated environments with a compliant role-based permission model for *authorization* (F6). Furthermore, as Bassi et al. [33] and Kovatsch et al. [22] argue, extending such methods to *access policies* on attribute level (F7), has more fine-grained distinctions as imposed by a role-based model and better fits to the relevant scenarios of the industrial manufacturing domain.

Different to personal data, the data creator (Manufacturer) must not only consider who is *directly accessing* its provided information but also how this *is used* later. Companies in general regard such risks as fundamental obstacles for any data exchange, as the potential misuse can also happen at upstream data consumers and even far in the future. The second crucial difference is the varying protection through legislative rules. Person-related data is, especially in the EU, protected through the General Data Protection Regulation (GDPR). This is not the fact for the business-critical data of companies. Knowledge and intellectual property need to be protected through licenses or contracts.

Furthermore, as stated by Otto et al. [12], access control is only one component of a future-proven data control architecture. As IoT applications are usually organized in workflows and complex communication networks, access control alone only observes the data exchange from one device

to another. Usage Control (F8) as a superset of access control is therefore required, especially in scenarios where different organizations pass on sensitive data in through common supply and communication chains [12].

4.2. Non-Functional Requirements

Security requirements for IoT applications are considered in many publications from the research community ([33,34]) but also from industrial consortia ([9,12,30], cf. Figure 4). For this category, *Trustworthiness* (N1) can be regarded as the root concern using this understanding [9,12]. In this paper, the trustworthiness of a systems is seen as its ability to perform as expected which is strongly influenced by its environment and purpose. Still, the required or achieved trust level directly depends on the lower level characteristics but also on the regarded context.

As Digital Twins form real-world workflows and physically affect both machines and humans, the reliability and trustworthiness of the devices and their behavior is crucial. Security in terms of *data privacy* (N2) of both personal and business-critical data, controlled *integrity* (N3) but also the guarantee of physical *safety* (N4) are fundamental requirements. *Integrity* ([9,34], N3) means that information is not altered. In addition, *reliability* (N5) in terms of expected accessibility ratio, responding times and general availability of a system [9,34], and resilience (N6) against intrusion and denial of service attacks are crucial.

These traditional requirements of automated manufacturing now need to be extended as the shop floor loses its boundaries and the local communication networks become open to the global Internet. Consequently, the very well-known requirements of Information Technology (IT) networks need to be addressed as well. *Scalability* [34] (N7) further adds the characteristic to easily grow and expand without facing sudden limitations. *Security* (N8) in the specific definition of Lin et al. [9] is interpreted as the protection ‘from unintended or unauthorized access, change or destruction’ ([9] page 16), different to the previously mentioned more general understanding of security as an overall requirement, and contains the following aspects. *Confidentiality* (N9) frames ([34]) the secrecy of stored or transferred data. *Anonymity* [34] is related to that, hiding the identity of the related real-world entity of a data object also to the receiver of the communication. *Non-repudiation* (N10) [34] gives the players in a network the assurance that executed actions by a certain party cannot be denied afterwards, allowing derivation of the necessary liabilities for serious business interactions.

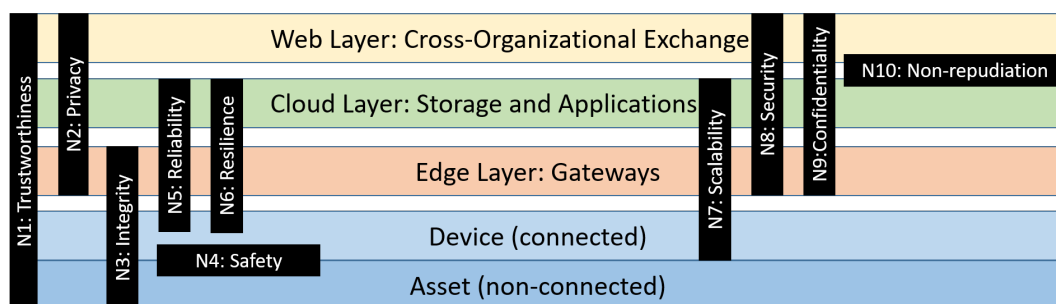


Figure 4. Overview of Non-Functional Requirements by their location in the network architecture.

4.3. Protocol Characteristics

Crucial for any information exchange is the grounding of the outlined characteristics towards well-known protocols (cf. Figure 5). This step makes the requirements actually implementable. A common practice is protocol bindings, which state the translation of the required functionalities into the abilities of already existing protocols (P1). Closely related is the provisioning of *control information* (P2). The link-based hypermedia approach exposes *interaction options* (P3) with the information document itself, while for instance Web Services require special interface descriptions. Similarly, the *coupling between client and server connectors* ([22], P4) and the required configuration effort at the client-side (P5) need to be provided. While the first is relevant for the scalability of a network

(see also N7), the later targets the issue whether interaction parameters are explicitly presented at the resource, externally defined by e.g., a standard document, or even implicitly stated and therefore hard to implement at the client connector.

Generally speaking, a binding must state how a specific protocol implements the demands of the functional requirements while complying to the non-functional ones. As such, the interactions with resources (F2 to F4) are crucial. However, Digital Twins in the regarded applications have several unique requirements as they are usually deployed on small, resource-constraint devices. *Network bandwidth* (P6) is the amount of binary data needed to transport an information unit over the network, closely related to its *energy consumption* (P7).

Especially for real-time applications, low *latency* (P8) is crucial, Latency is directly affected by the selected data format. While the data model and its semantic are described in the following section, the syntactical aspects and especially the supported *Media Types* (P9) are specified by the protocol binding [22].

In terms of the running example, both the Manufacturer and the Data Analyst require a safe and trustworthy but also easily accessible platform for the exchange of the Digital Twin. To stay open for further clients and use cases, they decide against the implementation of proprietary or customized interfaces but stick to a mature and widespread standard.

The Manufacturer and the Data Analyst connect their local machines to a gateway server running SOLIOT. The observed resources are represented by SOLIOT assets. As the whole configuration and every available information might be needed for future applications, all events and states are collected at the SOLIOT server. However, as the created dataset allows insights into their respective operational activities, and even might uncover their competitive advantages and technical knowledge, access must be restricted.

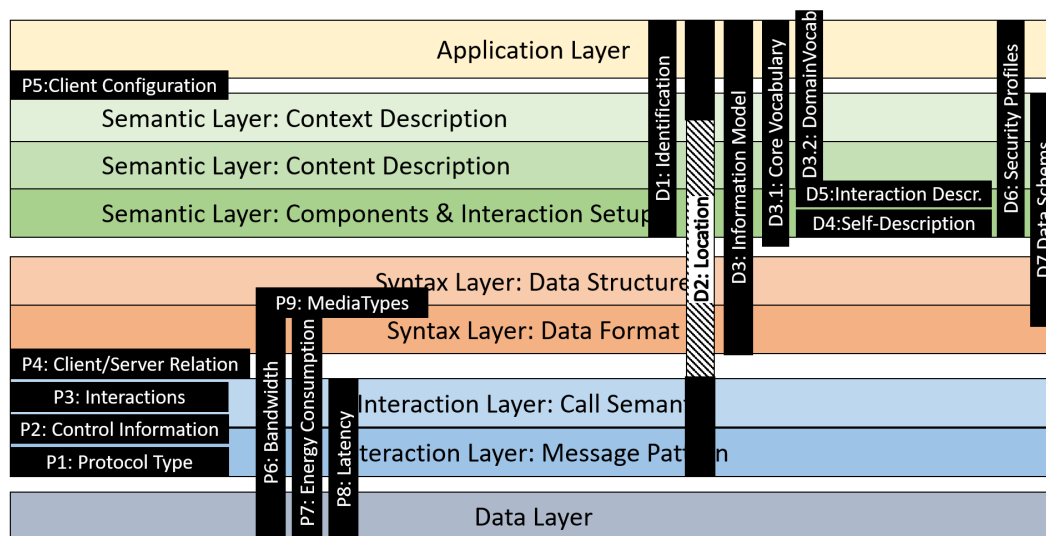


Figure 5. Overview of Protocol (left) and Data (right) Requirements, positioned in the communication stack.

4.4. Data Representation

While the serialized format of the regarded data object is part of the syntactical aspects, its representation, scheme and meaning of the used vocabulary is generally defined in an information model (as in [9,12,22,30,33]). A consistent *identification pattern* (D1) needs to be defined, usually in the form of IRDIs, UUIDs, or URIs. While both IRDIs and UUIDs usually require a certain resolution service, URIs in form of URLs can also directly serve as *resource locators* (D2, cf. Figure 5).

An *information model* needs to define the physical asset, the related Digital Twin, the interrelations its asset, its attributes and the operations it provides [30,33,35] (D3). This *core vocabulary* (D3.1) is then extended with optional *domain specific extensions* (D3.2) in order to describe specific features, which are

not relevant in all use cases. Still, as in the regarded settings a discovery at runtime is required, the resources need to provide *self-descriptive information* (D4) [22] and outline which of the interaction patterns of F2 to F4 are available for the specific resource (D5) [22]. Potential configuration of the access behavior of a hosting system can also be represented and changed through *security configurations* or profiles (D6) which might be exposed to users according to their permission setting [12,36].

In addition to the publishing of the resource itself and available operation, Kovatsch et al. [22] distinguish between the format of such metadata and payload data and argue for the provisioning of *data schemes* (D7). In contrast to the metadata, which must follow the specification of the IoT framework, the payload data might be sent in binary or proprietary serializations [12,22].

5. SOLID Characteristics

SOLID puts the user and its personal data at the center of its considerations. So-called *Pods* serve as container for the user's data, containing all its valuable data. Therefore, the Pod's content can be regarded as the Digital Twin of the user, especially as consuming applications make no further difference between the user itself and its representation through the SOLID Pod.

5.1. Functional Characteristics

The core SOLID design principle is the decentralized management of personal data. Therefore, the *distributed identity management* is the foundation for any interaction process. The usage of WebIDs (F5) allows the creation of identity proofs without a CA chain. Clients obtain a dereferenceable URI as their WebID, by which the obtained certificate is linked to the public key in the user's profile. The relevant data is managed as a *encapsulated data entity* in the form of a personal online data store (Pod), which can be seen as a Digital Twin of the human user. This Digital Twin frames a manageable entity where interactions and manipulations can be applied.

