A Modular Distributed IoT Service Discovery

Marc-Oliver Pahl and Stefan Liebald
Technical University of Munich
{pahl, liebald}@net.in.tum.de

***Abstract*—The Internet of Things (IoT) consists of distributed entities that federate to implement use cases. Driven by its applications, distributed IoT services federate dynamically to deliver complex functionality.**
A fundamental requirement for service composition is discovery. The IoT requires a semantically-rich, dynamically extensible, low complexity service discovery mechanism. Existing mechanisms fail in delivering this functionality.
We present a distributed modular directory of service properties. It can be extended at runtime. Combined our so-called search provider directories form an IoT ontology. With the novel modularization, our simple directories mash-up to a complex ontology. We introduce a query federation mechanism. Based on first-order logic it enables mapping complex semantic queries on the simple search providers. We evaluate our prototype regarding latency and usability.

I. Introduction

The Internet of Things (IoT) consists of distributed entities that provide functionality such as sensing light intensity, or opening windows. Different from previous automation systems, the IoT implements use cases as software services. IoT services manage other services that either manage hardware or software [1]. IoT hardware devices are Cyber-Physical Systems, as they run software to offer certain functionality such as remote control. All management in the IoT therefore happens in-between software services.

While early IoT systems implement functionality in monolithic large services, more recently researchers started to look at modular, Service-Oriented Architectures (SOA) for managing the distributed IoT services [2]–[4]. Modular service-oriented IoT systems are more flexible as they can dynamically reorganize to implement different use cases on the same hardware.

The resulting dynamic linking of services requires compatible interfaces. Different solutions for such interfaces have been proposed, e.g. using a data-centric approach [3], [5]. A fundamental requirement for dynamic linking of services at runtime is the availability of suitable discovery mechanisms.

IoT applications that implement use cases are diverse. Once suitable middleware such as [6] is established, similar to the smartphone App development the limit of an IoT application is the imagination of the developer and the locally available IoT hardware.

Different IoT use cases require different properties to identify suitable services to mash-up. A maintenance IoT application may for instance be interested in types, locations, and ownerships of other services to run queries such as, "give me all service end points `of type` light, `owned by` John Doe, `located in` building F". The discovery mechanisms typically used in deployed IoT systems such as UPnP [7], or DNS-based discovery [8] do not offer such a *semantically-rich* service discovery.

What should be discoverable does not depend on the available hardware devices but on the use cases that should be implemented. Thus, it depends on the software services running within an IoT site. As software can dynamically be added to an IoT site, the discovery mechanism has to be *dynamically extensible*. Following the above example, if an advanced management service is deployed that needs to identify lamps of a certain manufacturer, another predicate for the discovery, e.g. `manufactured by` is needed.

While commercially deployed solutions offer basic discovery only, research projects offer semantically richer discovery. The richest form of discovery is an ontology [9]. The information model of an ontology such as the Web Ontology Language (OWL) model allows expressing diverse metadata of a service.

However, a problem is that ontologies are typically not extensible at runtime, that they are slow to parse, and that they are complex. Especially in microservice architectures [2]–[4], mash-ups of services are frequent. Therefore, the discovery has to be *low latency*.

Since the IoT requires a large amount of entities to be modeled, and modeling becomes part of the software development in data-centric architectures [2], the modeling process has to be simple. Ideally it can be performed in a crowdsourced way [5], [10].

In a service-oriented IoT we need a discovery that enables:

- querying *semantically-rich* service descriptions,
- *dynamically extending* the service descriptions at runtime for flexible discovery, and
- discoveries with *low latency*,
- while having *low complexity* for crowdsourced use.

Existing mechanisms fail in delivering this functionality (section IV). Therefore, we introduce a distributed modular directory of service properties that can be extended at runtime (section II). The directory modules are called *search providers*. The combined search providers form an ontology. The modularization enables the use of simple directories to mash-up a complex ontology. We introduce a query federation mechanism based on first-order logic that allows mapping complex search queries to simple distributed search provider modules. We evaluate our prototype regarding latency and usability (section III).

Preprint from s2labs.org

II. A Dynamically Extensible Ontology

As written in the introduction, the introduction of an advanced service discovery mechanism requires the presence of a suitable middleware for enabling dynamic service coupling at runtime. Our Virtual State Layer (VSL) middleware provides such functionality [2], [6].

