Smart Components for Enabling Intelligent Web of Things Applications

Felix Leif Keppmann, Maria Maleshkova
AIFB, Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: felix.leif.keppmann@kit.edu, maria.maleshkova@kit.edu

***Abstract*—We are currently witnessing an increased use of sensor technologies, abundant availability of mobile devices, and growing popularity of wearables, which enable the direct integration of their data as part of rich client applications in a multitude of different domains. In this context, the Internet of Things (IoT) promises the capability of connecting billions of devices, resources, and things together in an integrated way. However, what we are currently witnessing is the proliferation of isolated islands of custom IoT solutions. A first step towards enabling some interoperability in the IoT is to connect things to the Web and to use the Web stack, thereby conceiving the so-called Web of Things (WoT). However, even when a homogeneous access is reached through Web protocols, a common understanding is still missing, specifically in terms of heterogeneous devices, different programmable interfaces and diverse data formats and structures. Our work focuses on two main aspects: overcoming device and interface heterogeneity as well as enabling adaptable and scalable (i.e., intelligent) decentralised WoT applications. To this end, we present an approach for realising decentralised WoT solutions based on three main building blocks: 1) smart components as an abstraction of a unified approach towards realising the devices' interfaces, communication mechanisms, semantics of the devices' resources and capabilities, and decision logic; 2) adaptability of devices' interfaces and interaction at runtime; 3) adaptability of the devices' data structures and semantics at runtime. We show how our approach can be applied by introducing a reference smart component design, provide a thorough evaluation in terms of a proof-of-concept implementation of an example use case.**

Keywords–Smart Components, decentralised applications, Web of Things, REST, Linked Data

I. INTRODUCTION

Current developments in many domains are characterised by the increased use of mobile devices, wearables, and sensors, which bring the promise of higher digitalisation and rich client applications. In this context, the vision of the Internet of Things (IoT) aims to achieve the capability of connecting billions of devices, resources, and things together in the Internet. Still, what we are currently witnessing is the proliferation of isolated islands of custom IoT solutions, which support a restricted set of protocols and devices and cannot be easily integrated or extended. A first step towards enabling some interoperability in the IoT is to connect things to the Web and to use the Web stack, thereby conceiving the so-called Web of Things (WoT). However, even when a homogeneous access is reached through Web protocols, a common understanding is still missing, specifically in terms of heterogeneous devices, different programmable interfaces, and diverse data. Semantic technologies can be used to describe dataflows on a meta level, capturing the meaning of devices' inputs and outputs, and thus abstracting away from the syntactic structure. However, having the semantics of the data is not enough. While we can describe the exchanged data, the resulting solutions are limited to a specific domain, and the heterogeneous device integration is still lacking.

In this context, our work focuses on two main challenges: 1) overcoming heterogeneity, not only in terms of data but also in terms of devices and interfaces, and 2) enabling intelligent WoT applications. In terms of handling the plenitude of existing devices, we advocate an approach based on providing a unified view on devices and describing them in terms of their programmable interfaces, since this is how their integration as part of applications is realised. The difficulty that we face here is that in multi-stakeholder scenarios, where devices are built by several manufacturers and integrated and used by other parties, it is hardly possible to know all requirements of every possible integration scenario at design time. As a result, we can only provide default interfaces and interaction, thus needing to be able to adapt the component to provide the optimal solution for a specific use case. To this end, we also focus on realising intelligent WoT applications, where the "intelligence" is in terms of being able to adapt to changing requirements, at deployment time, but more importantly at runtime.

In this context, we make the following contributions. First, we present an approach for realising decentralised WoT solutions based on three main building blocks: 1) smart components as an abstraction of a unified approach towards realising the devices' interfaces, communication mechanisms, semantics of the devices' resources and capabilities, and decision logic; 2) adaptability of devices' interfaces and interaction at runtime; 3) adaptability of the devices' data structures and semantics at runtime. Second, we show how our approach can be applied by introducing a reference smart component design, based on Web and Semantic Web paradigms and technologies. We back up the design by a specific implementation. Finally, we provide a thorough evaluation of a proof-of-concept implementation of an example use case.