Interoperability (F1) and transportability of pods are gained through *Uniform interfaces* based on the Linked Data Platform specification and SPARQL interfaces. Obviously, the focus is on the Cloud (F1.3) and Web Integration (F1.4) stages. These two patterns also provide the interaction methods with the resources and state transitions (F2). *Complex data retrieval* is required whenever single resources cannot directly fulfill an information need, which to some degree can be accomplished using SPARQL queries. However, SOLID does not specify any methods to call complex operations (F4) on the hosted resources. The design decision to rely on *state-less, RESTful interactions* enables the usage of current available web browsers and Web-based clients for interactions but restricts the direct applicability of richer operations.

Access control (F7) on pods and their subcontainers is implemented by Access Control Lists (ACL) formulating restrictions with terms and concepts of the Web Access Control (WAC) ontology [37], using the WebID or a user certificate (F6). The language states which groups or individual users, identified by their WebIDs, have access, write or control rights on each container. Each container comes with its own .acl file, containing the respective permissions encoded in RDF itself.

This access control mechanism therefore enables SOLID to connect the identity claims of user agents with the identity information behind the dereferenceable WebID. The approach combines identity claims, self-information of the user, and asymmetric keys into a decentralized system without a CA hierarchy. Usage Control (F8) in its wider sense is not regarded.

5.2. Non-Functional Characteristics

As explained in the following, SOLID's focus is on the provision and maintenance of the user's data with clear access rules (N2). Still, the user must trust the hosting service (N1). Data is, for now, not encrypted in the reference implementation and no certification or code review for the server is necessary. Although the open-source approach poses the ability of the community to search for harmful code, the user has no possibility to verify the internal server behavior (N3) at runtime.

The SOLID specification and its current implementations are regarded from a Web perspective. Consequently, safety regards (N4) have not influenced the concept. Also, reliability issues (N5) are in the responsibility of the hosting provider, for instance to ensure sufficient load balancing and protection against DDoS attacks (see resilience N6).

Despite the disregard of N4, N5, N6 and N10, the scalability (N7) of SOLID solutions is reached by the decoupling of the server and clients and by relying on the well-understood REST pattern. Here, SOLID makes use of the mature and proven fundamentals of the Web. The additional intention to move pods such as Digital Twins between hosting services further eases the growth of a SOLID-based network.

SOLID's security model (N8) emphasizes TLS as its core mechanism. One innovative integration feature is the combination of the identity claim in the form of the user's WebID with the hosting of public keys at the WebIDs location. The server is thereby enabled to look up the provided identity proof without having to know any user beforehand. The Pod itself however is not necessarily protected further and its confidentiality (N9) consequently relies purely on the hosting service. This implies the host's databases as a potential attack target.

5.3. Protocol Binding

SOLID is designed for purely HTTP(S) interactions. Consequently, the specification itself incorporates the capabilities of this protocol. Syntactical interoperability is reached by consequent reliance on RESTful interfaces in synchronous interactions (P1). Resources and available interaction methods are announced using HTTP Header fields (P2) as defined by the LDP specification. Additional patterns are available through Web Socket updates and the query language for the Semantic Web, SPARQL, which enables complex information requests. (P3). With both LDP and SPARQL commands, the client and server are only connected (P4) during the request time. Web Sockets however create a constant channel, therefore coupling the two parties together.

The client in the SOLID scenarios is either the human user, interacting through a web browser as its user agent. Alternatively, a SOLID application can use a HTTP client connector to request or update data. In both situations, the relatively high resource requirements of the HTTP stack are necessary (P5). Additionally, reducing bandwidth (P6) or the energy consumption by the clients or server (P7) are, determined by the usage of HTTP, rather high. Furthermore, network latency (P8) must only be sufficient for fulfilling the expectations of human users. Therefore, typical response times of web interactions are sufficient for SOLID use cases. This web-oriented view is also shown by the content negotiation mechanism (media type: P9) and extensive announcement through HTTP headers for on-the-fly consumption in state-less interactions.

5.4. Data Representation

Semantic interoperability is gained by RDF and the use of common vocabularies. Relying on this set of standards and specifications, data objects are identified by URIs; for SOLID actually dereferenceable URLs (D1). As the thereby referenced resources are usually also compliant to the linked-data standards, the client can also follow the links to the resource representation itself (D2). In addition, the structure of the Pod and its internal hierarchy is described with the Linked Data Platform concepts (D3.1). The actual content, the user data and personal information, is described by terms from dc/terms, foaf and vcard. However, any other RDF vocabulary (D3.2) can be used.

The Digital Twin of the user itself is represented by its profile card (D4). VCARD properties define the most common attributes (name, role, organization) while FOAF supports the structure of the social network. However, a clear technical interface description (D5), as for instance possible with WoT, OpenAPI or HYDRA, is not directly supported by the servers. Furthermore, data schemes for validating incoming RDF graphs are not regarded, therefore a broad range of updates are accepted.

As outlined before, SOLID relies on ACLs for managing the permissions (D6). The Web Access Control (WAC) ontology defines the terms and concepts to manage data requests. The namespaces

auth/acl and auth/cert contain the necessary terms for this purpose. SOLID is the combination of a transportable data/entity model and container with transparent and RESTful interaction methods, a clean authorization mechanism for identity management and authentication, and encapsulating processes leading to the data autonomy. How these capabilities can be transferred to Digital Twins in an industrial setting is shown in the next section.

6. Mapping and Interaction Model

The target environment of SOLID is the Web, where data and applications are placed in Cloud servers. Physical Assets and their digital counterparts however may reside on Cloud servers but also at the edge/fog or directly embedded on a device. The obvious differences of these environments must be taken into consideration for a suitable IoT mapping. While typical IoT frameworks and Digital Twin models start with the Thing, the primary target group of the SOLID specification is not the represented entity (human user). The interaction model and interfaces are not directly usable or useful for it but obviously require intermediary tools. Instead, the approach aims at the application level, and intends to simplify the integration at this point as much as possible. Different to the original IoT approaches, SOLID therefore focuses on the consuming, instead of the providing side. SOLIOT combines both views, and thereby lowers the integration gap between the physical Things and the consuming applications. In this section, an outline of the details of this vision, according to the previously outlined categories, is presented.

6.1. Functional Characteristics

At the core of all IoT specifications is the necessity to establish interoperability between the uncountable different systems and devices which are currently in place. While new protocols and interaction patterns are introduced on a regular basis, the core challenge of each new approach is certainly to reach a critical distribution. SOLID relies on the most successful and widespread interaction pattern currently known, the Web. The already proven and well-known techniques, like its server-client architecture or the connection of related information resources through hypermedia links, constitutes a powerful foundation for any integration challenge. SOLIOT therefore takes advantage of the SOLID interoperability (F1) approach, as a broad range of Web-compatible applications already exists and users, developers and administrators are familiar with its characteristics (cf. Table 1).

Table 1. Mapping of Functional Requirements.

	SOLID	SOLIOT
Interoperability (F1) Shared data:	LDP, RDF, SPARQL SOLID Pod	Read/write RDF for IoT Thing Description as the Digital Twin
resource Interactions (F2)	LDP specification, Web Socket Updates, SPARQL	CoAP CRUD operations, MQTT Updates
Manage Notifications (F3)	Linked Data Notifications	MQTT Updates
Invoke Operations (F4)	-	-
Identification (F5)	WebID	Identity and/or Security Token
Authorization (F6)	HTTP/1.1 Authentication	–
Access Policies (F7)	Web Access Control (WAC)	WAC through ACL files
Usage Control (F8)	–	–

The targeted use case of SOLID is an on-hop interaction between a defined Web client as the data consumer and the SOLID server acting as the origin server. The Pod is hosted in a Cloud environment, where the Cloud server complies to the SOLID specification but can be located anywhere in the Web. The users are either the data sovereign controlling the Pod, or consuming applications authorized by

the data sovereign. As SOLID models the social data of the data sovereign, the Pod can be regarded as the Digital Twin of the data sovereign.

In an IoT setting, this overlap of roles is not possible anymore. The referent of the Digital Twin is usually not a natural person but a physical object. Still, a (legal) entity must exist which has the authority over the target asset. An additional difference is that in the basic SOLID setting, all components communicating in the network have sufficient resources, in terms of computing power, local storage, memory or provided energy. One can certainly assume that a service hosting SOLID Pods, like a typical state of the Art Cloud service, can scale and allocate additional resources as needed. At the same time, the user agent, usually a modern Web browser executed on a regular PC or laptop, is not challenged by the computational or network requirements of SOLID (F1.4).

In an IoT setting however, constraint devices, battery-powered sensors, network disruptions etc. are common challenges. Usually a constant Internet connection cannot be maintained, neither the sending of complex data structures like full-flavored RDF or linked-data graphs. Consequently, SOLIOT-supporting devices cannot be assumed to be at the same network layer as a SOLID server (F1.2). Edge or Fog Gateways can serve as suitable bridges between such restricted devices and the rich expressions of SOLIOT. Such gateways serve as lifting and lowering points (F1.3) between the proprietary device communications. They however usually lack the rich expressions of the SOLIOT representation and can therefore not directly be integrated with higher-level applications (F1.1). The SOLIOT functionality extends the Edge Gateway service and links the represented resources, overcomes the heterogeneity challenges and connects what is often known as the shop floor using Operational Technologies (OT) with the Information Technologies (IT) of the office floor.

However, when it comes to the details, several restrictions have been made. While SOLID promotes LDP interactions but also enables Web Sockets updates and SPARQL queries, SOLIOT only implements the basic CRUD operations (F2). Although these operations follow the LDP guidelines as much as possible, the whole expressiveness must be reduced. Complex queries or even operational calls (F4)—beyond CRUD—are not possible.

6.2. Non-Functional Characteristics

Many requirements from OT systems are not supported by SOLID as they are not relevant for its use cases (cf. Table 2). For instance, code integrity (N3) of the hosting platform, reliability (N5) or resilience (N6) are either as part of the implementation details. Furthermore, SOLID itself does not regard physical safety (N4) as no SOLID application is intended to physically interact with its users or any other real-world object.

The SOLID use cases focus on decentralization of data storage and control, consequently privacy (N2) and scalability (N8) of SOLID networks are distinguishing features. The intended easy transmission of pods is currently without comparison. However, the current state of the specification [38] does not yet give sufficient guidelines how one hosting service can ship a Pod to another. Still, in a dynamic IoT network with potentially changing settings, the ability to adjust the location of resources and to ship all information seamlessly is an important feature.

Table 2. Mapping of Non-functional Requirements.

