Internet of Things Middleware: How Suitable are Service-oriented Architecture and Resource-oriented Architecture

Janick Kubela, Matthias Pohl, Sascha Bosse and Klaus Turowski
Magdeburg Research and Competence Cluster, Faculty of Computer Science, University of Magdeburg, Magdeburg, Germany

Keywords: Resource-oriented Architecture (RoA), Service-oriented Architecture (SoA), Internet of Things, Middleware, Web of Things.

Abstract: Over the last years, the Internet of Things was researched widely. Thus, various IoT applications are developed based upon different use-cases. Numerous middleware solutions for the IoT are based on the Service oriented Architecture and Resource oriented Architecture. Both approaches do support the connection of distributed objects but no research is done to check the suitability of SoA and RoA in the context of common IoT requirements in an adequate scope. In the context of this paper the fundamental mechanisms of SoA and RoA are compared regarding to connectivity, compatibility, scalability, robustness and security. Resulting out of this comparison, both approaches are suitable as the base of an IoT middleware. Nevertheless, RoA has a lack of supporting bi-directional communication and real-time analysis while SoA rapidly become a heavy middleware solution. Therefore, the use of a mixed-up middleware is recommended.

1 INTRODUCTION

Over the last decades, the internet developed into an essential part of the everyday life. Its focus shifted from an information network to a bidirectional communication network (Fielding and Taylor, 2000). In the era of mobile devices this development induces the omnipresence of communication channels. For instance, people communicate via smartphones, smartwatches, tablets and other web compatible devices with each other (Gubbi et al., 2013). Besides, the interaction of humans with smart devices gets easier by the advancement of voice, gesture and emotional recognition (Fielding and Taylor, 2000). The next logical step will be on one hand to enable the communication between humans and non-smart, physical things and on the other hand the communication among the things. This conception is summarized under the umbrella term Internet of Things (IoT). According to (Haller et al., 2009), the IoT builds a bridge between the physical and the digital world by integrating physical objects seamlessly into the current internet. Therefore, a reference architecture shows on which one or more IoT applications are based (Bandyopadhyay and Sen, 2011). The first challenge of the IoT is to extend those objects in a way that allows them to connect to a network. Thus, each physical object needs a digital representation of itself which can be realised by a barcode or an RFID chip (Gazis et al., 2015). More complex physical things can consist out of sensors to collect data from its environment. At this point, the IoT consists out of distributed and heterogenous things which use different communication channels and various types of data representation. Therefore, a middleware layer is needed to create an interoperability across the heterogenous things. Other requirements that needs to be managed by the middleware are connectivity, compatibility, scalability, robustness and security. In order to bundle, evaluate and use the data generated by various things, an application layer is needed. This layer indicates a user interface to establish the communication with the enduser (Bandyopadhyay and Sen, 2011). In this context, the enduser is a human being which accesses the information generated by the IoT. Since the middleware builds the core of the IoT architecture, the choice of its architecture is a significant decision. In common, the Service oriented Architecture (SoA) or the Resource oriented Architecture (RoA) is used (Gazis et al., 2015; Guinard et al., 2012). However, both approaches are originally designed to construct (web) services at distributed networks like the internet (MacKenzie et al., 2006; Richardson and Ruby, 2008). Therefore, it needs to be considered whether

Kubela, J., Pohl, M., Bosse, S. and Turowski, K.
Internet of Things Middleware: How Suitable are Service-oriented Architecture and Resource-oriented Architecture.
DOI: 10.5220/0006702202290236
In *Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security (IoTBDS 2018)*, pages 229-236
ISBN: 978-989-758-296-7

these approaches are suitable for the IoT. Currently there are no researches dealing with this problem in an acceptable scope. Thus, this paper shall show if SoA, RoA or both approaches are adoptable to the IoT. In the following chapter the concept of SoA and RoA is shortly explained. Section 3 of this paper gives a review about related literature by analysing various papers and pointing out named reasons for using SoA or RoA as a middleware solution. The forth chapter names essential IoT requirements and which characteristics of SoA and RoA fulfils them. Chapter 5 compares the main differences of both approaches regarding the named IoT requirements. In the end a short outlook for future work is given.