The remainder of this paper is structured as follows. In Section II, we introduce our motivation scenario, describing the challenges that we are focusing on. Section III describes the requirements for building WoT systems and the preliminaries that we build upon. Furthermore, it provides an architecture to realise this approach, and describes our implementation. For evaluation, in Section IV, we demonstrate the adaptability of our system to update at runtime the devices' interfaces and the controlling logic. We describe related work in Section V and conclude in Section VI.

II. MOTIVATION

In the following, we introduce a scenario that puts our work into context and use it to introduce the specific challenges that we are focusing on.

A. Scenario

To motivate our approach, we choose a generic body tracking component as an example, i.e., "thing". This component is able to track people in front of its video sensor

 ISBN: 978-1-61208-518-0

by interpreting the video through algorithmic analysis of the captured video images. The body tracking is provided in the form of coordinates of the different joint points of the skeleton of the tracked people. These coordinates are provided to or send to other components, depending on the specific use case, with the sensor (or a custom point) as the origin of the coordinates.

As typical parts of the body tracking components, we consider a depth video camera, an analysis middleware, and an access layer with network connectivity. The depth video camera provides depth images that are enriched with additional information about the distance from the camera for every pixel. The analysis middleware encapsulates various algorithms, which are required to interpret the stream of depth images. By comparing and classifying the content in a sequence of incoming images, these algorithms calculate and extract different information, e.g., the body tracking data in our case. The access layer provides access to the body tracking data, via an interface to other components, or interacts with the interfaces of other components. Furthermore, the access layer may also enable retrieving data, e.g., modification of configuration settings in our case. An operating system and further common software infrastructure augments these parts, which may be distributed across different hardware devices or embedded in one physical system. In both cases, the body tracking appears as one distinct component to the rest of the network via its access layer.

We explicitly abstract from a particular use case and instead design the "thing" as a generic body tracking component. Thereby, we keep the focus on the specific functionality, i.e., body tracking, which is encapsulated and may be combined with other "things" in a larger integration scenario. This integration scenario may be, for example, safety monitoring in a factory, gesture interaction with technical artefacts, or responsive art installations. By combining and integrating the component with other components, we build distributed applications, which exceed the sum of their parts in functionality.

Our body tracking component is just one example out of a heterogeneous landscape of "things". A multitude of functionalities ranges from simple temperature sensors to complex robots. Different hardware and software requirements range from low-energy embedded systems to processing-intensive calculations, e.g., body tracking. In this market, several stakeholders exist, e.g., different manufacturers of "things", technology integrators, or customers with specific integration scenarios. In this context, there are several challenges that we face while realising the integration of components into a coherent application with a value-added functionality, which is, by design, of distributed nature.

B. Problem Focus

In the following, we focus on two main challenges: 1) the information asymmetry between the design of a component and its use at runtime in different integration scenarios, and 2) the inefficiency that can occur when developing a generic component, which may have several specific use cases.

1) Requirements Asymmetry: In multi-stakeholder scenarios, where components are built by several manufacturers and are integrated and used by others, we hardly know all requirements of every possible integration scenario at design time. As a result, we can only provide default interfaces and interaction but are not able to adapt the component to provide the optimal solution for a specific use case.

2) Development Inefficiency: Even if all integration scenarios would be known, we face an inefficiency issue. Designing and developing the same component in several adapted versions for each and every use case does not only lead to a very complex and inefficient, i.e., time-consuming, development but in consequence may also be inefficient in terms of business requirements, i.e., be unprofitable.

III. Smart Component

In the following, we introduce our approach by clarifying the requirements, presenting the preliminaries, and elaborating on our architecture and implementation.

