KU LEUVEN

FACULTY OF ECONOMICS
& BUSINESS

# A Reference Architecture for IoT-Enhanced Business Processes

**Tevye JACOBS**[1]

Master's Thesis Submitted for the Degree of
Master of Business Administration
Obtained grade: 15/20

Supervisor: Prof. dr. Estefanía SERRAL

Cosupervisor: Faruk HASIĆ

Academic Year: 2018–2019

**FACULTY OF ECONOMICS AND BUSINESS - CAMPUS BRUSSELS**
**WARMOESBERG 26 – B 1000 BRUSSELS BELGIUM**

[1] *Email address: tevye.jacobs@hotmail.be, phone: +32476727297, address: Kapelweg 36, 2960, Brecht.*

A reference architecture for IoT-enhanced business processes

Tevye Jacobs

**List of Abbreviations**

| | |
|---|---|
| API | Application Programming Interface |
| AWS | Amazon Web Services |
| BP | Business Process |
| BPA | Business Process Architecture |
| BPEL | Business Process Execution Language |
| BPM | Business Process Management |
| BPMN | Business Process Model and Notation |
| BPMS | Business Process Management System |
| CoAP | Constrained Application Protocol |
| CRM | Customer relationship management |
| CSV | Comma-separated values |
| ERP | Enterprise resource planning |
| HTTP | Hypertext Transfer Protocol |
| ICT | Information and communications technology |
| IoT | Internet of Things |
| IoTA | Internet of Things Architecture |
| IP | Internet Protocol |
| IT | Information technology |
| JSON | JavaScript Object Notation |
| QoS | Quality of Service |
| LoE | Level of Elaboration |
| M2M | Machine to machine |
| MES | Manufacturing execution system |
| MQTT | Message Queuing Telemetry Transport |
| O2O | Object to object |
| OT | Operational technology |
| Pub/sub | Publish-subscribe |
| REST | Representational State Transfer |
| ROA | Resource-oriented architecture |
| S3 | Service-Oriented Solution Stack |
| SOAP | Simple Object Accessing Protocol |
| SoC | Separation of Concerns |
| TLS | Transport Layer Security |
| URI | Uniform Resource Identifier |
| WAPI | Workflow API |
| WfMC | Workflow Management Coalition |
| WS | Web Service |
| XACML | eXtensible Access Control Markup Language |
| XML | eXtensible Markup Language |
| XPDL | XML Process Definition Language |

# Table of Contents

**Abstract**
Future business processes are anticipated to be oriented on ensuring intelligence and actionable knowledge in real-time for all elements participating in the value chain. With the enhanced understanding of Internet of Things (IoT) capabilities, the required information can now be provided by intertwining both paradigms. The common incorporation of IoT into business process architectures is still constrained by the absence of a reference architecture. In this thesis, a reference architecture for the execution of IoT-enhanced business processes is proposed, preceded by an extensive literature review of the current state of the art which allows to establish an architecture to bridge the gap between the IoT and business processes. In this architecture, IoT sensors improve business processes with real-time information provisioning, while simultaneously business processes steer the IoT actuators to physically impact the process environment. The latter is what makes this architecture state of the art and can ultimately lead to fully automated and efficient processes. After being evaluated by comparing the architectural capabilities with other prominent architectures, it is perceived as the most functional and capable reference architecture.

**Keywords:** Internet of Things; Business Processes; Architecture

**JEL Code:** Z00

# 1 Introduction

Modern-day organisations require their business processes (BPs) to be agile, adaptive and flexible. A BP can be defined as “a collection of related events, activities, and decisions that involve a number of actors and resources and that collectively lead to an outcome that is of value for an organization or a customer” (Dumas, La Rosa, Mendling, & Reijers, 2013). Having a business process architecture (BPA) in place provides guidance for the actual modelling of the involved business processes and assures a consistent and integrated collection of process models with the aim to optimize the value creation across the value chain (Dijkman, Vanderfeesten, & Reijers, 2011). Future BPs are anticipated to be oriented on ensuring intelligence and actionable knowledge in real-time for the manufacturing process and all elements participating in the value chain. BPs require valuable information whilst being executed in order to make meaningful decisions. This information can be provided by the Internet of Things (IoT).