VSL services couple over a distributed tuple space that organizes itself. The tuples are accessed mainly via the methods `get`, `set`, and `subscribe` [2]. The tuple space is implemented as a peer-to-peer system of distributed, self-organizing so-called Knowledge Agents (KA).

IoT services connect to a KA on a local or well reachable compute node. This could be any IoT controller with enough computational resources [11]. Each KA manages all data for its directly connected services. It provides access to the data of any other service that is connected via its responsible KA.

The data-centric VSL design enables implementing interesting properties such as security-by-design [12]–[14]. In [5] we showed how it is possible to map regular function calls to VSL data accesses via virtual data nodes. In this work we use this mechanism to couple the different *search provider directories* in our *meta search provider*.

Each data item in the VSL has a globally unique address, a Unified Resource Identifier (URI): *vsl://siteID/kaID/serviceID/service/specific/nodes*. It consists of a unique identifier per IoT site, a site-locally unique KA identifier, a node-locally service identifier, and a subtree that represents the digital twin of an IoT service that potentially interfaces an IoT device.

In case hardware is interfaced, the digital twin represents the functionality of the hardware, e.g. a dimmable light as shown in listing 1. It contains different properties another service can interact with, e.g. by setting the `isOn` to *true* for switching the light on [5].

```
<dimmableLight>
  <isOn type="/derived/boolean" />
  <intensity type="/derived/intensity" />
</dimmableLight>
```

Listing 1: VSL type ”dimmableLight” data model.

Listing 1 shows the described VSL data model in XML notation. Each node has a type tag. The information model [15] provides typing (is-a) and composition (has-a) as ontology predicates [9]. The `type` tag represents the inheritance relation *is-a*. The field values refer to type specifications that are defined in a global directory [5]. The hierarchical structure of nested data nodes represents the containment relation *has-a*. All VSL data nodes together form a Directed Acyclic Graph (DAG) of nodes [6].

With its so-called Virtual Nodes, the VSL offers a way to couple services synchronously over the descriptive blackboard data structures [3]. By accessing its data, e.g. *vsl://siteID/search/type/derived/intensity*, a callback function within the remote service is invoked and can directly return a value. Parameters can be passed at a data request similar to calling a dynamic website, e.g. the data type identifiers “/derived/intensity”.

In this work we use VSL Virtual Nodes to enable a seamless plug-in of search providers at run time. Depending on the called address they return for instance a list of service URIs to a given type. See fig. 2. The resulting dynamic extensibility of the discovery mechanism is relevant for our proposed approach as IoT scenarios require dynamic extension of the system.

Though using the VSL as base for illustrating our work, the only requirement of the target system architecture to implement our methodology is that the entities to be discovered have a unique identifier that can be returned by a search query.

A. Ontologies

Ontologies are a semantically-rich formalism to describe entities. An ontology describes the relevant properties of a domain. The properties-of, and relationships-between entities can be formally described using ontology languages [9].

In a computer, semantic aspects are typically represented as triples: $\langle subject, \text{predicate}, object \rangle$. An example is, $\langle light\_23, \text{owned by}, JohnDoe \rangle$. As the URIs describe service resources, they become the *subjects* of our triples. The *objects* depend on the *predicates*, e.g. *room names* might be suitable classifiers for indoor *locations*.

The following triples describe properties of services:

⟨URI,	predicate, object⟩
⟨/a/b/sens23,	of-type, light⟩
⟨/a/b/sens23,	located-in, livingRoom⟩

The available ontology *predicates* define, which semantic relationships can be expressed in a system. In classic ontologies such predicates are *fixed*. This is also the case in widespread network management systems where the information models define the predicates. Examples are SNMP MIBs [16] and NETCONF configurations [17].

Ontologies are optimal for the IoT as they are very expressive. However, they are *complex* to use, and slow to process. Network management information models in contrast are *easy-to-use* and *fast-to-process* but *not expressive enough* for the IoT domain [5]. As ontologies are too complex for crowdsourced mass development, and network management information models not expressive enough for the IoT domain, we propose a hybrid of both in the following.

B. Search Provider

In most existing IoT discovery approaches, look-ups are driven by resource self-descriptions [18]: the existing infrastructure of hardware and software exposes certain *pre-defined* properties that can be used for service discovery. Different to classic management scenarios where the required properties are known at design time, the IoT is driven by the Pervasive Computing use cases it implements [19]. The emergence of unforeseen scenarios is an integral part of this domain.