	SOLID	SOLIOT
Trustworthiness (N1)	Depending on the hosting provider	Depending on the hosting provider
Data Privacy (N2)	TLS encryption, Access Control Lists (ACL)	TLS recommended Access Control Lists (ACL)
Integrity (N3)	-	-
Safety (N4)	-	-
Reliability (N5)	-	-
Resilience (N6)	-	-
Scalability (N7)	Decentralized Identity Provisioning, Decoupling of client and server	Depending on the MQTT Message Broker Decentralized Identity Provisioning Decoupling of client and server
Security (N8)	TLS encryption Access Control Lists (ACL)	MQTT over TLS (not implemented) CoAP with DTLS (not implemented)
Non-repudiation (N10)	-	Not regarded

The merging of local, decoupled networks with the global Internet is one of the most impacting developments. While previously the security and integrity was ensured by restricting the access to sensitive parts of the shop floor, the rising need for interconnections undermines any firewall or otherwise intended separations. Consequently, each component needs to implement the complete security stack, as if it were placed in the open Internet. Such sometimes called ‘Zero Trust’ (N1) approaches demand the same proofs from any connecting party. Data or functionality must be prohibited without proper identity claims and whenever not backed up by the ACL rules. This transparent data protection (N2) approach enables the arriving user agents to understand potential obstacles and deficiencies.

Measures to ensure a required Integrity (N3) and Resilience (N6) level are for instance proposed by the IDS [39]. Hardware-based trust anchors or virtual monitoring of a server’s state, together with certified software modules, can increase the trust level both of the party operating a SOLIOT instance and the interacting user agent. These concepts however are still in their maturation process and only mentioned here. In general, a guaranteed resilience or integrity level cannot be accomplished without regarding the use case context.

This is certainly also true for Safety (N4) considerations. Neither SOLID nor SOLIOT have out of the box understanding of the meaning—and potential risks—included in their data. Gradual approaches to this issue could be outlier detection or Complex Event Processing engines on top of the SOLIOT notifications. The self-descriptive nature of SOLIOT events, through their RDF-encoded content, certainly lowers the integration effort for third-party tools. As such concepts are however very use case specific they shall not be regarded further here.

In contrast to that, Reliability (N5) can be supported by common methods like periodic backups and load balancing. Such measures can be applied to the SOLIOT hosting instance itself. One important consideration is the distinction between the offered state of a data resource and its internally managed history. A SOLIOT implementation should use transaction-driven data handling for each resource, allowing the seamless reverse of harmful interactions.

As the network surrounding of a SOLIOT instance is not observed by the instance and maybe not observable at all, it must be assumed as compromised by malicious third parties. Therefore, a SOLIOT system must insist on the delivery of valid authentication and authorization proofs from anyone. The recommended pattern is Bearer Tokens from a local OAuth service (N7) with a duration of less than one hour. If the requesting of such dynamic tokens overstrains the abilities of a device, longer or even stable tokens may be accepted. Nevertheless, whenever unsafe interactions are requested, a TLS channel and an additional examination of the local ACL rules is required (N8).

The blank spaces in Table 2 are obvious indicators for open research gaps. Even though the idea of the IoT and the merging of production spaces with open networks is not new, the lack of commonly accepted and implementable technologies is certainly one of the most relevant obstacles for a seamlessly integrated IoT setting in productive use. SOLID and SOLIOT-inspired solutions, like any other currently available, have still significant weaknesses in the outlined areas. Traditional systems can perform better in some respects but usually are highly customized and therefore significantly harder to integrate with other applications.

6.3. Protocol Bindings

The following examination about protocol bindings and data representations (Section 6.4) is highly related with the design decisions of an IoT system. In contrast to the previous sections, only the identified challenges and issues are outlined. As SOLID's deep integration with HTTP presents a severe challenge for resource-restricted environments, the main focus is applied on the concept transfer and interaction mapping to the mentioned, less-demanding protocols. Naik has already examined the resource demands of IoT protocols [40]. The results are presented in Table 3. Obviously, CoAP has the lowest requirements of the selected protocols, followed by MQTT.

Table 3. Protocol Bindings: SOLIOT in MQTT and CoAP.

	HTTP	MQTT	CoAP
Interaction Type (P1)	Synchronous, request/response	Asynchronous, publish/subscribe	Synchronous, request/response
Interaction Announcement (P2)	Allow and WAC-Allow Headers announce available operations	TD Affordance	TD Affordance, WoT CoAP Binding
Resource Discovery	Link Header, HATEOAS	Topics under soliot/#	/.well-known/soliot HATEOAS
Operations (P3)	RESTful CRUD/SPARQL	Event notifications	RESTful CRUD
Server-Client Coupling (P4)	Decoupled, optionally coupled through Web Sockets	decoupled	Central Message Broker
Efforts at device-side (P5)	High (HTTP protocol stack)	low	low
Bandwidth (P6)	High	Low to medium	low
Energy consumption (P7)	High	Low to medium	low
Latency (P8)	Low priority	Real-time Control	near Real-time
Media Types (P9)	Turtle, JSON-LD, binary etc.	JSON-LD	JSON-LD

Lin et al. [9] argue that the single focus on core OT requirements is not sufficient anymore. The adoption of IT practices, including HTTP and its rich client landscape, can introduce a game changer for usage experience in IoT applications. While embedded and restricted devices may still be limited to particular protocols, support of HTTP at the edge or beyond may solve a huge set of interoperability problems.

6.3.1. CoAP for State-Based SOLIOT Interactions

As stated, CoAP follows a similar approach as HTTP but with smaller messages. Methods, header and status codes are easily mapped to their HTTP counterparts while the counter direction is more limited. Similar to HTTP, CoAP is request/response-based protocol, relying on resources and supporting RESTful interactions. However, instead of TCP, CoAP uses UDP as transport protocol. Consequently, TLS is not supported. Instead, the Datagram Transport Layer Security (DTLS) protocol can be used [41]. A first mapping of LDP interactions to CoAP has been created by Loseto et al. [27]. Extending this mapping to SOLID functionalities serves as the foundation of the SOLIOT CoAP

protocol binding explained in this section. The binding itself is outlined through a comparison of interaction patterns (cf. Table 4), a translation of SOLID header definitions (cf. Table 5), and a discussion on the different interpretations of the message body interpretation.

Table 4. Interaction methods with CoAP, based on Loseto et al. [27].

Interaction	Mandatory	HTTP Method	CoAP Method	CoAP Status Code
Create	no	PUT/POST	put/post	2.01 Created
Read Resource	yes	GET	get	2.05 Content
Read Metadata	yes	HEAD	get ?ldp=head	2.03 Valid
Read Operations	yes	OPTIONS	get ?ldp=options	2.05 Content
Extend	no	PATCH	put ?ldp=patch	2.04 Changed
Overwrite	no	PUT	put	2.04 Changed
Delete	no	DELETE	delete	2.02 Deleted
Subscribe	no	–	–	–
Unsubscribe	no	–	–	–
Notify	no	polling?	–	–
Alarm	no	polling?	–	–
Invoke Functions	no	–	–	–

Table 4 distinguishes the different interaction intentions relevant to an IoT scenario as explained in our example. Some of the thereby included patterns (subscriptions, notifications, alarms) are not part of the core SOLID use case. Still, as for instance Kovatsch et al. [22] or Bassi et al. [33] outline, a limited request/response approach is not sufficient. The design of the table reflects this insight, therefore also containing white spaces. These white spaces (e.g., row ‘invoke functions’) can be relevant for further specifications and therefore have been kept intentionally in the tables.

Obviously, the amount of protocol methods is lower for CoAP. As Loseto et al. [27] suggested, the replacement of the HTTP HEAD, OPTIONS, and PATCH headers can be expressed through query parameters. While one can argue that the core CRUD operations are still natively possible through the given CoAP methods (get, put, post, delete), the mandatory requirement for HEAD and OPTION support in SOLID makes a compromise necessary. As CoAP’s main intention is the reduction of resource consumption, several critical requirements of an IoT system must be disregarded. For instance, the limitation of the protocol headers enforces the expression of several information points in the message body, increasing the integration effort at the client.

Table 5. Protocol Headers, similar to Loseto et al. [27].

Function	HTTP Header	CoAP Option
Data Serialization	Content-Type	Content-Format (ct) option
Allowed Operations	Allow	(JSON in Response Body: TD Affordance)
Server-supported Media Types	Accept-Post	(JSON in Response Body: TD Affordance)
Server-accepted Media Types	Accept-Patch	(JSON in Response Body: TD Affordance)
Proposed Resource Name	Slug	title
Resource Location	Location	location-path
Security Token	Bearer Token	(custom Security Token option)

The CoAP headers are significantly different than the originally used ones through the HTTP binding. CoAP headers are encoded as numbers to reduce the message size. Important headers, for instance ‘Accept’ (‘17’) or ‘Content-Type’ (called Content-Format: ‘12’), are registered at IANA. In addition, common media types are globally defined as well (‘41’: ‘application/xml’, ‘50’: ‘application/json’). SOLIOT promotes the usage of the reserved media type code ‘432’: ‘application/td+json’ for Thing Descriptions. The resulting content is a JSON-LD serialized RDF representation of a ‘Thing’.

6.3.2. MQTT and SOLIOT Events

Different to CoAP and HTTP, MQTT is a message-based protocol and event-driven. While it is possible to realize all listed interactions through explicit descriptions in the message body—for instance, a READ request could be encoded as an especially formatted JSON attribute—such an approach would shift the interpretation task to every receiver of a message. The additional effort to parse and interpret the complete content before realizing its meaning seems not an efficient pattern.

Therefore, a three-stage process is applied to encode messages. The first step is realized by the native MQTT method itself. PUBLISH, SUBSCRIBE and UNSUBSCRIBE define the native separation of messages and are interpreted by the message broker without regarding the rest of the message (cf. Table 6). The second step is specified by the root part of the used topic. The reserved upper-level topic *soliot* tells the involved parties that the following message corresponds to the SOLIOT interaction semantics. The distinct interaction is then listed at the second position of the topic string. Please note that for the SUBSCRIBE and UNSUBSCRIBE calls reflect the variability with the wildcard at the second position as their meaning is already completely defined by the combination of the method itself and the *soliot* topic itself.

The data flow is also already defined at this stage. The SOLIOT servers—origin servers acting as MQTT clients—react to the unsafe interactions ‘Extend’ (*‘soliot/patch/#’*) and ‘Overwrite’ (*‘soliot/put/#’*). These messages originate at the user agents—also MQTT clients—and must contain a valid JSON-LD object. The SOLIOT server is therefore required to also subscribe to all topics related to its own MQTT-enabled resources. These resources are placed at the third part of the topic sequence through their base64-encoded URI identifier. However, even though receiving a change request, the origin server may or may not actually change the targeted resource.