IoT allows smart devices, i.e. capable of sensing, identifying, processing, communicating and networking, to be sharing data over a network, i.e. the Internet. In the near future a wide application of IoT to industries is expected due to the rapid advances in technology and industrial infrastructure (Xu, He, & Li, 2014). Sensing and perceiving of the process environment are the fundamental tasks of IoT. With emerging wirelessly sensory technologies, the concept of IoT goes beyond these capabilities and extends to ambient intelligence and autonomous control (Li, Xu, & Zhao, 2015). Sensor data must be aggregated, interpreted and made available to the Business Process Management System (BPMS) in order to trigger BP activities or human tasks (Schönig, Ackermann, Jablonski, & Ermer, 2018). The insights emerging from using the collected data from smart BP actors allow to facilitate advances in operations and customer services through machine failure prevention, real-time production performance improvement, etc. (Breivold, 2017). In other words, IoT sensors improve BPs with real-time information provisioning, while simultaneously BPs enact the IoT actuators to physically impact the process environment, which ultimately can lead to fully automated and efficient processes. Therefore, they are inseparable.

Until recently, the aforementioned paradigms were solely viewed upon as atomic domains. However, both fields could considerably benefit when they are combined (Schönig, Ackermann, Jablonski, & Ermer, 2018). By embedding intelligence through IoT and BP management (BPM) technology, this will help businesses to achieve cost savings and efficiency gains (Schönig, Ackermann, Jablonski, & Ermer, 2018). Also, the business world itself acknowledges that IoT yields a salient benefit, which is demonstrated by the increasing involvement of IoT in organisation's business strategy (Milojevic, 2017). Examples of pioneering industries are manufacturing, transportation and utilities, with anticipated expenditures in 2018 (in USD) of respectively $189 billion, $85 billion and $73 billion (Shirer & Torchia, 2017). In order to successfully combine both fields, an architecture needs to be in place which enables the capabilities of both domains.

In literature, the research gap on the cross-cutting domain of IoT-enhanced BPs (IoT-BPs), is starting to gain more attention. Due to the proliferating research on IoT, many IoT architectures (IoTAs) have been developed in parallel aiming to address slightly different purposes. On the other hand, the field of BP architectures (BPAs) is an established discipline in the academic world, with most reference architectures stemming from the 90's. Articles on combining the two domains are emerging, e.g. by describing the mutual benefits and challenges (Janiesch, et al., 2017). Based on this article, a solution was proposed to tackle a selection of identified challenges, by building an architecture which integrates IoT objects into a BPMS, concentrating on how the communication between both worlds can be established (Schönig, Ackermann, Jablonski, & Ermer, 2018). In addition, a reference architecture has been proposed to integrate IoT technology, using a resource-oriented approach to provide an end-to-end integration architecture (Dar, Taherkordi, Baraki, Eliassen, & Geihs, 2015). As formerly mentioned, both IoT and BPs yield capabilities, and also when combining both paradigms, possibly new capabilities arise which were not visible when examined separately. However, in literature there is no reference architecture which covers all capabilities, as well as the aspects that need to be considered whilst exploiting the opportunities of combining the capabilities. Therefore, in this thesis a reference architecture will be proposed which can be a starting block for future development of a concrete IoT-BP architecture (IoT-BPA).

This paper will contribute to current research, i) by providing a clear overview on the current state of the art in IoTA, BPA and IoT-BPA; ii) by collecting and discussing the requirements which the reference architecture must meet; iii) by proposing and discussing a reference architecture for the execution of IoT-BPs and iv) by evaluating the architecture by comparing the proposed architecture with other prominent architectures that are discussed in the state of the art section.

The remainder of this thesis is structured as follows: firstly, Section 2 discusses the adopted methodology. Next, the research background and state of the art are described in Section 3. Thirdly, Section 4 collects and discusses the common requirements for the reference architecture. Subsequently, Section 5 concerns the proposed reference architecture where the different components are explained. In Section 6, the reference architecture is evaluated and finally, Section 7 concludes the thesis whilst providing guidance for future research.

# 2 Methodology

As depicted in the flow diagram below, the paper will collect and review the state of the art in both domains of IoTA and BPA. Additionally, there will be a small section of current literature which combines both domains. All relevant findings will be discussed, with a focus on capturing the architectural requirements and capabilities. Next, the challenge will be to identify which of the requirements and capabilities are necessary in order to achieve a qualitative reference architecture for IoT-BPs. Subsequently, an architecture will be proposed which accommodates the furnished list of requirements and capabilities, such that the BPs can fully exploit the benefits enabled through IoT technology. Lastly,

the architecture will be evaluated by comparing the proposed reference architecture to existing architectures, which were discussed in the state of the art.