A. Requirements

As part of the IoT vision, we see applications built upon a number of different components that communicate data to provide a value-added functionality without the necessity of centralised control within the application. While central control is still a valid – thus still to be supported – integration pattern, we must acknowledge integration scenarios, in which distributed control is required, e.g., caused by the scenario itself, or by performance, redundancy, or latency requirements. The requirements for a component's architecture are, with respect to the previously mentioned problems, three-fold:

1) Adaptability of Interfaces and Interaction at Runtime: First, for communication and thus the ability to establish data flows between components, which are required to provide the value-added functionality of an application, components need to interact. This interaction can be supported by a component 1) by – passively – providing an interface for other components, or 2) by – actively – interacting with interfaces of other components. Components must be able to adapt their interfaces and interaction according to the specific situation in the integration scenarios. We derive this requirement from the inability to foresee or consider all possible integration scenarios during the design time of a component.

2) Adaptability of Data Structures and Semantics at Runtime: Second, complementary to the interaction between components, the data, which is communicated, must be handled and processed in an appropriate manner according to both the data structure and semantics. Components must be able to adapt the structure and align the semantic annotation of data to the specific situation in the integration scenarios. We derive this requirement again from the inability to foresee all possible integration scenarios during the design time of a component.

3) Adaptability of Controlling Logic at Runtime: Third, a distributed application, which is composed of several different independently developed components, must be controlled in some way, i.e., a controlling intelligence within the application must exist that coordinates the collaboration of components to achieve the value-added functionality of the application. By default, a central controlling component, custom for the specific integration scenario, actively controls all other components. However, to support scenarios with distributed control, as facilitated by the IoT vision, components must be adaptable in terms of their intelligence by being able to update the controlling logic at runtime.

B. Preliminaries

We build our contribution upon a number of well established paradigms for enabling large heterogeneous distributed

 ISBN: 978-1-61208-518-0

systems: Representational State Transfer (REST) for overcoming heterogeneity at interface level and Linked Data (LD) to ease the semantic integration of data.

While paradigm-wise being technology-agnostic, REST [1] is usually realised by utilising the Hypertext Transfer Protocol (HTTP). It incorporates the concept of Uniform Resource Identifier (URI) as unique identifier for resources and provides transport mechanisms for data transfer. HTTP – used as true application protocol – defines a constrained set of methods, e.g., GET, PUT, POST, and DELETE as the most known methods, with standardised semantics, i.e., the protocol defines how clients must interact with resources identified by URIs. Acknowledging the heterogeneous inconsistent nature of large distributed systems with multiple stakeholders, status codes for handling various types of successful and failing communication are part of the protocol.

The architectural paradigm Linked Data introduces shared semantics to data and builds – similar to REST – on URIs as unique identifiers. Technological buildings blocks of Linked Data, that we are taking advantage of, are the Resource Description Framework (RDF) [2], the SPARQL Protocol and RDF Query Language (SPARQL) [3], and the Notation3 (N3) [4] syntax for rule and assertion logic for RDF.

We introduce the notion of a "Smart Component (SC)", when this component is built following our architectural approach: 1) REST for realising interfaces and the communication between components; 2) Linked Data for describing the exchanged data, interface resources, and components' capabilities; and 3) decentralised smartness of each component, described in terms of rules.

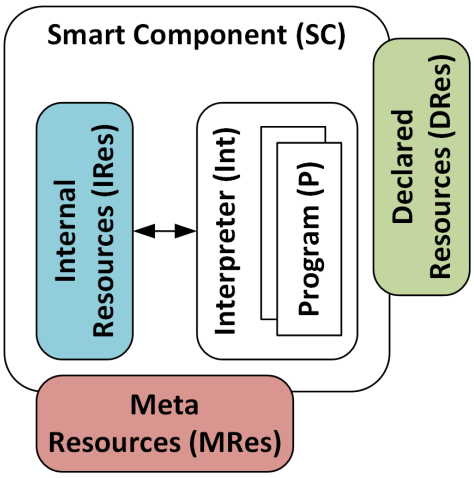

Figure 1. Smart Component Architecture

C. Architecture

Our approach tackles the requirements by combining and extending Web and Semantic Web technologies. In Figure 1, we present the internal architecture of a component that follows our Smart Component approach. It comprises a number of resources providing semantically annotated data, rule programs, which can be interpreted with respect to the data, and an interpreter for interpreting the rule programs.