The MQTT Message Brokers as proxies are responsible to handle the core MQTT methods PUBLISH, SUBSCRIBE and UNSUBSCRIBE. It is recommended that the message broker enforces encrypted communication (MQTT over TLS using default port 8883). Additional protection can be gained by maintaining access-control lists directly at the message broker and restrict access to known users. Using ACLs and user authentication at the MQTT Broker however creates redundancy with the native SOLIOT access control scheme using Web Access Control and maintaining the permissions directly at the resource.

User agents, also appearing as MQTT clients, signal resource changes to the origin server (*‘Extend’* or *‘Overwrite’*). This usually applies when the clients get informed by the resource itself, for instance a sensor sending a new observation. The user agent must regard the *‘Extend’* interaction as not idempotent, meaning that repeatedly submitting the same event will create an inconsistent state at the origin server. In addition, notifications and alarms are triggered by the origin server. In both cases, the subscribed MQTT clients are the information sinks. Similar to the update interactions, the server can use the topics *‘soliot/patched/#’* to only send the added attributes or *‘soliot/putted/#’* with the complete new state.

Table 6. Interaction methods with MQTT, based on Loseto et al. [27].

Interaction	Mandatory	MQTT Method	Status Code
Create	no	–	–
Read Resource	yes	–	–
Read Metadata	yes	–	–
Read Operations	yes	–	–
Extend	no	publish soliot/patch/{resource}	–
Overwrite	no	publish soliot/put/{resource}	–
Delete	no	–	–
Subscribe	no	subscribe soliot/+/{resource}	–
Unsubscribe	no	unsubscribe soliot/+/{resource}	–
Notify	no	publish soliot/{interaction}/{resource}	–
Alarm	no	publish soliot/alarm/{resource}	–
Invoke Functions	no	–	–

In contrast to CoAP, the MQTT binding only regards the event-driven interactions. Create, Read, and Delete are not regarded as their context is state-based, therefore being more effectively implemented by either CoAP or HTTP. Furthermore, Tables 4 and 6 both do not specify the invocations of remote functions or services. Even though this functionality is mentioned by several requirement lists, the exact semantic of such remote service calls is highly use case specific. For now, the given protocol bindings for their description and execution are intentionally left open for future work.

The *header* of MQTT are not intended for the transfer of rich interaction information. For instance, a typical MQTT client receiving information cannot understand or lookup the origin servers offers or capabilities. Nevertheless, the demand to receive information on the message broker's current state has led to the convention of the top-level topic '\$SYS'. A similar pattern would be possible for SOLIOT look ups. The event-based nature of MQTT however speaks against this approach. The continuous flooding of the same origin server descriptions, independent of any demand, usually only blocks important resources without added value. Whenever a client is interested in the origin server's state, it should use a state-driven protocol as CoAP or HTTP.

The *message body* must again contain a valid JSON-LD object with exactly one root resource. The URI of the root node must be either stated explicitly—and identical to the decoded resource URI of the message topic. Alternatively, a relative URI allowing a more flexible scheme requires the receiver of the message to relate the content with the resource stated in the topic. Using a differing topic but specifying the actually intended resource solely in the JSON-LD payload is therefore not allowed.

6.3.3. SOLIOT Network Architecture

Two basic architecture designs are possible. The SOLIOT server can host a MQTT message broker and expose its endpoint to subscribing clients. Alternatively, the SOLIOT application only implements a MQTT client socket and distributes the notifications using an external MQTT message broker. The obvious advantage is a reduced computing load on the SOLIOT instance and increased scalability through the usage of several Message Brokers. However, the access control—moreover, any kind of data protection—is thereby transferred to the MQTT Broker and thereby not under control of the provider anymore. A MQTT Broker may or may not take the respective ACL files into account, the providing SOLIOT application cannot be certain. While this challenge certainly can be solved through additional mechanisms, for instance exchanging encrypted payloads and at the same time handing out proper decryption keys only to authorized subscribers [42], this shall only be briefly mentioned at this position. Still, the outlined issues are highly relevant but regarded as out of scope for explaining the core of the SOLIOT approach.

The next requirement is the choice of the appropriate media type (P9). RDF and its serializations have been designed with a clear focus on interoperability and self-description. The schema-less

characteristics of RDF objects allow its flexible usage and simple extension at runtime. As the commonly appearing data files kilobyte to low megabyte range, this is usually problematic for both Web servers and clients. In the regarded use case however, this is obviously a significant challenge. One way to solve it is to examine the different serialization formats, namely RDF/XML, NTriples, Turtle and JSON-LD.

A differentiation must be made for JSON and its RDF variant, JSON-LD. A plain encoding of RDF with completely serialized URIs as JSON keys and values requires a lot of space and is therefore not suitable. Namespace replacements through the '@context' element reduces the required space significantly but increases the parsing effort. Receiving parties need to look up the reference if they do not know it yet, resulting in probably even higher traffic. The current context of the 'Thing Description' alone has more than 22,000 bytes. Still, the broad usage of JSON justifies this trade-off, and makes it the recommended format. To face the context issue, SOLIOT places a respective JSON file at the reserved path '/things/context.jsonld'.

The further identified requirements of protocol bindings (Bandwidth (P6) and Latency (P8)) are discussed in Section 8. Energy Consumption (P7) is by its nature hard to measure for a concept such as SOLIOT and was not examined separately. It is assumed that the bandwidth is a sufficient indicator for the power demand of a SOLIOT implementation.

6.4. Data Representation

SOLID's data model is oriented by the human user and its attributes, IoT devices require a different vocabulary and data scheme (cf. Table 7). The Thing Description (TD) ontology, Semantic Sensor Network (SSN) and Smart Appliances Reference (SAREF) ontology are only examples of the rich set of vocabularies and ontologies recently designed for the domain. eCl@ss and the IEC Common Data Dictionary (IEC CDD) include rich sets of non-RDF classes and attributes. While a broad range of terms and concepts is given, the actual integration between different parties and organizations requires more restrictions and selection of supported entities.

The LDP container model allows the usage of any correctly structured RDF whether the property or entity is known to the server or not. In an IoT system, the host must restrict the variety to some degree. Therefore, schemes and shapes such as SHEX or SHACL can be used to validate incoming information. Alternatively, the server can provide extended mapping capabilities and ensures the provisioning of known terms in its own authority. Nevertheless, this step, if executed autonomously, is error-prone and therefore not feasible in an operating environment.

Table 7. Data Representation Mapping.

	SOLID	SOLIOT MQTT/CoAP
Res. Identifier (D1)	HTTP(S) URL	Topic ID, CoAP URL
Resource Locator (D2)	HTTP(S) URL	CoAP URL, TD Affordance
Information Model (D3) generic (D3.1) domain specific (D3.2)	RDF, RDFS, LDP pim/space, posix/stat DCTERMS, FOAF VCARD	RDF, RDFS, LDP Thing Description SSN, SOSA, SAREF, (eCl@ss, IOF, IEC CDD)
Self-Description (D4)	</profile/card#me> as a Person	</things/thing#me> as a Digital Twin
Interaction Descr. (D5)	HTTP Headers (Allow, Allow-Post, Allow-Patch, WAC-Allow ...)	TD Affordances
Security Profiles (D6)	WAC in ACL files	WAC in ACL files
Data Schemes (D7)	–	(SHACL)

Consequently, using SOLID in an IoT environment requires additional restrictions and validation mechanisms even though this limits the expressiveness of the provided information. Using the Thing Description vocabulary (namespace prefix 'td' [43]) as the Core Vocabulary (D3.1) in combination

with the Linked Data Platform annotations, the instances of `td:Thing` are the top-level entities in the reserved top-level container `‘/things/’` (cf. Figure 6). The identifying URI of such an instance must be composed of the authority of the server, `‘/things/’`, and a locally unique character sequence. The Thing is treated and described as a `ldp:BasicContainer`, and implements the thereby required interaction behavior. Alternatively, a relative URI only using the locally unique character sequence can be applied. The full URI of the `td:Thing` is the implicitly set regarding the location on the hosting server.

An instance of `td:Thing` contains at least a minimal self-description (D4) outlining its type (class relations by *rdf:type*) and human readable annotations in English (*rdfs:label*, *rdfs:comment*) and potentially more languages. It can link to further resources using three different patterns. Datatype Properties are contained directly in the document representing the instance. Object Properties referencing external resources shall also be contained in this RDF document but limiting the provided information to the Object Property and referenced object URI. The origin server shall shift the decision to the client, whether it wants more information of that object entity. If so, the client resolves the identifier in a linked-data conform way, and discover the references resource itself.

The third pattern is the reference to locally hosted resources. The instance of `td:Thing` acts as a `ldp:BasicContainer`, using relative URLs to point to the intended target. Furthermore, it outlines the existence of child resources through `ldp:contains` properties. It is recommended to add a property to tell the client, which kind of relation the `td:Thing` connects to its child. Listing 1 contains a representation of the intended Digital Twin model. Please note that despite that JSON-LD is the recommended format, the Turtle serialization has been chosen to increase the readability of the example.

Listing 1: An IoT device presented as a SOLIOT Digital Twin.

```
<coap://18.157.197.66:5683/things/3S7PM0CP4BD>
  a td:Thing, ldp:BasicContainer, saref:Device ;
  rdfs:label "3S7PM0CP4BD"^^xsd:string ;
  td:title "3S7PM0CP4BD"^^xsd:string ;
  td:description "OVEL Vacuum generator"@en ;
  td:security <./acl> ;
  td:actions <./affordanceCreate>, <./affordanceDelete> ;
  td:events <./affordanceNotification> ;
  ldp:contains <./120363/>, <./affordanceCreate>, <./affordanceNotification>, [...]
```

All enabled interaction possibilities (D5) are expressed also using Thing Description Affordances. In addition to the discovery of interaction patterns through the linked-data specifications using HTTP Header, the affordances bridge the gap to the non-HTTP protocol bindings. A client, unaffected by its used protocol, can thereby find the related resources, respective endpoints with protocol specifications, and lookup the necessary input and provided output details. For CoAP, the CoRE Link Format further standardizes the principles of discoverable resources in RFC 6690. These attributes are used to reflect the HTTP headers wherever possible.