The building of the architecture will be established through a reference model-based approach (Dijkman, Vanderfeesten, & Reijers, 2011). The reason for following this approach, is that the study conducted by Dijkman et al. (2011) showed that architectures which find their roots in reference architectures are perceived useful and most likely to be adopted in practice. It unveiled that a reference model-based architecture scored the highest on all three measured aspects, i.e. ease of use, usefulness, and popularity with these aspects scoring 67%, 62%, and 52% respectively (Dijkman, Vanderfeesten, & Reijers, 2011). Consequently, 39 practitioners active in the field of BPM came to a consensus that reference model-based was the best approach. It is assumed that these findings are relevant for IoTAs as well. In addition, functional requirements vary over time, and therefore it is more useful to establish a flexible, adaptable and extensible reference architecture rather than an architecture which is built to meet the current prominent requirements of the business. The reference models, which will form the roots of the architecture will be discussed in Section 3.

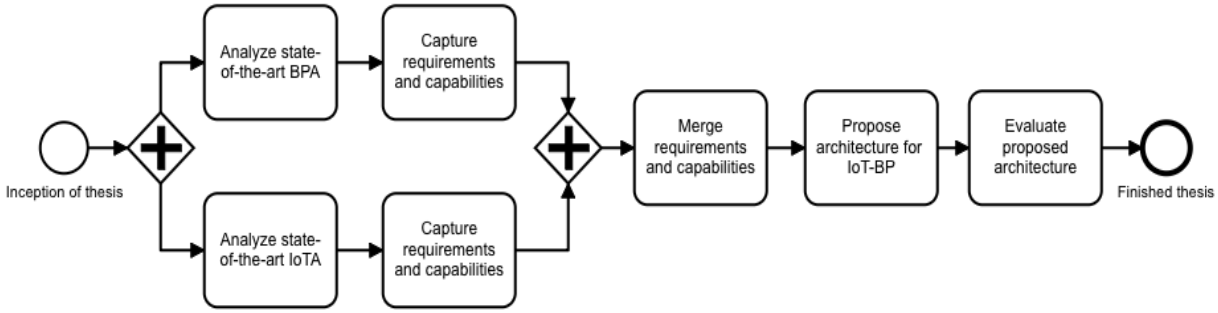


**Figure 1 Adopted methodology**

# 3 State of the art

## 3.1 Business Process Architecture

**"A business process is a collection of related events, activities, and decisions that involve a number of actors and resources and that collectively lead to an outcome that is of value for an organization or a customer." (Dumas, La Rosa, Mendling, & Reijers, 2013).**

The architecture of a BP is defined as "the overview of a set of business processes that reveals their inter-relations, which may be extended with guidelines to determine the various relations between business processes. Having a business process architecture in place provides guidance for the actual modelling of the involved business processes" (Dijkman, Vanderfeesten, & Reijers, 2011). A BPA can be enriched by *containers*, i.e. building blocks which can be reused by different components, along with guidelines for which processes can be contained in a particular container.

Currently, about half of all primary studies is built using a layered approach, while the other half is component-based (Pourmirza, Peters, Dijkman, & Grefen, 2017). Considering the ease of merging an IoTA and BPA, the layered-approach will be focused on since most IoTAs are layered. From these layered BPAs, 71% is designed following the object-oriented Separation of Concerns (SoC) principle. In other words, a modular architecture is currently most common. This SoC separates a presentation layer, a logic/business layer and a persistent/data access layer. Three reference architectures are considered prominent in both the academic world as well as in practice, being the Workflow Management Coalition (WfMC) reference model, the Mercurius reference architecture and the Service-Oriented Solution Stack (S3) reference architecture. These architectures will be discussed below. From

the examined primary studies by Pourmirza et al., none were directly based on the S3 reference architecture, only one was based on the Mercurius reference architecture and 35% followed the WfMC reference model. Note that 60% of the architectures were designed from scratch, and thus not based on any reference architecture. Additionally, well-known issues such as dynamicity and security are mostly ignored in current reference BPA. There appeared a general lack of modern reference architectures. However, more recent architectures will also be discussed such as the RESTful BPM architecture by Kumaran et al. (2008) and the process execution architecture by Li et al. (2010).