1) Internal Resources (IRes): Internal resources provide access to the core functionality and data of the component, that distinguishes it from other components. In our motivation scenario, the core comprises depth video recording, image analysis, body tracking, and configuration. Only relevant parts of the core functionality and data are exposed as internal resources, e.g., the complete body tracking data as well as selected configuration parameters. Common to all internal resources is the RDF-conform modelling of data and its integration with the interpreter. The internal resources together form an internal RDF knowledge graph. We do not explicitly prescribe how these resources are integrated with the interpreter to not overly restrict the development of components. Integration can range from programmatic integration, to file-based access, and to HTTP or other communication protocols.

2) Declared Resources (DRes): Declared resources form the Application Programming Interface (API) of the component exposed to the network at runtime. In our motivation scenario, we could, for example, expose the skeleton information of each tracked person as declared resources, or only the distance of specific joint points. These resources conform to the Linked Data and REST paradigms; thus they are identified by URIs, accessible via HTTP, and provide data in RDF serialisation formats. Declared resource are defined as SPARQL CONSTRUCT patterns, which are evaluated against the internal RDF knowledge graph.

3) Program (P): While construct queries are evaluated against the internal RDF knowledge graph, we enable its modification through programs. Programs are written in a declarative N3-based rules language, interpreted by the interpreter, and encode transformation between ontologies, enrichment by reasoning, decisions, and including of data from other components with built-in interaction functions. Optionally, the rule language may provide further built-in functions, e.g., for calculations, to ease the declaration of programs.

4) Interpreter (Int): We introduce the interpreter as a central element of our approach. On the one hand, the interpreter maintains the internal RDF knowledge graph that is build up during each interpreter run by 1) adding data from internal resources, 2) adding data of external resources of other components, if requested by interaction rules in programs, and 3) adding data, which is derived by deduction rules in programs. On the other hand, the interpreter 1) evaluates construct queries of declared resources against the internal RDF knowledge graph and 2) modifies, if requested by interaction rules in programs, external resources. In both cases, external resources are Linked Data REST resources, which belong to other components and are accessible via HTTP to the network. In summary, the task of the interpreter is to negotiate between the private API, the public API, and the interaction with resources of other components.

5) Meta Resources (MRes): With meta resources, we introduce the last type of resources for our design of a Smart Component. These resources are provided by a component as part of the public API, i.e., are Linked Data REST resources accessible by HTTP. In contrast to declared resources, which expose internal data and functionality of the component, meta resources expose the state of the interpreter and declared resources. In other words, they allow to create, update, modify, and delete, rule programs and graph patterns of declared resources. With meta resources, we enable the adaptation of components' behaviour at runtime.

 ISBN: 978-1-61208-518-0

D. Implementation

We implemented a Smart Component based on our motivating scenario to show the feasibly of our architecture and to support our evaluation. Natural Interaction via REST (NIREST) [5] integrates hardware support for depth video cameras, tracking middleware with body tracking algorithms, and, as intelligent access layer, a rule-based data integration framework for Linked Data REST resources. We abstract in the following from actual device manufacturers and software providers, which may change over time.

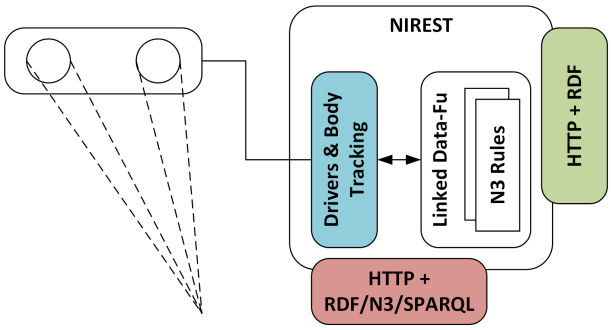

Figure 2. Scenario Component Implementation

In Figure 2, we provide an overview of the implementation that realises all parts of our architecture. The hardware part of our component consists of a separate depth video camera and a computer with appropriate processing power for video analysis. The camera provides raw depth image data and is connected by wire with the computer. Both hardware pieces may be merged in one embedded device. The software part of our component consists of three layers, that we directly integrated in program code. For the two lowest levels, we include third party frameworks, as this functionality is not in the focus of our work. We utilise a low-level framework for device connectivity on the lowest level that provides raw images to a tracking middleware at the second level. The body tracking data, provided by the second level, is then included in the intelligent access layer, which we describe in the following.