2 DEFINITION OF SoA AND RoA

SoA as well as RoA were not actually created to be used in the field of IoT (Gazis et al., 2015; Guinard et al., 2012). However, both architectures share the goal to increase the interoperability of heterogeneous, distributed objects (Guinard et al., 2011). Although SoA and RoA share the same goal, both architectural schemas are based on a different root concept. In the following, the principle of both architectural approaches is explained.

2.1 Service-oriented Architecture (SoA)

SoA is based on the root idea of splitting complex systems into small components called services (Bandyopadhyay and Sen, 2011). Those SoA services are defined as a logical, self-contained, independent and reusable unit of tasks (Bean, 2009; Laskey and Laskey, 2009; Schmidt et al., 2005; Spiess et al., 2009) and are commonly abstracted from business processes (Laskey and Laskey, 2009). From the user point of view, a service is a black box that receives input parameters and returns a specific output (Laskey and Laskey, 2009; Spiess et al., 2009). Applications can access the service interface and execute one or more services to accomplish their task. Moreover, new services can be created by chaining or enhancing existing services. Considering that, two services communicate by exchanging messages (Bean, 2009). But before two services are able to communicate they have to find each other using the Service Registry. The Service Registry handles the metadata of all published services. Those can be deposited as UDDI, WSDL or other formats (Wan et al., 2006). The services registered at the Service Registry does not have to be owned or hosted by the same person or company. Moreover, it allows the distribution of the SoA network (Bean, 2009). Thus, a service can find other services automatically by searching at the Service Registry. SoA is based on a high degree of freedom by choosing transport (like HTTP and FTP) and communication protocols (like SOAP and XML) which builds the interface of a service (Laskey and Laskey, 2009; Schmidt et al., 2005; Wan et al., 2006). In the field of SoA web services a commonly used technology is the WS-*-specification (Gazis et al., 2015). The result of this freedom is that the uniform interface of services is harmed if different protocols are used. To admit this problem, an Enterprise Service Bus (ESB) can be established. The ESB unifies all mechanisms of the SoA in one component (Schmidt et al., 2005). A SoA using an ESB as the central component which handles all messages of the service network is shown in (Bean, 2009).

2.2 Resource-oriented Architecture (RoA)

The term RoA was established in the year 2007 (Richardson and Ruby, 2008). It describes a software architectural style which is based on the Representational State Transfer (REST) paradigm stated out in (Fielding and Taylor, 2000) and the transfer protocol HTTP. Using these technologies, the root idea of RoA is to split complex systems into resources which are encapsulated in a unique interface (RESTful resources). Following the definition (Richardson and Ruby, 2008), everything can be defined as a resource if it has a digital representation that is addressable over a Unique Resource Identifier (URI). The interface of each resource is restricted to the methods PUT, GET, POST and DELETE. The POST method assigns a resource to the used URI (Guinard et al., 2011). If the URI is already assigned to a resource, the old resource is replaced. Afterwards, the GET method is used to read the data provided by the resource. As a result, the data represented in a predefined format is returned. The most common representations are XML, JSON and HTML (Guinard et al., 2011; Richardson and Ruby, 2008). Also, it is possible that a resource consists out of various representations. In this case, the calling instance can choose which representation fits best for the handled use case (content negation) (Guinard et al., 2011). Another method provided by HTTP is the PUT statement. It is used to update the data represented by the resource (Guinard et al., 2011). The last by standard HTTP supported method is DELETE and is necessary to free an URI by deleting the assigned resource. Based on the forced use of HTTP and REST, all resource of the RoA can communicate with each other without

the need of an intermediary. Moreover, the connection between resources is following the client-server principal. As a result, a resource (client) can only pull data from another resource (server). In (Fielding and Taylor, 2000), the authors amplify this concept by claiming that each request has to contain all information necessary to process it. This characteristic is named as statelessness (Fielding and Taylor, 2000; Richardson and Ruby, 2008). Another assumed characteristic is the possibility of resources to cache data that is already received and marked as cacheable. The resources communicate which each other and are linked to enable a navigation through them (Fielding and Taylor, 2000; Richardson and Ruby, 2008).