We therefore provide a novel approach for service discovery in the IoT. Instead of having fixed semantic properties with the discoverable entities, or in central directories, we propose

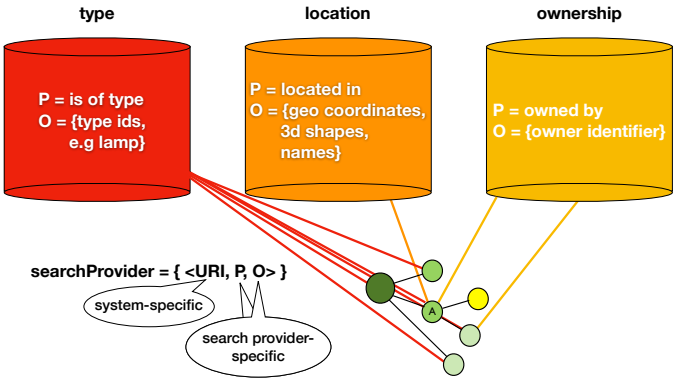

Fig. 1: Search providers implementing different predicates and their object directories.

distributed independent directories that only allow a lookup for certain semantic predicates and their properties. These directories are called *search providers*.

To enable complex queries with multiple predicates that require requests to multiple *search providers*, we introduce a mechanism for automatically federating the queries to the distributed directories using first order logic. We call this functionality *meta search provider*.

Each *search provider* implements a semantic predicate. In the example above the predicates are `of-type`, *located-in*, and *owned-by*. The objects are predicate-specific. Figure 1 shows the three search providers and suitable ontology *objects*.

The bottom right shows a VSL data model instance. All VSL data nodes only have the VSL built-in `of-type` semantic. Additional properties are added by offering a lookup of the unique node URI using each *search provider*'s *predicates*, and *object* database. Thus, the context metadata of a data node is distributed over all *search providers*.

This modularization brings the advantage that new *predicates* can simply be added to an IoT site by adding the corresponding *search providers*. The costs are potentially increased *latency* of queries compared to querying a single comprehensive directory (section III).

In existing *resource-specific* approaches, service developers typically describe *limited* semantic properties following a taxonomy of properties. This happens at a *fixed point in time*, e.g. at development time or latest at deployment time. In contrast, our approach follows a *discovery requester-centric* approach: semantic discovery queries are executed by *dynamically added* search providers. As the search providers bring a set of properties, service properties can *dynamically* be added at any time, even after a service was started.

Assuming that an infrastructure for service deployment exists, the installation of new IoT services that have new discovery requirements can directly trigger the installation of the corresponding search providers. When installing a service that deals with *location* for coupling, e.g. for switching off lights within a certain geo-spatial area only, a location search provider can be installed on demand. Consecutively, queries for `located-in` become possible.

Search providers encapsulate the entire logic associated with their implemented predicate. To feed useful data into the search provider's *object* database, in our example the location provider offers a map-based interface for graphically positioning IoT services. Offering *user friendly* interfaces that can be used by the end-users of a smart space solves the problem of adding context metadata at runtime without the need for ontology experts.

At the same time the modularization keeps the *complexity of the data models low*. It also facilitates the implementation of each predicate as its domain of concern is clearly defined and limited.

We successfully tested our location provider approach within different environments with different users, showing that the approach of keeping concerns separate and limiting the complexity indeed enables even inexperienced users to provide relevant context to the IoT.

C. Meta Search Provider

The key to the proposed modularization of an IoT site's ontology is that each data node (service access point) has a unique identifier that can be returned with a query. Analog to relational databases [20], we use the unique data item identifier to connect the results of different *search provider* queries. To enable more complex queries over multiple predicates, such as the example with the lights and their locations and owners, we introduce so called *meta search providers*.

A fundamental concept of the VSL is modularizing functionality to microservices. Their unified interfaces make mashing up services simple. Consequently, it becomes possible to implement a service for *federating* queries to any set of search providers. With the mechanisms for late coupling of the VSL it becomes even possible to create a generic federation *meta search provider* that can federate all kinds of *search providers*.

For illustrating the approach, we provide a *first order logic* meta search provider. It combines search results from different search providers using first order logic. With the meta search service developers can query multiple search providers without having to implement any federation logic. This approach works as the common denominator of all search providers is their return value format: All search providers return a set of VSL service URIs.