To realise the interpreter, programs, and declared resources, we utilize Linked Data-Fu (LD-Fu) [6][7][8]. LD-Fu comprises the LD-Fu rule language based on N3 syntax, the LD-Fu interpreter for execution, and follows a generic approach for the integration of Linked Data REST resources. The rule language supports the declaration of: 1) deductions rules for inferencing new knowledge, 2) interaction rules for encoding of HTTP interaction with built-in functions, and 3) built-in functions to ease decisions and transformations with mathematical calculations. The interpreter maintains an internal RDF graph and is capable of evaluating deduction rules or executing HTTP requests. Both the results of deduction rules and the payload of answers to requests are added to the internal graph and may be subject to further rules, until reaching a fixpoint.

We extended the LD-Fu implementation to support our Smart Component approach by introducing a REST API and enable time-based continuous evaluation of programs. The LD-Fu REST API is closely integrated with the LD-Fu interpreter and supports the creation of interpreter instances, creation and modification of rule programs per instance, as well as the creation and modification of declarative resources per instance. With separate interpreter instances, we support the participation of a component in more than one distributed application, i.e., we may adapt the behaviour of the component per distributed application with a distinct set of interpreter, programs, and declared resources. In addition, we introduced time-base continuous execution of the interpreter.

While LD-Fu supports several ways to include resources, e.g., file-based, pipe-based, or through HTTP requests, we utilise the libraries for direct code-based integration. Therefore, we enable the interpreter to read the body tracking data, annotated in RDF, directly from the tracking middleware and add it during each run to the internally maintained RDF graph. Subsequent, programs declared in the N3-based LD-Fu rule language and declared resources defined as SPARQL CONSTRUCT queries are evaluated against this internal RDF graph.

IV. Evaluation

We provide an implementation of our approach and a thorough evaluation in terms of: 1) evaluating the deployment and adaptability of decentralised logic within smart components, and 2) evaluating the integration and adaptability of interfaces and interaction.

```
<nirest://user/0>
    nirest:skeleton [
        nirest:jointPoint [
            nirest:coordinate [
                nirest:x "459.8463"^^xsd:float ;
                nirest:y "404.0497"^^xsd:float ;
                nirest:z "2037.2391"^^xsd:float ;
                a nirest:Coordinate ] ;
            a nirest:RightHandJointPoint ] ;
        ...
```

Figure 3. Internal Resources

In Figure 3, we provide a snippet of the RDF graph of internal resources, which is included during every interpreter run of our scenario component. We use the Turtle serialization format and omit, due to space constraints, prefixes in this and following figures. For each person in front of the sensor, the implementation of the component provides – once a person is tracked – an URI as well as a description of the skeleton' joint points, including coordinates. In the figure, we show the description of a right hand joint point, as one of the joint-points described by each skeleton. The unit of measurement for coordinates is millimetres and the descriptions are internally updated with a frequency of approximately 30hz [9] by the sensor.

Prior to the adaptation given in the following, the component has been developed, deployed, and started. With respect to the architecture (Figure 1) and implementation (Figure 2), the component is already tracking bodies in front of the sensor, providing internally access to this tracking data to interpreter instances, and is exposing the generic meta interface for adaptation at the network.

A. Deployment and Adaptability of Decentralised Logic

To integrate the component as part of an application, we need to instantiate and configure the interpreter, i.e., initiate an instance of LD-Fu. In Figure 4, we provide the command used to create an interpreter instance by interacting with the meta interface and to deploy a specific configuration (100ms between interpreter runs), which is shown in the lower part.