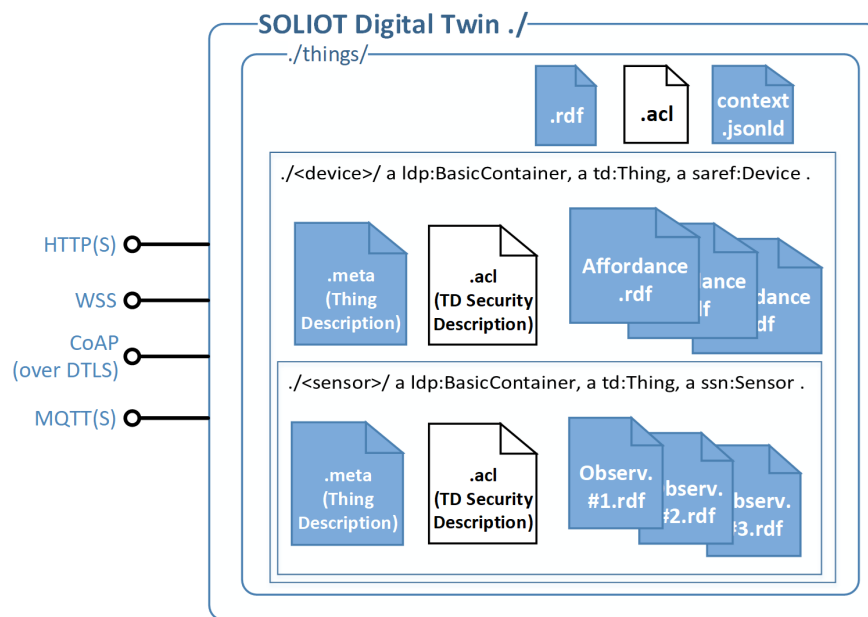


Figure 6. Representation of the Digital Twin through interlinked Containers, Resources and ACL Files in a SOLIOT instance.

Domain Vocabularies (D3.2) for now are the Semantic Sensor Network (SSN/SOSA) ontology [44] and the Smart Appliances REFERENCE (SAREF) ontology [45]. Further descriptions of concepts and aspects originating from the description of core manufacturing, logistics, or engineering attributes shall be used from the Asset Administration Shell ontology [46] and the currently developed Industrial Ontology Foundry (IOF) [47]. Furthermore, the rich domain vocabularies maintained by eCl@ss [48] and IEC CDD (IEC 61360) define lightweight concept definitions of huge term sets using IRDIs. The currently undergoing efforts to deliver eCl@ss also through linked-data principles as RDF Resources and URI identifiers promises an even simpler integration.

The security profiles (D6) are represented in the same way as proposed by SOLID. ACL files containing Web Access Control statements can be positioned at the resources and containers to describe and at the same time configure a Role-based Access Control model. A SOLIOT instance however should not rely on WebIDs, as its communication partners may not be able to prove their identity in this way. Shorter tokens, especially JSON Web Tokens with encoded claims through JSON-LD, can compose a suitable trade-off.

A further restriction of data intersections is composed through SHACL shapes. These shapes serve two purposes. They directly address the need for data validation and schema restrictions (D7) but also, as readable resources themselves, describe the expected or offered data. A sending client can thereby understand the expected data structure beforehand, and even test its outgoing resources locally. A receiving client can adjust its own expectations by analyzing the shapes, and derive a better forecast of the resulting resources.

The resulting virtual presentation of the Digital Twin is therefore, as shown in Listing 1, formally presented through as a 'td:Thing' as defined by the Thing Description ontology. The thereby incorporated machine-readable properties and attributes allow a client to autonomously interpret its capabilities, given that it is familiar with the vocabulary. If not, the client can also dereference the identifiers and further discover their meaning on the fly. The interaction model is further outlined through the presentation as a 'ldp:BasicContainer' and the 'td:actions' and 'td:events' attributes. These provide the client all information necessary to interact with the SOLIOT Digital Twin. Moreover, also every REST and HATEOAS-aware client can consume the complete representation, manipulate attributes and further discover the related sub-resources.

SOLIOT therefore covers required characteristic of Digital Twins, the virtual representation, through self-described data representations. Furthermore, the complete interaction model is explicitly outlined, giving both a consuming application and a producing data source all necessary information. As all related actors communicate relying on the same interaction patterns—even though applying the protocol of their choice—the overall complexity of the interfaces is reduced significantly. This second requirement for Digital Twins, the interaction between the Thing and the virtual representation, is therefore solved in the same manner as the outside communication. This increases the scalability and maintainability of a SOLIOT-based network and at the same time reduces the integration effort of higher-level applications.

The ‘Thing’ itself however is less regarded by the proposed model. To truly connect the physical with the virtual world, not only the virtual part of a Digital Twin needs to address the Thing but also vice versa. The common use case is the dereferencing of a Thing’s physical identifier. One could expect that each Thing directly refers to its virtual counterpart. This is however not yet sufficiently examined. Challenges are, for instance, the existence of more than one digital representation for a single Thing. Furthermore, a physical identifier must stay applicable throughout the complete life cycle of the underlying Thing, which can be decades for many production-related assets. Neither SOLIOT nor any other Digital Twin model known to the authors currently supports this kind of durability. While the Asset Administration Shell describes first steps in this direction using nameplates [10], the overall challenge remains unsolved.

7. Architecture and Prototypical Implementation

The proposed SOLIOT approach serves as gateway between the production networks and the open Internet. Similar to SOLID, the related entity is described by its profile and presented as an LDP Resource. However, the extension of the supported protocols to CoAP and MQTT added IoT-specific terms and concepts and designed tailored interaction patterns. Widely deployed architectures rely on a distinct integration layer or specialized gateways connecting the data-providing Things with the data-consuming applications [6,33,49,50]. Figure 7a shows a simplified representation of this concept, where the Digital Twins appear at the Edge Layer (also called integration layer, IoT platform, gateway depending on the context). The Thing from the example is deployed in the Manufacturer’s shop floor network, and its data output hosted on an edge gateway. The operator of this gateway is responsible to grant or deny access, lift information and forward requests. The basic architecture of the SOLIOT is closer to the approach of Pfrommer et al. [2] or Jammes and Smit [18] as an equal node on the same network level as the IoT device and the (external) consumers (see Figure 7b).

SOLIOT does not make any differences between the connected components. Theoretically, also the Data Analytics Platform of the Analyst could represent itself as a Digital Twin on a SOLIOT server. The distinction between data producers and consumers is therefore not specified by their location in a network but their applied permissions. The permissions themselves are also accessible resources (ACL files in Figure 7b). This follows the vision of reducing the network barriers, merging shop floor network with office floor application and even the public Internet. While both the Data Analyst and the Business Partner have encoded ACL policies, this is not the fact for the Competitor. Consequently, his access requests will be denied. Nevertheless, the dissolution of network barriers also exposes previously shielded areas and devices, requiring proper security and protection mechanisms. However, following the ‘zero trust’ argumentation, the increasing degree of interconnections, required gateways and introduced foreign applications makes a network-based security approach more and more challenging.

SOLIOT has been prototypically implemented as an open-source project [51] under an Apache 2.0 license. The NodeJS reference implementation of SOLID serves as the core server, extended with a CoAP server [52] and MQTT [53] capabilities. The SOLIOT reference server provides RESTful, LDP-conform interactions for CoAP clients, and resolves event-based interactions using its MQTT

adapter. Currently, an external Mosquitto MQTT message broker [54] collects and distributes MQTT messages.

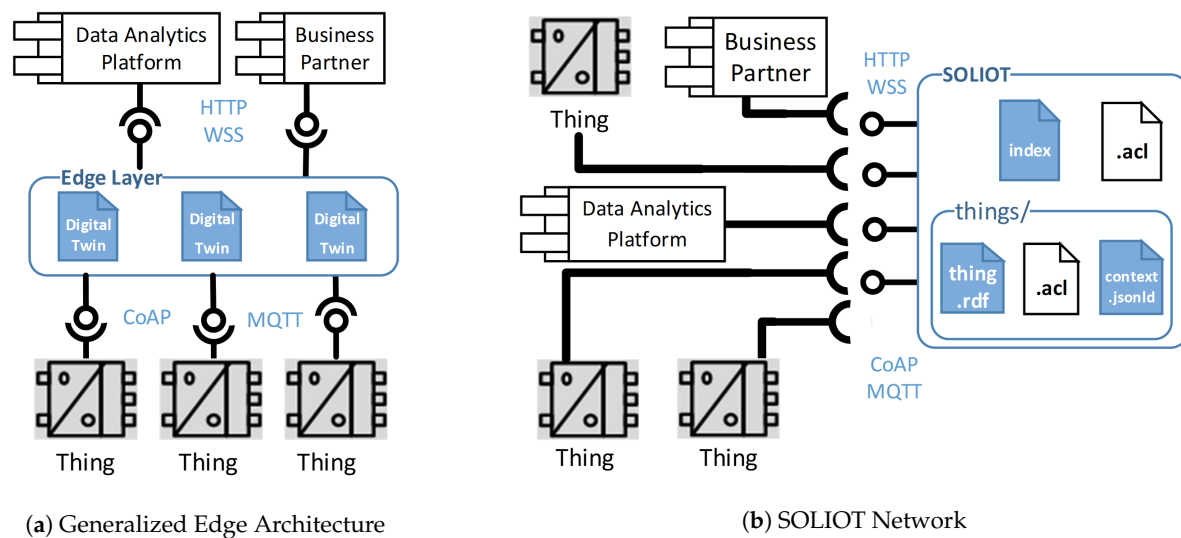


Figure 7. Architecture Comparison.

7.1. Interactions and Protocols

For the prototypical implementation, CoAP as a lightweight request–response protocol and MQTT as a publish–subscribe implementation have been selected. Respective endpoints for each protocol socket need to be opened, by default 1883 for MQTT and 5683 for CoAP.

At each stage of the Digital Twin container hierarchy, the possible interactions are described through TD affordances. The affordances themselves impose regular RDF resources, and are treated as LDP children of their describing container. As such, the container links to them both through `td:actions/td:events` and `ldp:contains` attributes.

In terms of MQTT interactions, several architectures are possible. For (a) point-to-point communication, an origin server could push its state to a MQTT Broker at the user agent. The user agent also deploys a client subscribing to its own broker, thereby receiving the sent message. Alternatively, the broker connector could be placed directly at the origin server, and all user agents interested into the resource state are subscribed there (b). Any event is then published by the origin server’s internal MQTT client connector to this broker. This approach would, for instance, allow the origin server to control which user agents receives the messages.

However, both approaches inherit significant drawbacks. In case (a), the origin server must know the user agent beforehand, which is an unrealistic assumption in a dynamic IoT network. Approach (b) requires that the device creating the data hosts an additional broker component. Embedded devices may not have the computing power or energy resources to do so. Furthermore, a 1-to-m, n-to-1, or 1-to-1 interactions as imposed by both (a) and (b) do not consider the strengths of MQTT but misuse the protocol to some degree. Communication patterns such as that can be implemented easier with other protocols and less overhead.