## Reference architectures

Up until today, the WfMC reference architecture is one of the most popular reference architectures in the field of BPA. This architecture provides a high-level overview for workflow management systems. The main aim of this architecture was to achieve interoperability among different workflow products. This was established through a standard set of interfaces and data interchange formats (Hollingsworth, 1994). This architecture, as shown in Figure 2, is designed in a component-based manner, thus not with a layered approach. The workflow enactment engine is placed central in the model, surrounded by interfaces called Workflow APIs (WAPI). These are the connections between the workflow engine and the other components of the architecture. The interaction with BPs is enabled through the interface to Process Definition Tools, i.e. Interface 1, which are used for workflow modelling. These models were stored in a workflow repository, ready to be executed at runtime. Hence, implying a strict distinction between build time and runtime. BPs were modelled with tools developed by specific vendors, and therefore the modelled BPs were heterogenic. The aim of the interfaces is to create a standardized representation of the BPs. Communication through this interface is established based on the XML Process Definition Language (XPDL). Process diagrams expressed in BPMN can be represented with the XPDL package metamodel.

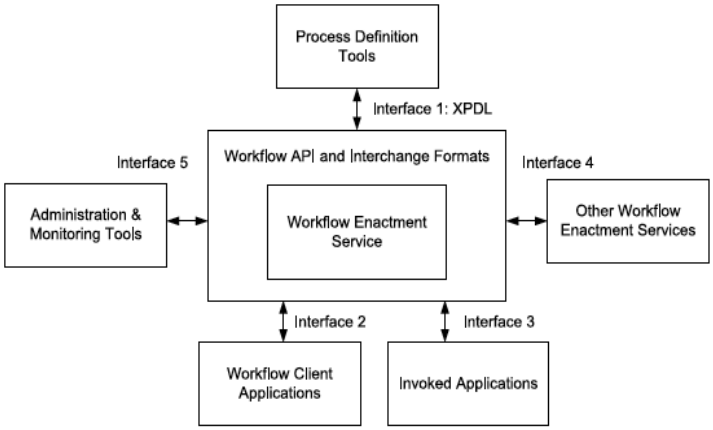


**Figure 2 WfMC reference architecture, retrieved from Weske (2012)**

Currently, recent attainments in service-oriented architectures (SOA) motivate standardization efforts in general. For instance, the XPDL has been an important step towards solving the workflow management system interoperability issue (Weske, 2012).

As opposed to the WfMC reference architecture, the Mercurius reference architecture explicitly includes platform interfaces and data management. To ensure not to be constrained by legacy decisions when building on other reference models, the Mercurius reference architecture was built from scratch. This architecture, as shown in Figure 3, is proposed by Grefen & de Vries (1998). It was considered ideal because of three reasons; i) it was