 ISBN: 978-1-61208-518-0

```
$ curl -X "PUT" -H "Content-Type: text/turtle" \
    http://localhost:8888/scenario \
    --data-binary @config-scenario.ttl
---
<> ldfu:delay 100 ; a ldfu:Configuration .
```

Figure 4. Instance Configuration

```
$ curl -X "PUT" -H "Content-Type: text/n3" \
    http://localhost:8888/scenario/p/program \
    --data-binary @program-alarm.n3
---
{   ?point nirest:coordinate ?coordinate .
        ?coordinate nirest:x ?x ; nirest:y ?y ; nirest:z ?z .
    (?x "2") math:exponentiation ?x_ex .
    (?y "2") math:exponentiation ?y_ex .
    (?z "2") math:exponentiation ?z_ex .
    (?x_ex ?y_ex ?z_ex) math:sum ?sum .
    ?sum math:sqrt ?square_root .
    ?square_root math:lessThan "1000.0" . } =>
{ ?point scenario:alarm "true" . } .
```

Figure 5. Program Deployment

As already described, programs are interpreted by the interpreter and enrich the RDF of internal resources with inferred knowledge, e.g., triggering of alarms when specific distances become too short. Due to space constrains for figures, we simplify our example to a pure distance-based alarm. As soon as a part of a person's body intrudes the space within one meter of the tracking sensor, an alarm in triggered. In Figure 5, we provide a program containing a single N3 rule, which calculates the euclidean distance to the sensor for each point provided by internal resources. For the calculation, coordinates are matched in the body of this rule, patterns adhering to a built-in ontology, i.e., the "math" prefix, are interpreted, mathematically evaluated, and the calculation results are bound to respective variables. As a consequence, if the condition "distance less than 1000mm" is fulfilled, we enrich the RDF sub-graph of the point with a custom "alarm" property by deriving a respective triple in the rule head.

By deploying this rule, we adapt the component's data structure and semantics at runtime (second requirement; Section III-A2), by triggering an alarm based on the given coordinates, and at the same time, adapt the controlling logic at runtime (third requirement; Section III-A3), by including a distance condition.

```
$ curl -X "PUT" -H "Content-Type: application/sparql-query" \
    http://localhost:8888/scenario/r/shutdown \
    --data-binary @resource-shutdown.rq
---
CONSTRUCT { ?point scenario:shutdown "true" . }
WHERE { ?point scenario:alarm "true" . }
```

Figure 6. Declared Resource

B. Integration and Adaptability of Interfaces and Interaction

We show the integration of the component with other components of a distributed application. As stated before, we can establish this integration either by 1) providing resources for interaction with other components or by 2) actively interacting with resources of other components. In Figure 6, we provide an example for the first case. A SPARQL CONSTRUCT query is deployed as a declared resource at the instance of our scenario's interpreter, by interacting with the meta interface. It constructs a simple "shutdown" triple if an "alarm" triple from the preceding program evaluation is found. The query is evaluated during every interpreter run and the result is accessible as RDF via HTTP by requesting the media type of supported RDF serialization formats.

```
$ curl -X "PUT" -H "Content-Type: text/n3" \
    http://localhost:8888/scenario/p/shutdown \
    --data-binary @program-shutdown.ttl
---
{ ?point scenario:alarm "true" . } =>
{   [] http:mthd http-m:PUT;
       http:requestURI <http://localhost:8889>;
       http:body { <> scenario:shutdown "true" . } . } .
```

Figure 7. Interaction Program

For the second case (actively interacting with resources of other components), we provide and example in Figure 7. Again by interaction with the meta interface, we deploy a second program at the scenario interpreter instance. It contains a single N3 rule that matches to the body of "alarm" triples generated by the first program. In the head of the rule, we use the interaction capabilities of LD-Fu, encoded with respective ontologies, i.e., "http" and "http-m" prefixes. If the condition is fulfilled, i.e., an "alarm" triple was generated before, the interpreter executes a HTTP PUT request at the specified URI, containing our custom "shutdown" triple as payload.

By deploying the declared resource and the rule, we adapt the component's interface and interaction (first requirement; Section III-A1), by passively exposing alarms through a resource to other components at the network and by actively communicating alarms to other components. At the same time, we adapt again the controlling logic at runtime (third requirement; Section III-A3), by including the alarm triple as a condition for the interaction.