Figure 2 illustrates the federation of the two search providers (A, B) using the logic operator `AND`. The meta search provider (C) implements `AND` by intersecting the two sets of VSL URIs, the individual search provider queries return.

Queries like the one with the type and the location that was described in the introduction become possible: `get /search/metaSearch/[(p=of-type, o=light) AND (p=located-in, o=livingRoom)]`. Via the parameters `p` for predicate, and `o` for object, the meta search provider can determine, which search provider to query.

The predicate `of-type` is built-in to the VSL KAs. It is also used to identify instances of other search providers. Figure 2 shows the VSL type of each search provider on top. It equals the implemented predicate from the query.

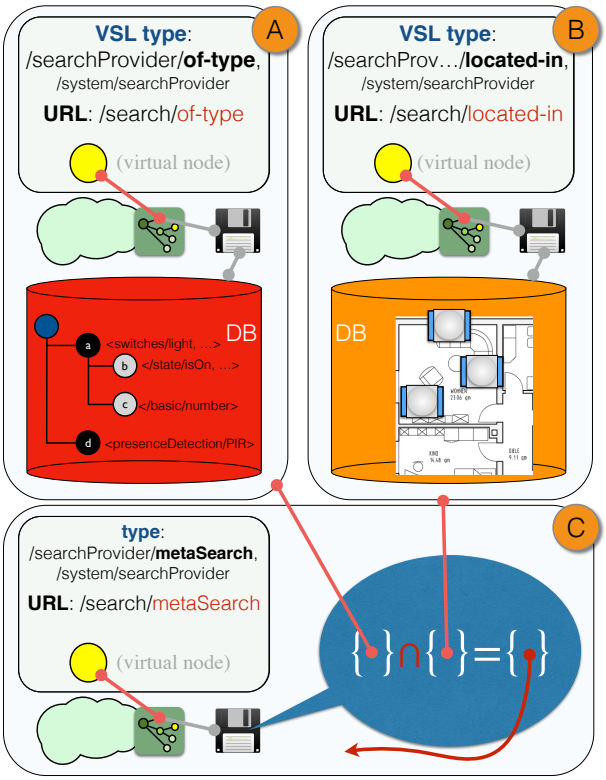

Fig. 2: Three example VSL search providers.

Using the late binding feature of the VSL, the meta search provider only has to search for all VSL nodes of type `/searchProvider/located-in` in case of the location search provider. It can then pick any of the resulting VSL search provider service instance addresses to send its query, e.g. `/[returnedVslUri]/livingRoom`.

The encapsulation of the search providers enables implementing a generic *divide and conquer* strategy for federated search queries. All search provider interfaces and return values are identical. The VSL REST interface [21] and the data type "/system/searchProvider" define the interface. The return value is a list of VSL URIs.

The meta search provider is fully agnostic of the used search providers. It simply identifies them using the predicate `p` and locates them via the built-in VSL type search. Then it hands the query object `o` over to them.

D. Reverse Search

The described approach works, as the additional semantic attributes that a search provider brings are only required at discovery time. However, for providing certain services it is also desired to know for instance the location or ownership of a data node / service that is interfaced via the VSL.

For this purpose, search providers offer a *reverse search*. Instead of passing the objects that determine the criteria for the forward search that returns the URIs of associated data nodes, a URI is passed to the reverse search. The result is a list of objects in the search provider's database that are associated with the given URI.

E. Security

Accessing meta data can be *security critical*. This includes the exposure of private data such as ownerships and locations. The VSL implements a role-based access model that grants access to data nodes only to those requesters that are of a certain role [12].

The VSL can implement security-by-design as it has full control over the inter-service communication. As search provider requests have well known return values, we parse the resulting URI lists and remove all nodes a requester has no access to. The same applies to reverse queries, where the object lists are locally filtered before a query is sent out.

With the described mechanism, the VSL enforces access control external to the search providers. This is important to us, as we want to enable third party developers to create search providers on the one hand, and on the other hand we do not want to reveal secret knowledge to insider attackers that have access to the VSL and its search providers.

III. Evaluation

Based on our Java VSL pilot we discuss the semantic richness and dynamic extensibility, measure the latency of federated queries with different amounts of search providers, and report the results of our qualitative user experiments.

A. Semantic Richness and Dynamic Extensibility

Ontologies are typically described via triples, <Subject, Predicate, Object> [9], [22], [23]. The VSL data nodes with their URIs are the subjects. The built-in predicates of the VSL information model are typing (is-a) and containment (has-a). They are implemented via `of type` metadata per node, and via providing a *hierarchical data structure*.