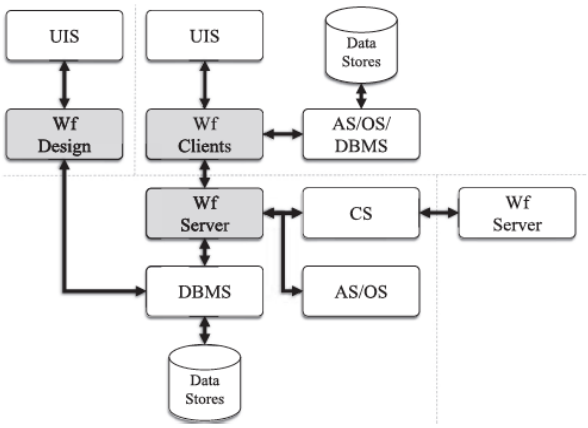


**Figure 3 Mercurius reference architecture by Grefen & de Vries (1998)**

independent of any existing workflow management system, and thus not influenced by legacy decisions, ii) the design was governed by specific design principles which ensure architectural quality and finally, iii) it was designed in context of an extensive project team which ensures completeness of design. From a functional point of view, six software layers can be distinguished, i.e. the user interface system, application systems, workflow management system, database management system, communication system and lastly an operating system. The Wf Server (central on Figure 3) provides all central services for workflow enactment and operates on the central database. The design approach maintained for the Mercurius reference architecture embraces two aspects; *abstraction*, i.e. low-level details are excluded, and *completeness*, i.e. they provide a very complete architecture including parts which are mandatory and parts which are optional. This trade-off always is an important issue for reference architectures. The most recent reference architecture is the Service-Oriented Solution Stack (S3) reference architecture (Arsanjani, Zhang, Ellis, Allam, & Channabasavaiah, 2007). This is a response to the everchanging business environment which demands more flexible and agile information systems (Pourmirza, Peters, Dijkman, & Grefen, 2017). The authors sought to fulfil the architectural needs for the modern-day world, and this would be enabled through a comprehensive nine-layered architecture. This BPA is supposed to provide the required flexibility to furnish a solution to better align information technology (IT) functions, BPs and business goals. Similar to the Mercurius model, the S3 also proposes a comprehensive set of layers, to which the user can decide which subset is required to build their SOA solution. The degree of integration that can be obtained depends on the maturity of the enterprise's service integration. The layered approach also complies with the SoC-principle. Considering two points of view, i.e. from a service provider or a service consumer perspective, allows the S3 to be flexible enough to tightly integrate consumer-provider relation, entirely decouple consumer and provider, or any degree of integration in between.

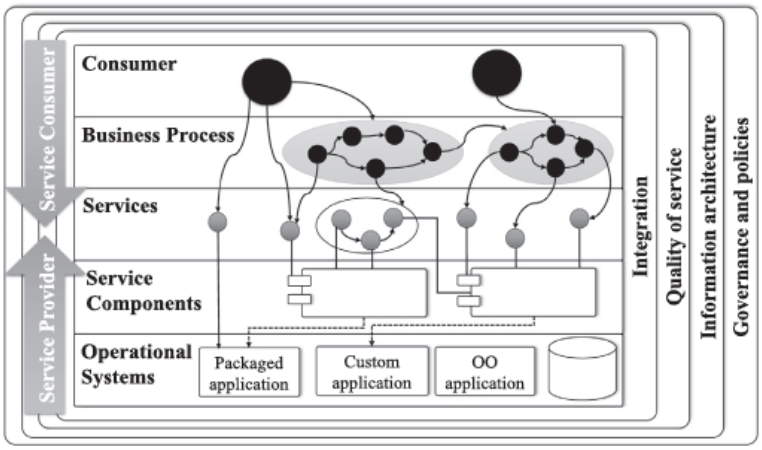


**Figure 4 S3 Reference Architecture by Arsanjani et al. (2007)**

## Other architectures

BPs nowadays can be very large, long running, manipulate vast quantities of data, and require thousands or millions of concurrent process instances (Li, Muthusamy, & Jacobsen, 2010). For that reason, an efficient BPA helps to for instance cut in communications time between workers or enables automation which leads to efficiency gains. Typically, as well as in the aforementioned reference architectures, BPs are executed by a centralised orchestration engine, based on for example the Business Process Execution Language (BPEL). This XML-based language facilitates a SOA by enabling Web-services to share data over a network. However, the distributed property of BPs requires a dispersed architecture such that processes can be enacted by distributed light-weight agents. Hence, a choreography engine is a better fit to be able to cope with this property. Li et al. (2010) propose an orchestration architecture which is in line with the distributed nature of BPs in a globally operating organisation. In their architecture the process execution is distributed across light-weight agents, which all execute a portion of the process. Working with distributed agents implies that parts of the process are executed close to the data they operate on, which eliminates the scalability bottleneck, and even provides additional efficiencies by reducing data streams over the network. Additionally, the distributed architecture allows to change portions of the process, whilst the system remains operational.

Communication across the network is established over a publish-subscribe (pub/sub) pattern, which simplifies the interaction among agents. This layer implies the agents only need to be aware of each other's content-based addresses and thus decouples the agents. However, it is not discussed how bidirectional communication would be established, where agents not only execute a part of the process, but also share acquired data. This is an important feature which will be required by the architecture proposed in this thesis, since not only acquiring IoT data, but also steering IoT devices is aimed to be an architectural capability. The benefit of working with an extensive broker network, is the reduction in network traffic by letting the broker network decide the optimal point to collect and correlate the publications. This also enables fine-grained monitoring without requiring any additional effort. An important drawback, however, is that this architecture requires each involved organisation to deploy a federation of PADRES brokers. Notable features of this architecture, which are useful infrastructural properties for mission-critical enterprise applications include i) user-tuneable fault-tolerance; ii) load-balancing; iii) system policy management; iv) historic data access; and finally, v) route in cyclic overlay networks.