3 RELATED WORKS

Some researches already try to show the suitability of RoA and SoA as an IoT middleware. Nevertheless, either the focus of their research is on a specific IoT use case or the justification of their decisions is limited. As a result, none of the found papers analyses the fitting of fundamental characteristics of SoA and RoA to common IoT requirements in an adequate scope. A Web of Things (WoT) architecture using the concept of RoA as middleware is presented in (Guinard et al., 2011). As a reason for using RoA instead of SoA Guinard et al. mention that SoA is often to heavy and complex for the field of IoT. Moreover, the strict use of HTTP and REST simplifies the integration of things into the current internet. In contrast, a SoA based middleware only uses the Internet as a transport infrastructure to exchange data inside of the SoA network. On one hand, the authors conclude that WoT applications stand out due to high scalability, high flexibility and simplicity without justifying their conclusion. On the other hand, they state out that working with real-time data streams, avoiding a request overhead, low latency and searching for things are challenges of an RoA based middleware. A comparison between the use of RoA and SoA as an IoT middleware regarding the developers' perspective is provided. The result of the study is that the concept of RoA "is easier to learn, more intuitive and more suitable for programming IoT applications" than SoA (Guinard et al., 2012). Otherwise, the authors mention that the fixation of RoA to HTTP does not allow the use of various security mechanisms. Instead, SoA can make use of a wide range of security tools. Nevertheless, they raise the question if voluminous security mechanisms like they are provided by WS-* are necessary. Another result of their study is that the participations would rather choose SoA for business applications based on the superior security mechanisms and the possibility to use a Service Registry. In (Spiess et al., 2009), a similar result is concluded. The authors create a SoA middleware to adapt business processes to the IoT. As reasons for their choice the authors name the discovery functionality provides by the Service Registry and the scenario-specific way of creating new services based on processes. Another aspect is the need for event-handling and real-time processing in the field of manufacturing. Therefore, the created SOCRADES architecture uses amongst others a Service Catalogue and a Service Monitor. Furthermore, they suggested the various security aspects provides by SoA. Nevertheless, they decided to use a SoA based middleware without considering the possibility to use RoA instead (Spiess et al., 2009). Moreover, it seems that the decision of using SoA is made because of its origin to digitalise business processes. A comparison of different IoT middleware solutions is provided in (Chaqfeh and Mohamed, 2012). At the beginning, the authors name common IoT requirements as well as known IoT middleware solutions like SoA and RoA. Afterwards, a table is presented that shows which middleware solution fulfils the given requirements. According to the authors, SoA fulfils all stated requirements. In contrast, RoA has a lack in security and providing adequate interfaces. Unfortunately, the authors neither justify their conclusion nor name mechanisms of SoA and RoA that cover the name requirements.

4 IoT REQUIREMENTS COMPARED TO SoA AND RoA

In the following, the fundamental aspects of SoA and RoA are compared to common IoT requirements. At first, each IoT requirement and its sub-requirements is defined. Afterwards, mechanisms of SoA and RoA that suits the requirement are named and rated. Table 1 shows the results of the performed comparison. The first column names common IoT requirements that should be fulfilled by an IoT middleware. Those are identified from various academic papers like (Bandyopadhyay and Sen, 2011; Bonomi et al., 2014; Guinard et al., 2011; Haller et al., 2009; Kolozali et al., 2014; Pang et al., 2015). The reviewed requirements are connectivity, compatibility, scalability, robustness and security. At the next two columns of the table rate the ability of SoA and RoA to handle these requirements. For this, the symbol "-" signalises that the current requirement cannot be fulfilled by the architectural approach in a suitable way. Additionally, a "o" shows that the chosen architecture can

handle this requirement partly. The symbol "+" signalizes that the characteristics RoA or SoA are suitable to fulfil this requirement.

Table 1: IoT Requirements and their Accomplishment by SoA and RoA.