Regarding the publish/subscribe pattern of MQTT, the assigned broker should be deployed outside of the SOLIOT instance but also of the origin server. The SOLIOT server only implements an MQTT client connector. In contrast to a state-based view of e.g., HTTP and CoAP, none of the CRUD interactions have been implemented yet. The information flow is therefore currently only supported from the SOLIOT origin server to the user agent. Next iterations of SOLIOT will address this issue and extend the current model unsafe interactions. This way, ‘Things’ with client sockets will be enabled to directly influence their own Digital Twin’s Thing Description.

7.2. Use Case Scenario

Following the scenario from Section 3, the Manufacturer and Data Analyst connect their local machines via CoAP or MQTT. The SOLIOT instances are the only providers, waiting for incoming requests. These requests must follow SOLID specifications, and are therefore easy to implement for any connecting party.

The administrators responsible for A and B initialize pods for all devices, hosted sensors, machines or even complete facilities relevant for their operation. They create according profiles, deposit the master data and configure the security regime. One part is the storage of identification information to let the SOLIOT server know the addresses of the incoming data. Furthermore, the .acl files are adjusted so that all administrators obtain full control, internal systems gain read and write permissions where needed, and external applications only can access the negotiated information. They do this through the same interaction patterns (Read: GET, Update: PUT etc.) as the clients at runtime.

As soon as the system has been set up, the local sensors push their observations via their preferred protocol to the SOLIOT server. For supplier A, the devices send CoAP PUT messages with the measured values in JSON. The SOLIOT server validates the token, maps and validates the content. If the server can process the message, the according attribute in the sensor profile is updated.

8. Evaluation

At the core of the proposed SOLIOT approach is the combination of the uniform interfaces of the LDP specification (interoperability requirement) with the mature access control regime (data control) as defined by SOLID with the restricted resources of IoT applications (constraint devices). Evaluating these stated contributions target different dimensions and therefore requires different methods. As the interoperability features are heavily relying on the provided SOLID specifications, a distinct validation of their features would only provide a minimal added value. It has been decided to supply proof of concept also as a deployed sandbox instance at (<https://18.157.197.66:8443/8883/5683>). The typical ports for HTTPS (8443), CoAP (5683), and MQTT (1883) are used. The open code base allows the interested reader to verify the created endpoints implementation and examine the security implementation.

The evaluation setup relies on Amazon AWS EC2 instances. The prototypical SOLIOT server has been shipped to T2 Micro instances and deployed using the AWS Client API. This way, the fast deployment of many SOLIOT servers is possible. All instances have their own hostname and thereby act as distinguishable nodes in the evaluation network. Currently, all instances have been equipped with one Thing Description representing one single device. As SOLID, and therefore also SOLIOT, maintains the resources as files on the local disk instead of keeping it in its local RAM, it is assumed that more resources would not significantly affect the performance behavior.

Each instance has an allocated disk space of 8 GB, which limits in no test scenario have been reached. The complete identical setting was used for SOLID reference servers as baselines. Both the SOLIOT and SOLID test instances asynchronously report information on incoming and outgoing messages to one central Web server. This server collects and persists the performance results, which serve as the inputs for the following assessments.

The instances without data files require around 257 MB, where most (243 MB) is used for external libraries. The exemplary Digital Twin consists of 32 Turtle files (88 KB), with in total 376 triples. The container structure, reflecting in many different files and folders also on the disk, increases the amount of required storage space. However, the management of each RDF subject in one file simplifies the discovery look ups and lets clients easier the relations more easily.

The available memory (0.5 and 1.0 GB) and computing power of the EC2 Micro and Nano instances are sufficient for all tests. A memory usage above 50% of the available amount was never observed. Still, the computing power of the host is challenged by the prototype. CPU usages around 80% and higher indicate a bottleneck here.

The use case scenario has been implemented through the architecture shown in Figure 8. Tables 8 and 9 for one SOLIOT and SOLID instances being requested simultaneously by each 1 and 10 clients, and a cluster of ten instances, respectively. The latency has been measured as perceived at the client itself but also at the server endpoint to show the influence of the network. Please note that the whole setup does not measure the parsing and of and from the Internet socket. This typical bottleneck is included in the network latency as it could not be influenced but also not reliably measured.

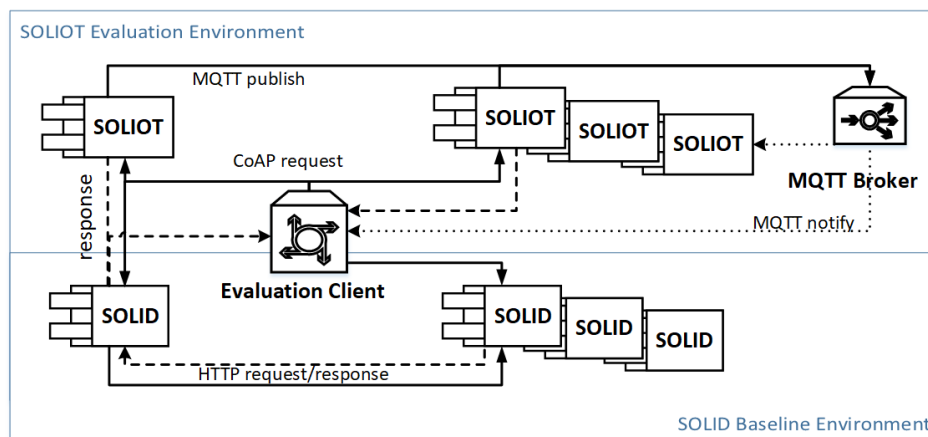


Figure 8. Information flow in the evaluation setting. The use case is implemented with SOLIOT (top) and equivalently with plain SOLID instances (bottom).

Table 8. Performance measures for one distinct SOLID and SOLIOT instance with 1 and 10 clients requesting in parallel.

Criteria	SOLID, 1 Cli.	SOLIOT, 1 Cli.	SOLID, 10 Cli.	SOLIOT, 10 Cli.	Setting
Start Time	1109 s (on average of 7 runs, variance: 4.53)	1580 s (on average of 35 runs, variance: 0.091)			(column denotes
Disk Size	225 MB	257 MB			requests/client)
GET @Client	224 ms	123 ms	242 ms	132 ms	1000 requests (SOLID: HTTP, SOLIOT: CoAP)
GET @Server	10 ms	20 ms	19 ms	18 ms	
Bandwidth GET	184 Bytes	338 Bytes	184 Bytes	338 Bytes	
Bandwidth Resp.	2845 Bytes	1988 Bytes	2845 Bytes	1988 Bytes	
POST @Client	215 ms	89 ms	284 ms	80 ms	1000 requests (SOLID: HTTP, SOLIOT: CoAP)
POST @Server	7 ms	11 ms	9 ms	9 ms	
Bandwidth POST	828 Bytes	908 Bytes	828 Bytes	908 Bytes	
Bandwidth Resp.	772 Bytes	0 Bytes	772 Bytes	0 Bytes	
PUT @Client	221 ms	72 ms	287 ms	80 ms	1000 requests (SOLID: HTTP, SOLIOT: CoAP)
PUT @Server	2 ms	15 ms	4 ms	15 ms	
Bandwidth PUT	810 Bytes	980 Bytes	810 Bytes	980 Bytes	
Bandwidth Resp.	704 Bytes	0 Bytes	704 Bytes	0 Bytes	
DELETE @Client	226 ms	64 ms	283 ms	72 ms	1000 requests (SOLID: HTTP, SOLIOT: CoAP)
DELETE @Server	4 ms	9 ms	5 ms	10 ms	
Bandwidth DEL.	195 Bytes	406 Bytes	195 Bytes	406 Bytes	
Bandwidth Resp.	942 Bytes	250 Bytes	942 Bytes	250 Bytes	
Publish (MQTT)					
Latency Client	–	0.11 ms	–	0.08 ms	200 events
Bandwidth Pub.	–	675 Bytes	–	675 Bytes	(SOLIOT: MQTT)

Table 9. Performance measures for 10 instances each with 1 and 10 parallel requesting clients.

Criteria	SOLID, 1 Cli.	SOLIOT, 1 Cli.	SOLID, 10 Cli.	SOLIOT, 10 Cli.	Setting
GET @Client	184 ms	116 ms	205 ms	132 ms	1000 requests (SOLID: HTTP, SOLIOT: CoAP) per instance and cli.
POST @Client	202 ms	60 ms	252 ms	81 ms	
PUT @Client	202 ms	62 ms	252 ms	82 ms	
DELETE @Client	202 ms	59 ms	250 ms	79 ms	

Table 10 provides a comparison on the used bandwidth for each request. This sequence has been 100 times by each client. One can see that typically the request size of the CoAP and HTTP messages are quite similar, while the response deviates significantly. One reason are the rich headers provided by the SOLID endpoint. Loosing this kind of information is a significant drawback of the current state of the IoT protocol mapping. Furthermore, the MQTT publishing behavior only reflects the actual publishing party. The receiving clients or the performance of the MQTT message broker have not been regarded. Another limitation of this evaluation approach is the limited representativeness of the used demo data. The Digital Twin model has been aligned with an existing product, unrelated to the SOLIOT developments. Still, a representative Digital Twin test bed would further increase the informative value of the experiments.

Table 10. Interaction sequence for the evaluation setting. Each iteration was executed 10 times (in total 100 CoAP and 20 MQTT interactions) per test instance. Presented are average measures per request.

Resource	SOLIOT Interaction	Size (bytes)	Resp. Size	SOLID Interaction	Size (bytes)	Resp. Size
/th[...]/maximumTemperature *	CoAP get	398	641	HTTP GET	204	1495
/things/[...]/currentTemperature	CoAP get	398	705	HTTP GET	204	1556
/things/3S7PM0CP4BD/120636/	CoAP get	308	7067	HTTP GET	161	8853
/things/	CoAP get	109	2426	HTTP GET	142	3324
/things/[...]/testTemperature	CoAP put	980	0	HTTP PUT	810	704
/things/[...]/testTemperature	pub: updated	675	–	HTTP PUT	810	704
/things/[...]/testTemperature	CoAP get	406	372	HTTP GET	203	1218
/things/[...]/testTemperature	CoAP delete	406	128	HTTP DELETE	190	627
/things/[...]/	CoAP post	980	0	HTTP POST	828	772
/things/[...]/	pub: created	675	–	HTTP POST	828	772
/things/[...]/testTemperature	CoAP get	406	960	HTTP GET	200	1212
/things/[...]/testTemperature	CoAP delete	406	128	HTTP DELETE	187	666

* complete URL for the first entry: <coap://18.157.197.66:5683/things/3S7PM0CP4BD/120636/2018-10-24T01-22-30-866Z/maximumTemperature>.