V. Related Work

We focus in our related work on three areas: 1) read-write Linked Data (LD), 2) the Web of Things (WoT), and 3) the Semantic Web of Things (SWoT).

Read-write Linked Data is built upon the idea of combining the architectural paradigms of Linked Data (LD) [10] and Representational State Transfer (REST) [1]. This combination has been used in several approaches, e.g., Linked Data Fragments (LDF) [11], Linked APIs (LAPIS) [12], Linked Data Services (LIDS) [13], RESTdesc [14], or Linked Open Services (LOS) [15]. The Linked Data Platform (LDP) [16] standardizes this combination, including RDF and non-RDF resources, containers, and rules for HTTP-based interaction with resources. Our approach aims at the adaptation of components to specific application scenarios, while still being compatible with arbitrary Linked Data REST interfaces.

The IoT [17] paradigm is about connecting every device, application, object, i.e., thing, to the network, in particular the Internet and thus to ensure connectivity. The Web of Things (WoT) [18] builds on top of this paradigm to provide integration not only on the network layer but also on the application layer, i.e., the Web. The goal is to make things part of the Web by providing their capabilities as REST services.

 ISBN: 978-1-61208-518-0

Therefore, common existing Web technologies are introduced, e.g., URIs for identification and HTTP as application protocol for transport and interaction. Integrating these technologies has been, for example, addressed for embedded devices in [19].

The extension of IoT to WoT is primarily focused on the interoperability between things on the application layer. In order to foster horizontal integration and interoperability the Semantic Web of Things (SWoT) [20] focuses a common understanding of multiple capabilities and resources towards a larger ecosystem by introducing Semantic Web technologies to the IoT. Challenges related to SWoT have been, for example, addressed by the SPITFIRE [21] project, or the Micro-Ontology Context-Aware Protocol (MOCAP) [22], both in the area of sensors. We build upon several synergies introduced by a common resource-oriented viewpoint of the Linked Data and REST paradigms. These paradigms also play a key role in WoT and in particular SWoT to cope with heterogeneous data models and interaction mechanisms. However, integrating decentralised components into applications without central control, even with a clear interaction model and semantically powerful data model, requires to distribute the controlling intelligence, at least to some extent, to the components. In this context, our approach aims to enable the adaptation of components to specific application scenarios at runtime, while still being compatible with other approaches based on read-write Linked Data REST interfaces.

VI. Conclusion

The growing use and popularity of mobile devices, wearables and sensors offers new opportunities for the way that products and services are being designed, developed and offered. The IoT and WoT lay the foundation for integrating devices by providing network connectivity and a stack of communication protocols, while SWoT aims to enhance these to address the lack of interoperability. In this context, our work focuses on two main aspects: overcoming not only data but also device and interface heterogeneity as well as enabling adaptable (i.e., intelligent) decentralised WoT applications. To this end, we introduce Smart Components as a unified approach towards realising the devices' interfaces, communication mechanisms, semantics of the devices' resources and capabilities, and controlling logic. We provide support for the adaptability of devices' interfaces and interaction, as well as of devices' data structures and semantics, at runtime. We believe that enabling interoperability but also offering simple mechanisms for adaptability are key for contributing towards the evolution of the Web.

References

[1] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, USA, 2000.

[2] R. Cyganiak, D. Wood, and M. Lanthaler, "RDF 1.1 Concepts and Abstract Syntax," W3C, Recommendation, 2014, http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/. Latest version available at http://www.w3.org/TR/rdf11-concepts/ [retrieved: 10, 2016].

[3] C. B. Aranda, O. Corby, S. Das, L. Feigenbaum, P. Gearon, B. Glimm, S. Harris, S. Hawke, I. Herman, N. Humfrey, N. Michaelis, C. Ogbuji, M. Perry, A. Passant, A. Polleres, E. Prud'hommeaux, A. Seaborne, and G. T. Williams, "SPARQL 1.1 Overview," W3C, Recommendation, 2013, http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/. Latest version available at http://www.w3.org/TR/sparql11-overview/ [retrieved: 10, 2016].