Requirement	SoA	RoA
Connectivity		
Interoperability	o	+
Addressability	+	+
Loose Coupling	o	+
Self-Connection	o	-
Searchable Things	+	+
Bi-Directional Communication	+	o
Compatibility		
Middleware Connectivity	o	o
Back-/Forward Compatibility	o	o
Scalability		
Various Number of Things	+	+
Latency	o	o
Real-Time Processing	+	-
Robustness		
Drop out of Things	+	-
Validity of Data	+	-
Security		
Authentication	+	+
Privacy	+	+

4.1 Connectivity

The term connectivity unifies all characteristics of an IoT middleware to form a homogenous network out of a heterogenous mass of things (Bandyopadhyay and Sen, 2011; Bonomi et al., 2014). It is a fundamental characteristic of the IoT that all things have the ability to communicate with each other without the need of human interaction (Bandyopadhyay and Sen, 2011; Haller et al., 2009). This sub-requirement is referred to as interoperability. To enable a communication, things have to be addressable (Haller et al., 2009). Moreover, the coupling between the things has to be loosely which means that things are modular and independent of each other and the communication channel of other things (Bandyopadhyay and Sen, 2011). Another part of connectivity is the self-connection of things which is bond to the discovery mechanism of the network (Gubbi et al., 2013; Sundmaeker et al., 2010). The possibility of the autonomously connection of new things is necessary because of the fact that the IoT is a dynamic network which allows things to move and connect to another middleware (Sundmaeker et al., 2010). The last sub-requirement of connectivity is the bi-direction communication (Bandyopadhyay and Sen, 2011; Haller et al., 2009; Spiess et al., 2009). Thus, in some IoT use cases things have to push data to an application or other things. In this case, the pull method of the client-server principal is not sufficiently. Creating a SoA based IoT middleware means that things, parts of things or associations of things are interpreted as services. As a result, a number of heterogenous things is turned into a homogenous network of services. The services collaborate irrespective their underlying technologies by using their service interface (Bean, 2009; Haller et al., 2009; Spiess et al., 2009). Resulting, the interoperability and loosely coupling of things is received. Nevertheless, the architectural design of SoA does not specify how the interface has to be build. So, the designer of the IoT middleware has to define standard protocols that build the interface. Alternatively, a translator has to be established as intermediary (Bean, 2009). Besides the effort of an initial implementation and continuous maintenance (Bean, 2009), the intermediary harms the requirement of loosely coupling by establishing a dependence on the translator (Schmidt et al., 2005). Another violation of the loose coupling at a SoA occurs if a session state can be created during the communication of two services. A possible situation of this circumstance is the use of various security mechanisms (Hafner et al., 2009). Nevertheless, it is mentioned that the registration of services can be too complex for IoT objects because of their limited amount of computing power and data memory (Guinard et al., 2012). If a new thing needs to get part of the network it just has to be published at the Service Registry. The last mentioned sub-requirement of connectivity is the bi-directional communication. Because of the high degree of freedom, each service can handle an arbitrary number of methods at its interface. Moreover, SoA is not bound to the client-server principal whereby the pushing of data is possible like it is needed for event handling (Spiess et al., 2009). Like SoA the RoA approach can be adopted as an IoT middleware. Thus, all things, part of things or associations of things are interpreted as resources. Those encapsulate the things by giving them a unique interface and hiding their fundamental implementation. As a result, a homogenous network of independent resources arises which only allows a collaboration using the standard HTTP methods. The handling of various message protocols like JSON and XML is handled at a RoA by the use of content negotiation or the use of different resources for each content representation style (Richardson and Ruby, 2008). As a result, the interoperability inside of a RoA network is given. Additionally, the characteristic of stateless connections supports the requirement of loosely cou-

pled things. In contrast to SoA, RoA do not need a structure like a Service Registry to make resources discoverable. Thus, all RESTful things can be found easy if they are described via microformats or similar (Gubbi et al., 2013; Guinard et al., 2011). On one hand, the use of HTTP ensures that also new things are immediately integrated to the RoA network. On the other hand, non-RESTful things need to get a RESTful interface or proxy to become part of the RoA network (Guinard et al., 2011; Richardson and Ruby, 2008). The requirement of bi-directional communication is contrary to the characteristic of RoA that data can only be pulled (client-server principal). Thus, the realisation of a bi-direction communication is only possible if a high network traffic and latency is accepted. One possible solution is the use of web hooks which needs a specific and public URI that is called by another resource if a specific event occurs (Guinard et al., 2011).