The objects of the `of type` search are the data type identifiers that are specified by the crowd in a Central Model Repository (CMR) [5]. The CMR thereby acts as continuously updated directory for developers to look up which objects they can program into their type searches.

With the introduced search providers, the VSL data models can be dynamically extended with *more predicates at runtime* (section II). The result is a *high level of expressiveness* without introducing the typical *complexity* of an ontology [9], [22], [23]. Each search provider only has to deal with one predicate only, facilitating the use significantly.

As shown in fig. 1, predicates and qualifying objects can be added via linking search providers to the data nodes. This implements a *modular ontology*. Using our modular approach provides *simpler* use than a full-blown ontology [23], and the distributed data can be *processed faster*.

B. Latency Measurements

The VSL is a distributed system. Data access happens either locally on the node a service resides on, or on a remote node. Remote coupling of VSL services has a higher latency than node-local coupling due to additional processing and network overhead.

IoT systems can have heterogeneous links, resulting in different latencies per link. However, there is no standard IoT setting and therefore, we assume equal latencies between all compute nodes. We evaluate the latency differences of local and remote search providers.

We use a GBps connection with standard computers for the measurement for the same reason: there is no standard IoT hardware setting. The computers did neither have any significant CPU, memory, or network load. Therefore, the results shown in fig. 3 represent the actual latencies of the operations without significant external limitations.

Our federation meta search provider enables formulating queries over multiple search providers by combining them with logical operators. For the evaluation we only use the operator "AND". Using other first order logic operators does not make a relevant computational difference and thereby has similar execution time.

We compare two strategies for federating queries: a (naive) approach that resolves all search queries sequentially, and a second approach that splits each complex query into independent sub queries that are resolved in parallel following the previously described divide and conquer strategy. Listing 2 shows actual VSL code for the sequential case of intersecting two search providers.

```
1  List<string> URIs1 = vsl.get("/search/"+
       predicate1+"/"+objects1);
2  List<string> URIs2 = vsl.get("/search/"+
       predicate2+"/"+objects2);
3  return URIs1.retainAll(URIs2); // AND
```

Listing 2: Sequentially intersecting two search providers.

For parallel queries we use a thread pool of size 5 for the different requests. The size is sufficient to always execute all query part requests in parallel.

One VSL KA overlay node and multiple different search providers run on each host. During the evaluation we send search queries from our probing service to the meta search provider. Our test queries require different amounts of local or remote search providers.

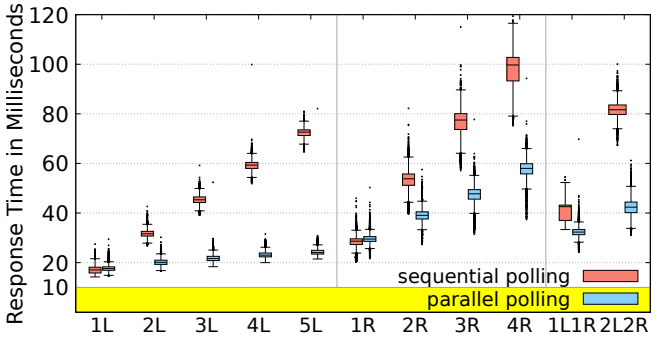

Fig. 3: Latency with 1-5 (L)ocal and (R)emote directories.

Figure 3 shows the results of different latency measurements with the median and the 0.25 percentiles. All measurements were repeated 10000 times. The outliers are those values that are above 1.5 times the InterQuartile Range.

The yellow area at the bottom shows an artificial latency for emulating a lookup and local computation. Having identical processing latency enables focusing on the latencies introduced by the proposed search provider federation.

The first 5 measurements (1L-5L) in the left part of fig. 3 show the results for combining up to 5 search providers running locally on the same VSL node. As expected the parallel requests (right, blue) are faster than the sequential (left, red) for requests to more than one search provider.

As expected, the sequential complex query processing latency grows linearly with the amount of search providers. For the parallel queries the latency increase is mainly caused by the computation needed to intersect (AND) the results (see listing 2), and a small overhead for the threading. This threading overhead also explains why sequential polling of 1 provider is slightly faster than with parallel polling.

The middle part of fig. 3 (1R-4R) shows the querying search providers on remote hosts. As expected, the remote access adds a basic latency and more variance. The steeper increase for the parallel remote requests is caused by our current VSL version having a bottleneck with communication.