Comparing the results of the SOLIOT prototype, one can state a significant reduction in necessary transferred bytes but also in execution time. That is noteworthy regarding the fact that the handling of IoT requests takes longer than their HTTP counterparts (right side of columns in Tables 8 and 9, rows '@Server'). The implication is that the reduced data size results in a higher network speed. Furthermore, the SOLIOT prototype heavily relies on the core SOLID code. It mainly wraps the IoT endpoints but calls the underlying LDP functions in the same way as the SOLID baseline server does. Having a native integration directly on the persisting functions should further increase its speed.

Figure 9 further supports this insight. One can easily see the speed advantage. The interested reader may note the marked outlier for one CoAP get request. This one might be caused through the still at some times unreliable behavior of the SOLIOT prototype. Another relevant insight is the difference in the behavior of GET requests for both SOLID (faster than the others) and SOLIOT (slower), an effect which requires further investigations. All collected measures together with the resources to reproduce the measurements are provided publicly [55].

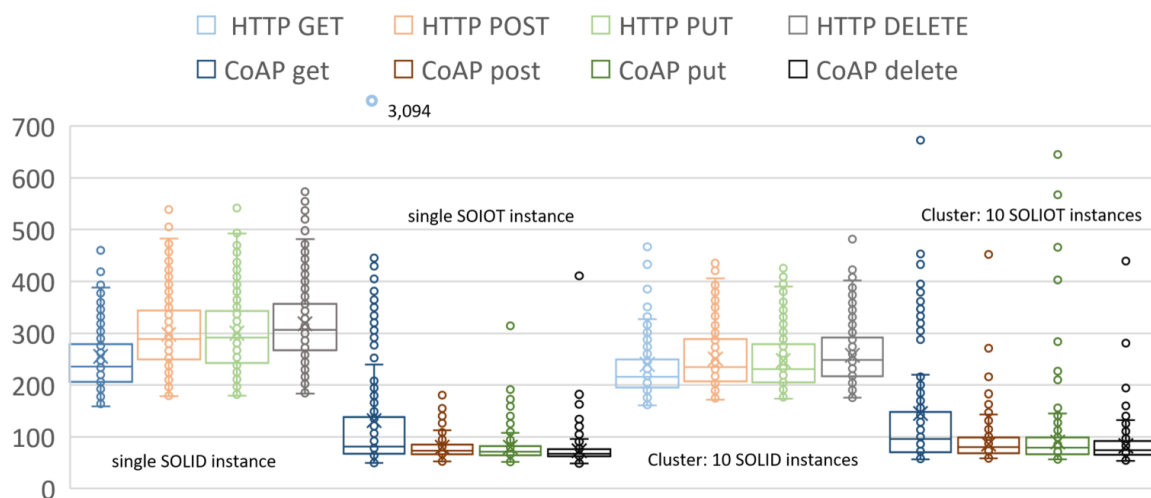


Figure 9. Variance of measures presented in Tables 8 and 9 (in milliseconds, 10 clients requesting each instance).

9. Conclusions and Outlook

This paper presents SOLIOT, an approach to merge concepts following Linked Data Platform and SOLID patterns aligned with IoT requirements. The respective capabilities and requirements have been discussed according to functional, non-functional, protocol- and data-related aspects. The basic functionality is shown through a prototypical server implementation, together with the provisioning of Web and IoT protocol endpoints, discovery mechanisms and how self-descriptive representations can model a Digital Twin.

SOLIOT outlines how the latest developments from the web community can be transferred to IoT challenges. The solely resource-driven view on the representation simplifies both the consumption and interaction with the thereby supported Digital Twins, and allows visiting clients to directly understand the Digital Twin itself but also helps developers and system integrators to adjust their workflows and APIs. This is achieved by the reduction of interaction patterns together with the self-descriptive nature of all involved concepts, promising a decrease of the overall integration complexity and increasing maintainability. The Digital Twin outlines both its content, interaction patterns and authentication model to the client. In addition, this is not only the case for the consuming applications, usually with higher computational power, but also for the restricted devices acting as the information sources.

The proposed approach still holds significant gaps. While the most crucial interaction patterns have been mapped, the true potential of the SOLID to IoT approach demands the complete coverage of all specified discovery mechanisms. The IoT mappings do not yet cover all headers and response mechanisms of a full-sized SOLID solution. In addition, the identity and authentication potentials of WebIDs are not sufficiently reflected at this stage. This is especially relevant, as a decentral yet reliable and at the same time simple to implement identity mechanism is essential for any scalable IoT approach. The core challenge is certainly the proper treatment of a physical Thing and its Digital Twin moving across networks, organizations and life cycles. The thereby created challenges towards interoperability, legislative obligations, ownership rights etc. are still not sufficiently understood.

Nevertheless, the general feasibility of SOLIOT is demonstrated through a CoAP and MQTT binding, more powerful protocol stacks have not been tested yet. In particular, the OPC UA specifications define not only a communication protocol but also an elaborated interaction and information model. Respective endpoint adapters, data lifting and lowering modules, and appropriate mappings for authorization need to be developed to examine the characteristics further.

The provided analysis showed a series of relevant but still unaddressed requirements. As directly visible though the significant number of gaps shown in the mapping tables, more work is necessary

for a complete picture. Even though one can certainly question whether each IoT concept needs to address all outlined requirements, the awareness of their relevance and how they impact the desired solution is necessary. This work should be regarded also as an inspiration for further approaches by stating the current possibilities but also limitations.

At the core of SOLIOT is the translation of established and clean interaction patterns from the Web into the IoT where all necessary information is explicitly described and available for look ups. The further decoupling of IoT communication will introduce a new flexibility and dynamics into device-to-device networks. An ongoing integration of previously separated and segregated OT networks with the global Internet network can be stated. This requires a shared understanding on how ‘Things’ need to interact and how they need to be described. SOLIOT is comprising the combined practices of the Semantic Web community and intends to outline one contribution to this challenge.

Author Contributions: Conceptualization, S.R.B.; methodology, M.M.; software, S.R.B.; visualization, S.R.B. and S.L.; writing—original draft preparation, S.R.B.; writing—review and editing: M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the German Federal Ministry of Education and Research through the project “Industrial Data Space Plus” (grant number 01IS17031), the European Union H2020 project “BOOST4.0” (780732) and the Fraunhofer Cluster of Excellence “Cognitive Internet Technologies” (CCIT).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sambra, A.V.; Mansour, E.; Hawke, S.; Zereba, M.; Greco, N.; Ghanem, A.; Zagidulin, D.; Abounaga, A.; Berners-Lee, T. *Solid: A Platform for Decentralized Social Applications Based on Linked Data*; Technical Report; 2017; Available online: http://emansour.com/publications/paper/solid_protocols.pdf (accessed on 10 June 2020).
2. Pfrommer, J.; Grüner, S.; Goldschmidt, T.; Schulz, D. A common core for information modeling in the Industrial Internet of Things. *at-Automatisierungstechnik* **2016**, *64*, 729–741. [CrossRef]
3. Raymor, B.; Coppen, R.; Banks, A.; Briggs, E.; Borgendale, K.; Gupta, R. MQTT Version 5.0; OASIS Committee Specification 02. 2018. Available online: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/cs02/mqtt-v5.0-cs02.html> (accessed on 10 June 2020).
4. Shelby, Z.; Hartke, K.; Bormann, C. The Constrained Application Protocol (CoAP). RFC 7252. 2014. Available online: <https://tools.ietf.org/html/rfc7252> (accessed on 30 April 2020).
5. Tao, F.; Cheng, J.; Qi, Q.; Zhang, M.; Zhang, H.; Sui, F. Digital twin-driven product design, manufacturing and service with big data. *Int. J. Adv. Manuf. Technol.* **2017**, *94*, 3563–3576. [CrossRef]
6. Mrissa, M.; Médini, L.; Jamont, J.P.; Le Sommer, N.; Laplace, J. An avatar architecture for the web of things. *IEEE Internet Comput.* **2015**, *19*, 30–38. [CrossRef]
7. Malakuti, S.; van Schalkwyk, P.; Boss, B.; Sastry, C.R.; Runkana, V.; Lin, S.W.; Rix, S.; Green, G.; Baechle, K.; Nath, S.V. Digital Twins for Industrial Applications. Industrial Internet Consortium (IIC), White Paper. **2020**. Available online: <https://www.iiconsortium.org/stay-informed/digital-twins-for-industrial-applications.htm> (accessed on 14 April 2020).
8. Glaessgen, E.; Stargel, D. The digital twin paradigm for future NASA and US Air Force vehicles. In Proceedings of the 2012 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference and 20th AIAA/ASME/AHS Adaptive Structures Conference, Honolulu, HI, USA, 23–26 April 2012; p. 1818.
9. Lin, S.W.; Miller, B.; Durand, J.; Joshi, R.; Didier, P.; Chigani, A.; Torenbeek, R.; Duggal, D.; Martin, R.; Bleakley, G.; et al. Industrial Internet Reference Architecture. *Ind. Internet Consort. (IIC) Tech. Rep.* **2015**. Available online: <https://www.iiconsortium.org/IIRA.htm> (accessed on 7 June 2020).
10. Bader, S.; Barnstedt, E.; Bedenbende, H.; Billmann, M.; Boss, B.; Braunmandl, A.; Clauer, E.; Deppe, T.; Diedrich, C.; Flubacher, B.; et al. *Details of the Asset Administration Shell Part 1*; Technical Report, Plattform Industrie 4.0; ZVEI: Frankfurt, Germany, 2019.