4.2 Compatibility

The vision of a ubiquitous IoT intends the existence of various IoT applications and also loose things. Nevertheless, the IoT is a network where all components should be able to communicate with each other. Therefore, the compatibility between various middleware implementations is necessary to enable a cross-application collaboration. Thus, an IoT middleware has to handle backward and forward compatibility to avoid communication errors based on different protocol versions. The implementation of a SoA designed middleware is based on a high degree of freedom, as mentioned before. Therefore, the use of an intermediary to translate different transport and messaging protocols is necessary to ensure the communication inside a network and between various networks (Bandyopadhyay and Sen, 2011). The use of an ESB for this translation task is suggested in (Schmidt et al., 2005). As a result, the requirement of compatibility is fulfilled. Besides translation messages the ESB can also queue message and provides a synchronous as well as asynchronous messaging. Nevertheless, the effort of implementing an ESB respectively the extension of an ESB needs to be considered. On one hand, the initial effort is demanded on the number of various protocols used by the connected things and networks. On the other hand, a recurring effort appears by the continuous addition of new protocols and protocol versions. This dependence on the ESB results in possible single point of failure. RoA benefits from its tight relation to REST and HTTP. Thus, the connection to other RoA based network is automatically established without any extra effort. In contrast, the things of a non-RoA based network has to be covered by an RESTful interface which can result in a high effort. A smart gateway which is realised by a tiny server that acts as a proxy for the things is recommended in (Guinard et al., 2011). Nevertheless, the advantage of this approach is that no central intermediary is needed and thus no single point of failure is created. However, the case of incompatibility of protocols because of the use of different versions is rare for a RoA middleware. The reason for this is that the HTTP protocol is backwards compatible by default (Richardson and Ruby, 2008). Moreover, all versions of HTTP support the methods PUT, GET, POST and DELETE. Nevertheless, the forward compatibility is not guaranteed. Thus, a solution to handle forward incompatibility is to catch and handle HTTP error codes (Richardson and Ruby, 2008) or to use the HTTP header to make clear that the called resource fits the requirements.

4.3 Scalibility

The IoT paradigm assumes a dynamic and evolving environment. Thus, the number of things connected to a middleware can vary as well as the number of applications based on this middleware (Bandyopadhyay and Sen, 2011; Kruger and Hancke, 2014). Therefore, an IoT middleware has to perform well irrespective of the number of handled things (Chaqfeh and Mohamed, 2012). This characteristic is referred to as scalability. Moreover, scalability means that the number of accesses to things triggered by various applications should not affect the performance of the IoT network spanned by the middleware. An indicator for this circumstance is the latency which describes the time needed to answer a request. Consequently, the middleware has to handle and transfer the data generated by the things in-time. The number of things connected to the network influences a SoA middleware in different ways. On one hand the size of the Service Registry grows or shrinks with the number of services. On the other hand, the number of things influences the status monitor which has to handle more heartbeats. Both circumstances can be handled by providing flexible processing and storage power by using cloud computing (Bonomi et al., 2014). Additionally, the number of calls of one as well as all services influences the SoA middleware. The transmitter can be scaled by using cloud computing (Schmidt et al., 2005). Thus, the limiting factor delays on the things which mostly have a small processing power. Therefore, SoA supports different mechanisms to relieve the load on the things. One aspect is the Service Broker which can distribute messages to similar services if one service is busy (Schmidt et al.,