Even with the VSL-caused additional latency, complex remote queries involving up to four predicates/ search providers are carried out in less than 80ms. Typically, about 300ms are still perceived as realtime. This leaves 220ms for further application logic when implementing perceived realtime IoT applications. With local search providers the added latency by the modularization gets negligible.

The right part of fig. 3 (L1R1,L2R2) shows the latency when mixing local and remote search providers. As expected, for the sequential requests, the times of the non-combined local and remote requests roughly sum up. For the parallel requests the remote providers dominate the resulting latency.

Overall the plots show that the approach has very low latencies and is therefore well suitable for IoT operations that appear realtime to the user. Compared to the state of the art with 450-600ms [24] our results are much better. At the same time our approach allows much more complex queries.

Our latency compares well with standard UDDI with 163ms [25] that has less semantic expression than our approach. OWL-S+UDDI has comparable semantic expression but is much slower with latencies above 1000ms [25]. This is typical for ontology-based approaches. The evaluation shows that our modular approach meets the latency requirements of the IoT while providing its required semantic expressiveness.

C. Low Programming Complexity

A continuous evaluation using qualitative questionnaires with more than 100 student testers confirms a very good usability. Quantitative task time measurements for the modeling work confirms this. In literature the values for implementing complex IoT tasks in particular [26] and for the work with ontologies [9], [22] are significantly higher.

The qualitative evaluation shows that our approach is well usable and overcomes the complexity of ontologies while providing comparable expressiveness.

IV. Related Work

Our work differs from classic discovery approaches such as UPnP [7] or DNS-based approaches [8]. Those are well suitable for connecting resource constraint devices but do not match the required semantic richness of the IoT.

Service-oriented mechanisms such as UDDI are not suitable for our requirements (Sec. I) either, as they are not extensible with predicates at runtime.

Our solution implements a modular ontology that can be extended at runtime. As we use a data-centric service coupling approach via the VSL middleware, discovering services equals discovering data (or devices) [2], [6]. Therefore, service discovery becomes querying the VSL ontology.

With its high semantic expressiveness, our solutions compares directly with ontologies such as OWL. Classic ontologies are *more complex to use*, and *slower to process* [9], [22], [23]. Our solution is simpler and faster as section III shows.

From its target, enabling service data discovery, and implementation as distributed directories, our work is closer to service discovery mechanisms. Different surveys give an overview regarding IoT service and data discovery [27], [28].

With discovery related state of the art we share the view that semantic service discovery today is a key challenge for the IoT. With some state of the art we also share the need for distributed implementation for offering higher scalability and performance.

None of the related works sees directories/ search providers as regular services that can be installed at runtime. Rather they are seen as special platform features. Our approach is more developer-friendly. It can implement more complex functionality than approaches using fixed service properties.

In [29] the authors describe a semantic discovery based on an ontology. In contrast to us, their approach is centralized. Using the Web Ontology Language (OWL) makes their approach more complex for developers and less extensible.

The authors of [24] propose a centralized repository for service configurations. Such configurations can be searched and return the URI of services matching the search criteria. The proposal stays very vague. We want to avoid a single repository for performance and scalability reasons.

The works [30], [31] propose using a DHT for storing metadata to services. Different to our approach, using a DHT requires implementing the query logic in all peers. Our system is more flexible as it can be updated with new search predicates by simply plugging in search providers at any peer.

The COnstrained Application Protocol (COAP) [18] implements directory functionality distributed on each node. Limited requests can be sent to a node to discover its semantic offerings. Like the DHT approaches, this requires implementing the functionality in each node.

[32] present an approach for federating multiple Webservice Universal Description, Discovery, and Integration (UDDI) directories. Their purpose is to increase the results and not to enrich the search semantics. In contrast to our solution the search semantics are fixed.

Several works provide solutions for fuzzy discovery to services best matching search criteria such as Quality of Service (QoS) parameters [33], [34].

The approach presented in [35] offers search in the Web of Things (WoT). Their "search semantics" could be used in our meta search providers to enable more powerful search federations than first order logic. They also offer extensibility in that sense that future predicates can be added to the discovery logic. Their approach is different from ours as the resource descriptions with the WoT are distributed and attached to the resources. Our mapping between search terms, predicates, and service URIs happens locally within the search providers, enabling a faster search. Federating the searches locally, our approach is also has a clear termination.