[4] T. Berners-Lee and D. Connolly, "Notation3 (N3): A readable RDF syntax," W3C, Team Submission, 2011, http://www.w3.org/TeamSubmission/2011/SUBM-n3-20110328/. Latest version available at https://www.w3.org/TeamSubmission/n3/ [retrieved: 10, 2016].

[5] "Natural Interaction via REST (NIREST)," http://github.com/fekepp/nirest/ [retrieved: 10, 2016].

[6] "Linked Data-Fu (LD-Fu)," http://linked-data-fu.github.io/ [retrieved: 10, 2016].

[7] S. Stadtmüller, S. Speiser, A. Harth, and R. Studer, "Data-Fu: A Language and an Interpreter for Interaction with Read/Write Linked Data," in *International World Wide Web Conference*, 2013, pp. 1225–1236.

[8] S. Stadtmüller, "Dynamic Interaction and Manipulation of Web Resources," Ph.D. dissertation, Karlsruhe Institute of Technology, Karlsruhe, Germany, 2016.

[9] F. L. Keppmann and S. Stadtmüller, "Semantic RESTful APIs for Dynamic Data Sources," in *Workshop on Services and Applications over Linked APIs and Data at the European Semantic Web Conference*, 2014, pp. 26–33.

[10] C. Bizer, T. Heath, and T. Berners-Lee, "Linked Data - The Story So Far," *Semantic Web and Information Systems*, vol. 5, pp. 1–22, 2009.

[11] R. Verborgh, O. Hartig, B. De Meester, G. Haesendonck, L. De Vocht, M. Vander Sande, R. Cyganiak, P. Colpaert, E. Mannens, and R. Van de Walle, "Querying Datasets on the Web with High Availability," in *International Semantic Web Conference*, 2014, pp. 180–196.

[12] S. Stadtmüller, S. Speiser, and A. Harth, "Future Challenges for Linked APIs," in *Workshop on Services and Applications over Linked APIs and Data at the European Semantic Web Conference*, 2013, pp. 20–27.

[13] S. Speiser and A. Harth, "Integrating Linked Data and Services with Linked Data Services," in *Extended Semantic Web Conference*, 2011, pp. 170–184.

[14] R. Verborgh, T. Steiner, D. van Deursen, R. van de Walle, and J. Gabarró Vallès, "Efficient Runtime Service Discovery and Consumption with Hyperlinked RESTdesc," in *International Conference on Next Generation Web Services Practices*, 2011, pp. 373–379.

[15] R. Krummenacher, B. Norton, and A. Marte, "Towards Linked Open Services and Processes," in *Future Internet Symposium*, 2010, pp. 68–77.

[16] S. Speicher, J. Arwe, and A. Malhotra, "Linked Data Platform 1.0," W3C, Recommendation, 2015, http://www.w3.org/TR/2015/REC-ldp-20150226/. Latest version available at http://www.w3.org/TR/ldp/ [retrieved: 10, 2016].

[17] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, pp. 2787–2805, 2010.

[18] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, "From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices," in *Architecting the Internet of Things*. Springer, 2011, pp. 97–129.

[19] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle, "The Web of Things: interconnecting devices with high usability and performance," in *International Conference on Embedded Software and Systems*, 2009, pp. 323–330.

[20] A. J. Jara, A. C. Olivieri, Y. Bocchi, M. Jung, W. Kastner, and A. F. Skarmeta, "Semantic Web of Things: an analysis of the application semantics for the IoT moving towards the IoT convergence," *Web and Grid Services*, vol. 10, no. 2-3, pp. 244–272, 2014.

[21] D. Pfisterer, K. Romer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, A. Kröller, M. Pagel, M. Hauswirth, M. Karnstedt, M. Leggieri, A. Passant, and R. Richardson, "SPITFIRE: Toward a Semantic Web of Things," *Communications Magazine*, vol. 49, no. 11, pp. 40–48, 2011.

[22] K. Sahlmann and T. Schwotzer, "MOCAP: Towards the Semantic Web of Things," in *Posters and Demos at the International Conference on Semantic Systems*, 2015, pp. 59–62.

 ISBN: 978-1-61208-518-0