2005; Spiess et al., 2009). Another aspect is the possibility to queue requests so that they can be handled one after another without overloading the thing (Du et al., 2011; Garces-Erice, 2009). Consequently, a high number of requests can result in a high latency. Another aspect that has influence on the latency is the translation of each request by the ESB (Du et al., 2011). Thus, also the handling of real time data is influenced. To solve this problem, the use of a real-time assurance mechanism at the ESB is proposed in (Du et al., 2011). This mechanism is responsible to prioritise all messages and giving them a deadline. A similar approach is an Event Scheduler (Garces-Erice, 2009). The main focus of RoA is to handle a various number of resources respectively things by handling them loosely coupled and independent. Moreover, RoA has no central intermediary which handles the connection between the resources. Thus, a point-to-point communication is established between the resources which avoids a performance influence by a fluctuating number of things. Similar to SoA, the communication bottleneck of RoA is the low processing power of the things. As a result, several calls of one resource at once has to be handled by the resource itself (Richardson and Ruby, 2008). The statelessness of a resource ensures that one access to the resource is not influences by other, simultaneous accesses (Richardson and Ruby, 2008). The result is that no connection state needs to be loaded and that parallel processing is supported. Moreover, RoA provides a client side cache which allows the buffering of cacheable data (Guinard et al., 2011). Another aspect to handle multiple requests is the concept of a layered system which allows the redirection of requests to sub-resources. The layered system concept results in a higher latency because of the intermediary that is embodied by the resource (Guinard et al., 2011). On the contrary, the possibility to cache data and redirect requests to sub resources can lower the latency by handling the request faster (Guinard et al., 2011; Richardson and Ruby, 2008). As mentioned before, real-time processing is hard to handle using RoA. The reason for that is that straight HTTP and REST only support the pulling of data. Thus, the use of web hooks to implement event handling and AtomPub for implementing data streaming at a RoA middleware is suggested (Guinard et al., 2011).

4.4 Robustness

The robustness of a network describes its ability to handle the availability of things (Haller et al., 2009; Kolozali et al., 2014). Another aspect of robustness is the validity of measured and supported data (Kolozali et al., 2014). A SoA based middleware can handle the drop out of service by using a status monitor (Du et al., 2011). Each service frequently sends a message (heartbeat) to this monitor to show that it is still working. In combination with an ESB the failure of a service can be regulated by searching for an alternative service with equal functions (Haller et al., 2009). Moreover, the ESB can support a validity check of data by reading and checking all passing messages. A disadvantage of this process is that the latency will increase as well as the workload of the ESB. Based on the lack of bi-direction communication and event handling, the detection of dropped out resources at a RoA networks is difficult. One solution can be to define one or more resources which checks if other resources are still available. The disadvantage of this solution is that an information and traffic overhead will result out of the frequently send GET requests to all resources to get their status (Guinard et al., 2011). As a result, the application designer has to handle a possible drop out of a resource and implement an alternative behaviour like the searching for similar resources. The advantage of this approach is that the failure of one resource is handled by each application itself which results in an individual and therefore suitable solution. Nevertheless, the rising effort of the application design has to be mentioned. This circumstance can be avoided by extending a mashup software by an autonomous error handling (Guinard et al., 2011). This solution can also be used to check the validity of data at the application layer. Moreover, the data validity check can also be moved to the things layer.

4.5 Security

The creation of networks by linking-up real-world things and connect them to the internet is not free of dangers. The security of IoT, IoT application and IoT architectures is an important aspect that should be mentioned. The number of connected things and the amount produced data offers a huge contact surface for cyber criminals. Therefore, various security aspects and typical attacks that have to be mentioned while create an IoT middleware are analysed (Farooq et al., 2015; Riahi et al., 2013; Gou et al., 2013). It should be mentioned that an effective security level can only be accomplished if all layers of the IoT structure are secured. Based on the fact that SoA is not bound to specific technologies, a wide range of security tools can be used. Standardised WS-Security in combination with SOAP is suggested and by the use of security tokens an authentication process is established (Tiburski et al., 2015). Additionally, XML Encryption ensures the privacy of data and wards man-

in-the-middle attacks off (Tiburski et al., 2015; Farooq et al., 2015). Nevertheless, the implementation of these security standards can result in a high effort especially regarding the ESB which can be a security vulnerability, too. Thus, Hafner et al. (Hafner et al., 2009) created a blueprint for Security as a Service (SeAAS). As already mentioned, RoA is based on HTTP and REST. Therefore, HTTP Basic, HTTP Digest or WS-Security Extension are suggested to use for authentication (Richardson and Ruby, 2008). However, both methods are not save (Lee et al., 2015). HTTP Basic can be a victim of replay and injection attacks while HTTP Digest is not save regarding man-in-the-middle attacks. Therefore, the authors developed a method called ID-based Authentication which performs authentication using private and public keys in combination with the resource URIs. Furthermore, the existing APIs of social networks can be used as a proxy to authenticate users and manage their access to resources (Guinard et al., 2011). The aspect of privacy can be fulfilled by the use of HTTPS instead of HTTP which encrypts the transferred data (Richardson and Ruby, 2008; Clark and van Oorschot, 2013).