Different works propose introducing directories. The closest is [36]. The authors generally describe the functionality of semantic catalogs that enable searching for data items with unique URIs. Like us, they propose a modular a modular design and search combinations with intersections and unions. However, the use of the Resource Description Framework (RDF) makes their implementation more complex than ours. Our prototype should be compliant with their high-level specification, supporting our design decisions.

V. Conclusion

Service discovery is a key challenge of today's Internet of Things (IoT). We presented a distributed modular approach for discovering services using various predicates and search terms (objects) (section II). By focusing on semantically rich data discovery our approach complements and extends existing research and standardization in service discovery.

Our work implements a semantically-rich, dynamically extensible ontology for IoT systems. The introduced modularization adds little latency, suiting time critical operation (section III).

Continuing our crowdsourced data modeling work [5], we consider our approach better understandable and scalable than monolithic concepts such as the Web Ontology Language (OWL) while providing a comparable level of expressiveness. Ontological-wise, our search providers extend the minimalistic built-in VSL ontology with new predicates and objects.

In contrast to the typical IoT service discovery solutions [7], [8] our approach is dynamically extensible at run time. It can implement more complex functionality than existing approaches that typically use fixed semantics for describing discoverable service attributes. With the implemented modularity we are more developer friendly and faster than solutions with expressiveness comparable to a full-blown ontology.

Another major advantage of our approach is that it enables adding attributes to services a posteriori at runtime. Typically used discovery concepts requires defining the discovery parameters at development time or latest at deployment time. Our approach fits the requirement of the dynamically changing IoT and its applications better.

We see our work as significant contribution towards overcoming the inherent complexity of today's IoT.

ACKNOWLEDGMENT

This research has been supported by the German Federal Ministry of Economic Affairs and Energy (BMWI) project DECENT (0350024A) and the German-French Academy for the Industry of the Future project SCHEIF.

REFERENCES

[1] M.-O. Pahl and G. Carle, "The Missing Layer - Virtualizing Smart Spaces," in *10th IEEE International Workshop on Managing Ubiquitous Communications and Services 2013 (MUCS 2013, PerCom 2013 adjunct)*, San Diego, USA, 2013, pp. 139–144.

[2] M.-O. Pahl, G. Carle, and G. Klinker, "Distributed Smart Space Orchestration," in *Network Operations and Management Symposium 2016 (NOMS 2016) - Dissertation Digest*, 2016.

[3] M.-O. Pahl, "Data-Centric Service-Oriented Management of Things," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, Ottawa, Canada, May 2015, pp. 484–490.

[4] D. Lu, D. Huang, A. Walenstein, and D. Medhi, "A Secure Microservice Framework for IoT," in *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE*. IEEE, 2017, pp. 9–18.

[5] M.-O. Pahl and G. Carle, "Crowdsourced Context-Modeling as Key to Future Smart Spaces," in *Network Operations and Management Symposium 2014 (NOMS 2014)*, May 2014, pp. 1–8.

[6] M.-O. Pahl and S. Liebald, "Designing a Data-Centric internet of things," in *2019 International Conference on Networked Systems (NetSys) (NetSys'19)*, Garching b. München, Germany, Mar. 2019.

[7] M. Boucadair, R. Penno, and D. Wing, "Universal Plug and Play (UPnP) Internet Gateway Device - Port Control Protocol Interworking Function (IGD-PCP IWF)," RFC 6970, Jul. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc6970.txt

[8] S. Cheshire and M. Krochmal, "DNS-Based Service Discovery," RFC 6763, Feb. 2013. [Online]. Available: https://rfc-editor.org/rfc/rfc6763.txt

[9] U. Aßmann, S. Zschaler, and G. Wagner, "Ontologies, Meta-models, and the Model-Driven Paradigm," in *Ontologies for Software Engineering and Technology*, C. Calero, F. Ruiz, and M. Piattini, Eds. Berlin Heidelberg: Springer, 2006, pp. 249–273.

[10] M.-O. Pahl and G. Carle, "Taking Smart Space Users into the Development Loop," in *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*. New York, NY, USA: ACM, 2013, pp. 793–800.

[11] R. Want, "When Cell Phones Become Computers," *Pervasive Computing, IEEE*, vol. 8, no. 2, pp. 2–5, 2009.

[12] M.-O. Pahl and L. Donini, "Giving iot edge services an identity and changeable attributes," in *International Symposium on Integrated Network Management (IM)*, Washington DC, USA, Apr. 2019.