The next architecture seeks to provide a solution for several issues inherent to BPEL, which is believed to be the de facto orchestration language for SOA (Kumaran, et al., 2008). Drawbacks of BPEL include that it is often regarded as not supportive on flexible, scalable, and dynamic BPM (Zimmermann, Doubrovski, Grundler, & Hogg, 2005). Kumaran et al. (2008) acknowledge this challenge and have come up with an execution model based on the architectural style named Representational State Transfer (REST). REST holds several beneficial properties over BPEL-based BPM. These include i) the ability to evolve gracefully as business requirements change; ii) backward navigation and event-driven processes are easy and natural to model in our approach; iii) Restful BPM easily supports conversational business processes using a conversation pattern between the client and server consisting of a pair of self-describing request and response messages; iv) Scalability. The reason why this style is not widely applied to BPM, is argued to be the lack of an effective mechanism to easily manage the URI namespace. Considering the large number of resources in a process, which is specific to BPs nowadays, the management of URI's can be cumbersome. To overcome this issue, an information-centric perspective is taken, and a process model is regarded as a set of communicating behaviour models of business entities. This RESTful approach yields features such as process adaptability, backward navigation and conversational business services. This RESTful BPM architecture is depicted in Figure 5. The resource broker manages both the metadata and the instance data, which are both treated as resources. The client uses REST interfaces to request resources from brokers, e.g. HTTP methods GET, PUT, POST and DELETE (Kumaran, et al., 2008). A client can either be an administrator or regular user. The latter only has the ability to query metadata from the broker, while the former has the ability to either request metadata of a business entity, create a new business entity definition, modify this definition or remove the business entity. Note that REST is a layered architecture, and thus a resource broker may also serve as a client to another resource broker in an extended REST model.

BPM is an established discipline for modelling and executing complex processes in enterprises. If this discipline could be extended such that it can model IoT-aware BPs, this would help achieve a broader implementation of IoT technologies (Haller & Magerkurth, 2011). Moreover, whereas BPs are currently modelled top-down based on the Model-Enact paradigm, the increase in accessible big data can be exploited by working bottom up, as in the Discover-Predict paradigm. This implies collecting and interpreting aggregated data as input for process mining

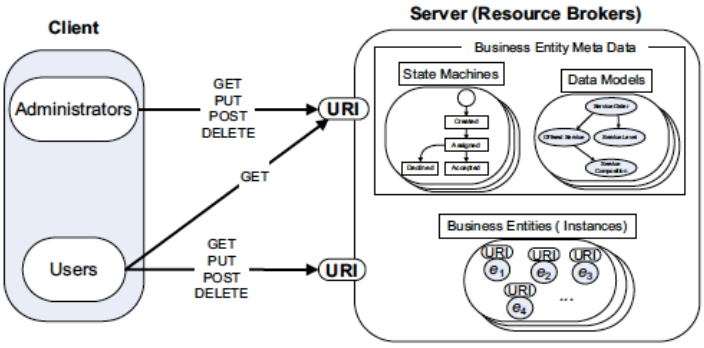


**Figure 5 RESTful BPM architecture by Kumaran et al. (2008)**

algorithms, which can support decision-making (van der Aalst, 2011). Additionally, if the BP execution engine would allow for partial adjustments or improvements to the BPs, this would make the architecture more flexible, efficient and agile. Examples of flexible process engines include the Activiti BPMN execution engine[2], and Camunda BPM execution engine[3]. In Section 3.3, the architecture proposed by Dar et al. (2015) utilises the Acitiviti BPMN execution engine. Also in Section 3.3, the architecture proposed by Schönig et al. (2018) uses the Camunda BPMS and communicates with the workflow engine through the Camunda REST API. The modelled BPs can be executed in two ways; either i) through code generation, where models are executed by translation into an equivalent system code, or ii) by model interpretation, where the engine interprets XML-files and directly executes what is specified in the file (Serral, Valderas, & Pelechano, 2013). Benefits of working with an interpretation engine include; i) faster changes, i.e. a change does not require an entire cycle for regenerating, rebuilding, retesting and redeploying; ii) changes at runtime, i.e. the model is available at runtime; iii) update and scale, i.e. the interpreter can be changed and restarted with the same model, and scaling can be enabled by performing parallel behaviour; and iv) it is more secure since the interpreter provides an additional layer on top of the infrastructure, and everything underneath is abstracted from (Serral, Valderas, & Pelechano, 2013). However, code generation engines have a strong advantage over interpretation models on memory usage and system response time (Serral, Valderas, & Pelechano, 2013).