5 DISCUSSION AND CONCLUSION

The performed analysis of SoA and RoA shows that both approaches do not fulfil all requirements raised by the IoT perfectly. Table 1 shows that SoA supports all requirements at least in an elementary way. The deciding factors of SoA are the ESB and the freedom to choose the used protocols free. On one hand, the ESB allows the management and monitoring of all services and its communication which is useful to provide real-time processing, robustness and security. On the other hand, the ESB causes a high effort by its initial implementation and continuous maintenance. Moreover, the use of an intermediary results in a single point of failure and harms the loose coupling of things. Nevertheless, SoA cannot be used as an IoT middleware without providing an ESB or a similar intermediary. The high degree of freedom to choose the service interface harms the interoperability of services but allows to use use-case specific protocols. Regarding Table 1, RoA does not support all requirements of the IoT. The reason for not fulfilling the requirements self-connection, real-time processing and robustness is that the REST concept does not allow the pushing of data. Thus, a bi-directional communication can only be implemented with a huge effort and by causing unnecessary traffic for example by using web hooks. Omitting the lack of bi-directional communication, RoA is well suited for IoT applications because of its strict definition of a unique interface which avoids the use of an intermediary. As a result, a lightweight and simple network can be build. Moreover, RoA benefits from the dependence on HTTP by embedding the IoT into the existing internet. In contrast, SoA uses the internet as an infrastructure to exchange its data instead of merging both approaches. All in all, SoA as well as RoA can be suitable to build a IoT middleware. The decision to one of the two approaches depends on the specific use-case and the future use of the created network. The analysis of SoA and RoA regarding various IoT requirements can be the basis for this decision. Thus, RoA is recommended if no real-time analysis or event-handling is necessary because of its lightweight and connectedness to the existing internet. In contrast, SoA should be used if complex processes should be designed or real-time analysis are needed. Moreover, the lack of security mechanisms of RoA mentioned in (Guinard et al., 2012; Chaqfeh and Mohamed, 2012) can be proven. Nevertheless, the security mechanisms of RoA is suitable for common IoT applications. It is also possible to match both approaches to provide from the lightweight of RoA and the flexibility of SoA. Future researches can build a mix-up architecture of SoA and RoA based on the reviewed IoT requirements and mechanisms. Moreover, the security mechanisms of both middleware solutions need to be reviewed in more detail. For this, the things layer and the application layers needs to be included because of the high dependency in the field of security. Another research topic is to evaluate how strong the named IoT requirements depend on each other and which architectural characteristics are necessary to fulfil all requirements.

REFERENCES

Bandyopadhyay, D. and Sen, J. (2011). Internet of things: Applications and challenges in technology and standardization. *Wireless Personal Communications*, 58(1):49–69.

Bean, J. (2009). *SOA and web services interface design: principles, techniques, and standards*. Morgan Kaufmann.

Bonomi, F., Milito, R., Natarajan, P., and Zhu, J. (2014). *Fog Computing: A Platform for Internet of Things and Analytics*, pages 169–186. Springer International Publishing, Cham.

Chaqfeh, M. A. and Mohamed, N. (2012). Challenges in middleware solutions for the internet of things. In *2012 International Conference on Collaboration Technologies and Systems (CTS)*, pages 21–26.

Clark, J. and van Oorschot, P. C. (2013). SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In *2013 IEEE Symposium on Security and Privacy*, pages 511–525.

Du, L., Duan, C., Liu, S., and He, W. (2011). Research on service bus for distributed real-time control systems. In *2011 6th IEEE Joint International Information Technology and Artificial Intelligence Conference*, volume 1, pages 401–405.