[13] ——, "Securing IoT Microservices with Certificates," in *Network Operations and Management Symposium (NOMS)*, Apr. 2018.

[14] M.-O. Pahl and F.-X. Aubet, "All eyes on you: Distributed Multi-Dimensional IoT microservice anomaly detection," in *2018 14th International Conference on Network and Service Management (CNSM) (CNSM 2018)*, Rome, Italy, Nov. 2018.

[15] A. Pras and J. Schoenwaelder, "On the Difference between Information Models and Data Models," RFC 3444 (Informational), Internet Engineering Task Force, Jan. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3444.txt

[16] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "Simple Network Management Protocol (SNMP)," RFC 1157 (Historic), Internet Engineering Task Force, May 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1157.txt

[17] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," RFC 6241 (Proposed Standard), Internet Engineering Task Force, Jun. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6241.txt

[18] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, Jun. 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7252.txt

[19] M. Weiser, "The Computer for the 21st Century," *Scientific American*, vol. 265, no. 3, pp. 94–104, Sep. 1991.

[20] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970. [Online]. Available: http://doi.acm.org/10.1145/362384.362685

[21] R. Fielding, "Principled design of the modern Web architecture," *ACM Transactions on Internet Technology*, vol. 2, 2002.

[22] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen, "From SHIQ and RDF to OWL: the making of a Web Ontology Language," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 1, pp. 7–26, Dec. 2003.

[23] M.-O. Pahl, "Distributed Smart Space Orchestration," Ph.D. dissertation, Technische Universität München, München, Jun. 2014.

[24] S. K. Datta and C. Bonnet, "Search engine based resource discovery framework for Internet of Things." *GCCE*, 2015.

[25] N. Srinivasan, M. Paolucci, and K. Sycara, "Semantic web service discovery in the owl-s ide," in *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, vol. 6, Jan 2006, pp. 109b–109b.

[26] R. Grimm, J. Davis, B. Hendrickson, E. Lemar, A. Macbeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, and D. Wetherall, "Systems Directions for Pervasive Computing," in *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, May 2001, pp. 147–151.

[27] A. Bröring, S. K. Datta, and C. Bonnet, "A Categorization of Discovery Technologies for the Internet of Things," *6th International Conference on the Internet of Things*, 2016.

[28] S. K. Datta, R. P. F. Da Costa, and C. Bonnet, "Resource discovery in Internet of Things - Current trends and future standardization aspects." *WF-IoT*, 2015.

[29] C. H. Y. C. H. Yun, Y. W. L. Y. W. Lee, and H. S. J. H. S. Jung, "An evaluation of semantic service discovery of a U-city middleware," *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on*, vol. 1, pp. 3–6, 2010.

[30] S. Cirani, L. Davoli, G. Ferrari, R. Léone, P. Medagliani, M. Picone, and L. Veltri, "A Scalable and Self-Configuring Architecture for Service Discovery in the Internet of Things." *IEEE Internet of Things Journal*, 2014.

[31] S. Cirani and L. Veltri, "Implementation of a framework for a DHT-based distributed location service," *SoftCom 2008: 16th International Conference on Software, Telecommuncations and Computer Networks*, pp. 279–283, 2008.

[32] P. Rompothong and T. Senivongse, "A query federation of UDDI registries." *ISICT*, 2003.

[33] S. B. Mokhtar, D. Preuveneers, N. Georgantas, S. B. Mokhtar, D. Preuveneers, N. Georgantas, and Y. Berbers, "EASY : Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support Yolande Berbers To cite this version : EASY : E fficient Sem A ntic S ervice Discover Y in Pervasive Computing Environments with QoS and Context S," 2009.

[34] L.-H. Vu, M. Hauswirth, and K. Aberer, "Towards p2p-based semantic web service discovery with qos support," in *Proceedings of Workshop on Business Processes and Services BPS*, vol. 3812, no. 507483, 2006, pp. 18–31. [Online]. Available: http://www.springerlink.com/index/q211622119212541.pdf

[35] S. Mayer and D. Guinard, "An extensible discovery service for smart things." *WoT*, 2011.

[36] BSI PAS 212, "Automatic resource discovery for the Internet of Things – Specification Publishing and copyright information," no. 1, 2016. [Online]. Available: https://shop.bsigroup.com/upload/276605/PAS212-corr.pdf