11. D'Agostino, N.; Annacondia, E.; Bentkus, A.; Bianchi, G.; Briant, J.; Heidel, R.; Hoffmeister, M.; Lambole, P.; Lensi, R.; Rossi, G.; et al. *The Structure of the Administration Shell: Trilateral Perspectives from France, Italy and Germany*; Alliance Industrie du Futur: Courbevoie, France; Ministero dello Sviluppo Economico: Roma, Italy; Plattform Industrie 4.0: Berlin, Germany, 2018. Available online: https://www.de.digital/DIGITAL/Redaktion/EN/Publikation/the-structure-of-the-administration-shell.pdf?__blob=publicationFile&v=3 (accessed on 16 June 2020).
12. Otto, B.; Steinbuss, S.; Teuscher, A.; Lohmann, S.; Auer, S.; Bader, S.; Bastiaansen, H.; Bauer, H.; Birnstil, P.; Böhmer, M. *Reference Architecture Model for the Industrial Data Space*; version 3.0; International Data Spaces Association: Dortmund, Germany, 2019.
13. International Electrotechnical Commission. *OPC Unified Architecture—Part 1: Overview and Concepts*; IEC TR 62541-1:2016; International Electrotechnical Commission: Geneva, Switzerland, 2016.
14. Volz, F.; Stojanovic, L.; Lamberti, R. An Industrial Marketplace—The Smart Factory Web Approach and Integration of the International Data Space. In Proceedings of the 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), Helsinki, Finland, 22–25 July 2019; Volume 1, pp. 714–720.
15. Malakuti, S.; Schmitt, J.; Platenius-Mohr, M.; Grüner, S.; Gitzel, R.; Bihani, P. A Four-Layer Architecture Pattern for Constructing and Managing Digital Twins. In *European Conference on Software Architecture*; Springer: Cham, Switzerland, 2019; pp. 231–246.
16. Perzylo, A.; Profanter, S.; Rickert, M.; Knoll, A. OPC UA NodeSet Ontologies as a Pillar of Representing Semantic Digital Twins of Manufacturing Resources. In Proceedings of the 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Zaragoza, Spain, 10–13 September 2019; pp. 1085–1092.
17. Schiekofer, R.; Grimm, S.; Brandt, M.M.; Weyrich, M. A formal mapping between OPC UA and the Semantic Web. In Proceedings of the 2019 IEEE 17th International Conference on Industrial Informatics (INDIN), Helsinki, Finland, 22–25 July 2019; pp. 33–40.
18. Jammes, F.; Smit, H. Service-Oriented Paradigms in Industrial Automation. *IEEE Trans. Ind. Inform.* **2005**, *1*, 62–70. [\[CrossRef\]](#)
19. Weyer, S.; Meyer, T.; Ohmer, M.; Gorecky, D.; Zühlke, D. Future Modeling and Simulation of CPS-based Factories: An Example from the Automotive Industry. *IFAC Pap.* **2016**, *49*, 97–102. [\[CrossRef\]](#)
20. Bader, S.R.; Maleshkova, M. Virtual representations for an iterative IoT deployment. In Proceedings of the Companion Proceedings of the Web Conference 2018, Lyon, France, 23–27 April 2018; pp. 1887–1892.
21. Maleshkova, M.; Philipp, P.; Sure-Vetter, Y.; Studer, R. Smart Web Services (SmartWS)—The Future of Services on the Web. *arXiv* **2019**, arXiv:1902.00910.
22. Kovatsch, M.; Matsukura, R.; Lagally, M.; Kawaguchi, T.; Toumura, K.; Kajimoto, K. Web of Things (WoT) Architecture. W3C Recommendation. 2020. Available online: <https://www.w3.org/TR/wot-architecture/> (accessed on 16 April 2020).
23. Web of Things Community Group. Web of Things Binding Templates. Available online: <https://github.com/w3c/wot-binding-templates/blob/master/ontology/> (accessed on 10 June 2020).
24. Bermudez-Edo, M.; Elsaleh, T.; Barnaghi, P.; Taylor, K. IoT-Lite: A lightweight semantic model for the Internet of Things. In Proceedings of the Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress, Toulouse, France, 18–21 July 2016; pp. 90–97.
25. Agarwal, R.; Fernandez, D.G.; Elsaleh, T.; Gyrard, A.; Lanza, J.; Sanchez, L.; Georgantas, N.; Issarny, V. Unified IoT ontology to enable interoperability and federation of testbeds. In Proceedings of the 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), Reston, VA, USA, 12–14 December 2016; pp. 70–75.
26. Gyrard, A.; Atemez, G.; Bonnet, C.; Boudaoud, K.; Serrano, M. Reusing and unifying background knowledge for internet of things with LOV4IoT. In Proceedings of the 2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud), Vienna, Austria, 22–24 August 2016; pp. 262–269.
27. Loseto, G.; Ieva, S.; Gramegna, F.; Ruta, M.; Scioscia, F.; Di Sciascio, E. Linking the web of things: LDP-CoAP mapping. *Procedia Comput. Sci.* **2016**, *83*, 1182–1187. [\[CrossRef\]](#)
28. Binz, T.; Breiter, G.; Leyman, F.; Spatzier, T. Portable cloud services using toasca. *IEEE Internet Comput.* **2012**, *16*, 80–85. [\[CrossRef\]](#)
29. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge Computing: Vision and Challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [\[CrossRef\]](#)

30. Adolphs, P.; Berlik, S.; Dorst, W.; Friedrich, J.; Gericke, C.; Hankel, M.; Heidel, R.; Hoffmeister, M.; Mosch, C.; Pichler, R.; et al. DIN SPEC 91345: Reference Architecture Model Industrie 4.0. 2016. Available online: <https://webstore.ansi.org/Standards/DIN/DINSPEC913452016> (accessed on 7 June 2020).
31. Mahmud, R.; Srirama, S.N.; Ramamohanarao, K.; Buyya, R. Quality of Experience (QoE)-aware placement of applications in Fog computing environments. *J. Parallel Distrib. Comput.* **2019**, *132*, 190–203. [CrossRef]
32. Lazarescu, M.T. Design of a WSN platform for long-term environmental monitoring for IoT applications. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2013**, *3*, 45–54. [CrossRef]
33. Bassi, A.; Bauer, M.; Fiedler, M.; Kramp, T.; Kranenburg, R.V.; Lange, S.; Meissner, S. *Enabling Things to Talk: Designing IoT solutions with the IoT Architectural Reference Model*; Springer: Berlin/Heidelberg, Germany, 2013.
34. Kavianpour, S.; Shanmugam, B.; Azam, S.; Zamani, M.; Narayana Samy, G.; De Boer, F. A Systematic Literature Review of Authentication in Internet of Things for Heterogeneous Devices. *J. Comput. Networks Commun.* **2019**, *2019*, 5747136. [CrossRef]
35. Kaebisch, S.; Kamiya, T.; McCool, M.; Charpenay, V.; Kovatsch, M. Web of Things (WoT) Thing Description. W3C Recommendation. 2020. Available online: <https://www.w3.org/TR/wot-thing-description/> (accessed on 16 April 2020).
36. IoT-A. IoT-A Unified Requirements List. 2016. Available online: <https://web.archive.org/web/20160322053934/http://www.iot-a.eu/public/requirements> (accessed on 7 June 2020).
37. SOLID Project Team. Web Access Control. Specification Draft. 2020. Available online: <https://github.com/solid/web-access-control-spec> (accessed on 10 June 2020).
38. SOLID Project Team. Solid Specification Draft. 2020. Available online: <https://github.com/solid/solid-spec> (accessed on 10 June 2020).
39. Eitel, A.; Jung, C.; Kühnle, C.; Bruckner, F.; Brost, G.; Birnstill, P.; Nagel, R.; Bader, S. Usage Control in the IDS. Whitepaper. 2019. Available online: https://www.internationaldataspaces.org/wp-content/uploads/2019/11/Usage-Control-in-IDS-V2.0_final.pdf (accessed on 26 May 2020).
40. Naik, N. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In Proceedings of the 2017 IEEE international systems engineering symposium (ISSE), Vienna, Austria, 11–13 October 2017; pp. 1–7.
41. RFC 6347. Datagram Transport Layer Security Version 1.2. 2012. Available online: <https://tools.ietf.org/html/rfc6347> (accessed on 29 April 2020).
42. Han, Q.; Zhang, Y.; Li, H. Efficient and robust attribute-based encryption supporting access policy hiding in Internet of Things. *Future Gener. Comput. Syst.* **2018**, *83*, 269–277. [CrossRef]
43. Charpenay, V.; Lefrançois, M.; Villalon, M.; Käbis, M. Thing Description (TD) Ontology. W3C Editor's Draft, 2020. Available online: <https://www.w3.org/2019/wot/td#> (accessed on 10 June 2020).
44. Haller, A.; Janowicz, K.; Cox, S.; Phuoc, D.; Taylor, K.; Lefrançois, M.; Atkinson, R.; García-Castro, R.; Lieberman, J.; Stadler, C. Semantic Sensor Network Ontology. W3C Recommendation, 2017. Available online: <https://www.w3.org/TR/vocab-ssn/> (accessed on 10 June 2020).
45. Technical Committee Smart Machine-to-Machine Communications. Smart Appliances Reference Ontology. ETSI Technical Specification, 2017. Available online: <https://ontology.tno.nl/saref/> (accessed on 10 June 2020).
46. Bader, S.R.; Maleshkova, M. The Semantic Asset Administration Shell. In *International Conference on Semantic Systems*; Springer: Cham, Switzerland, 2019; pp. 159–174.
47. Wallace, E.; Kiritsis, D.; Smith, B.; Will, C. The Industrial Ontologies Foundry proof-of-concept project. In *IFIP International Conference on Advances in Production Management Systems*; Springer: Cham, Switzerland, 2018; pp. 402–409.
48. eCI@ss. Classification and Product Description. eCI@ss Major Release 11.0, 2019. Available online: <https://www.eiclass-cdp.com/portal/info.seam> (accessed on 10 June 2020).
49. Souza, V.; Cruz, R.; Silva, W.; Lins, S.; Lucena, V. A Digital Twin Architecture Based on the Industrial Internet of Things Technologies. In Proceedings of the 2019 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 11–13 January 2019; pp. 1–2.
50. Boyes, H.; Hallaq, B.; Cunningham, J.; Watson, T. The industrial internet of things (IIoT): An analysis framework. *Comput. Ind.* **2018**, *101*, 1–12. [CrossRef]
51. Bader, S. SOLIOT Reference Implementation. GitHub Project, 2020. Available online: <https://github.com/sebbader/SolIoT> (accessed on 10 June 2020).

52. Collina, M. Node CoAP. GitHub Project, 2020. Available online: <https://github.com/mcollina/node-coap> (accessed on 10 June 2020).
53. Rudd, A.; Collina, M.; Agor, M.; Buntsevich, S. MQTT.js NodeJS Library. GitHub Project, 2020. Available online: <https://github.com/mqttjs/MQTT.js> (accessed on 10 June 2020).
54. Eclipse Foundation. Mosquitto—An Open Source MQTT Broker. Eclipse Project, 2020. Available online: <https://mosquitto.org/> (accessed on 10 June 2020).
55. Bader, S. SOLIOT Evaluation Tool and Results. GitHub Project, 2020. Available online: https://github.com/sebbader/soliot_evaluation (accessed on 26 May 2020).



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).