Farooq, M. U., Waseem, M., Khairi, A., and Mazhar, S. (2015). A critical analysis on the security concerns of internet of things (IoT). *International Journal of Computer Applications*, 111(7).

Fielding, R. T. and Taylor, R. N. (2000). *Architectural styles and the design of network-based software architectures*. University of California, Irvine Doctoral dissertation.

Garces-Erice, L. (2009). Building an enterprise service bus for real-time SOA: A messaging middleware stack. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, volume 2, pages 79–84.

Gazis, V., Grtz, M., Huber, M., Leonardi, A., Mathioudakis, K., Wiesmaier, A., Zeiger, F., and Vasilomanolakis, E. (2015). A survey of technologies for the internet of things. In *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1090–1095. IEEE.

Gou, Q., Yan, L., Liu, Y., and Li, Y. (2013). Construction and strategies in IoT security system. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 1129–1132.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645 – 1660. Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications Big Data, Scalable Analytics, and Beyond.

Guinard, D., Ion, I., and Mayer, S. (2012). *In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective*, pages 326–337. Springer Berlin Heidelberg, Berlin, Heidelberg.

Guinard, D., Trifa, V., Mattern, F., and Wilde, E. (2011). From the internet of things to the web of things: Resource-oriented architecture and best practices. *Architecting the Internet of things*, pages 97–129.

Hafner, M., Memon, M., and Breu, R. (2009). SeAAS-a reference architecture for security services in SOA. *Journal of Universal Computer Science*, 15(15):2916–2936.

Haller, S., Karnouskos, S., and Schroth, C. (2009). *The Internet of Things in an Enterprise Context*, pages 14–28. Springer Berlin Heidelberg, Berlin, Heidelberg.

Kolozali, S., Bermudez-Edo, M., Puschmann, D., Ganz, F., and Barnaghi, P. (2014). A knowledge-based approach for real-time IoT data stream annotation and processing. In *2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*, pages 215–222.

Kruger, C. P. and Hancke, G. P. (2014). Implementing the internet of things vision in industrial wireless sensor networks. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, pages 627–632.

Laskey, K. B. and Laskey, K. (2009). Service oriented architecture. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(1):101–105.

Lee, S., Jo, J. Y., and Kim, Y. (2015). Method for secure RESTful web service. In *2015 IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS)*, pages 77–81.

MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., Metz, R., and Hamilton, B. A. (2006). Reference model for service oriented architecture 1.0. *OASIS standard*, 12:18.

Pang, Z., Zheng, L., Tian, J., Kao-Walter, S., Dubrova, E., and Chen, Q. (2015). Design of a terminal solution for integration of in-home health care devices and services towards the internet-of-things. *Enterprise Information Systems*, 9(1):86–116.

Riahi, A., Challal, Y., Natalizio, E., Chtourou, Z., and Bouabdallah, A. (2013). A systemic approach for IoT security. In *2013 IEEE International Conference on Distributed Computing in Sensor Systems*, pages 351–355.

Richardson, L. and Ruby, S. (2008). *RESTful web services*. O'Reilly Media, Inc.

Schmidt, M. T., Hutchison, B., Lambros, P., and Phippen, R. (2005). The enterprise service bus: Making service-oriented architecture real. *IBM Systems Journal*, 44(4):781–797.

Spiess, P., Karnouskos, S., Guinard, D., Savio, D., Baecker, O., d. Souza, L. M. S., and Trifa, V. (2009). SOA-based integration of the internet of things in enterprise services. In *2009 IEEE International Conference on Web Services*, pages 968–975.

Sundmaeker, H., Guillemin, P., Friess, P., and Woelfflé, S. (2010). Vision and challenges for realising the internet of things. *Cluster of European Research Projects on the Internet of Things, European Commision*, 3(3):34–36.

Tiburski, R. T., Amaral, L. A., Matos, E. D., and Hessel, F. (2015). The importance of a standard security archit ecture for SOA-based iot middleware. *IEEE Communications Magazine*, 53(12):20–26.

Wan, K.-M., Lei, P., Chatwin, C., and Young, R. (2006). Service-oriented architecture. pages 998–1002.