### Conclusion

The state of the art shows that most reference architectures are built on a layered approach and the SoC-principle is maintained. From current reference BPAs, the WfMC appears to be the most commonly accepted architecture and surprisingly the S3 framework is barely acknowledged as a viable reference architecture although a service-oriented BPA is argued to be the optimal solution to tackle issues such as flexibility and interoperability. Where BPAs used to have a centralised orchestration engine, BPs currently demand an approach which is more in line with the distributed nature of BPs. Hence, a decentralised process execution is argued to make a BPA more flexible and scalable. Moreover, this way reconfiguration of the processes at runtime can be achieved more easily. Additionally, a model interpretation engine is also preferred over a code-generated engine because this also facilitates convenient reconfiguration at runtime. And finally, to overcome the burdens of BPEL, communication is mostly established through REST interfaces.

## 3.2 Internet of Things Architecture

**“The Internet-of-Things is a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual ‘Things’ have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network.” (van Kranenburg, 2007).**

The terms *Internet* and *things* embody a globally interconnected network fostered by sensory, communication, networking and information processing technologies, which may be the successor of the current information and communications technology (ICT) (Li, Xu, & Zhao, 2015). IoT aims to bridge the gap between the physical and the digital world. Hence, the latter is provided with accurate real-time data from the physical environment, captured by sensors and processed by a computer. Simultaneously, the digital world has an impact on the physical world by remotely controlling actuators (Guth, Breitenbücher, Falkenthal, Leymann, & Reinfurt, 2016). However, because these *things* may move geographically and the processes need real-time interaction, it requires an architecture which is adaptive, supports dynamic interaction and ensures seamless integration among devices. The

[2] https://www.activiti.org/

[3] https://docs.camunda.org/

decentralised and heterogenous nature of IoT requires an efficient event-driven capability of the architecture in place. In addition, loose coupling enhances the ease of developing, maintaining and modifying an application, since components can be reused and upgraded efficiently. Abstracting components and their functionalities into services hides the heterogeneity and allows for an improved interoperability. Therefore, the SOA paradigm is considered as the most viable approach to achieve an efficient integration of services provided by IoT devices, since its properties best align with the aforementioned requirements IoT poses on an architecture (Li, Xu, & Zhao, 2015).

A mapping study provided a clear overview of the existing research on IoT solutions, i.e. for smart industries (Breivold, 2017). A challenge in this transformation is the transition from the traditional multi-layered architecture to an open structured service-oriented automation system architecture. According to this study, the least examined topics are architectural aspects (6%, 7 papers) and another 7% (8 papers) discussed business aspects. The contribution types of most primary studies are model (42%) and method (30%), with a shortage in contributions from experience. Additionally, many studies qualify as *solution proposal* and *conceptual proposal*. These insights indicate that most techniques and solutions are still not mature enough for industry-wide adoption (Breivold, 2017). In literature, IoTAs have emerged and mostly provide the same or similar functionalities. The main challenge lays in comparing IoT platforms and finding one that is suitable for certain application fields, as their implementation and underlying technologies differ (Guth, Breitenbücher, Falkenthal, Leymann, & Reinfurt, 2016). Therefore, a reference architecture must provide an abstract overview of the disposable components and clearly define their semantics. In what follows, a reference architecture will be discussed which was based on a comparison of several IoT platforms, i.e., OpenMTC[4], FIWARE[5], SiteWhere[6] and Amazon Web Services IoT[7] (AWS IoT) (Guth, Breitenbücher, Falkenthal, Leymann, & Reinfurt, 2016). The reference architecture by Guth et al. (2016) is argued to be a very abstract overview in order to be broadly applicable. Similar to the reference BPAs, some components, i.e. layers, are optional and thus can be omitted. For example, in case the IoTA is only expected to acquire information about the physical environment, e.g. it measures the tire pressure of a truck, actuators do not have to be included. However, if the IoT device should also be able to inflate or deflate the tire when a certain pressure threshold is exceeded, actuators should be included in the IoTA as well.

## Reference architectures

To commence, the IoT reference architecture discussed first is depicted in Figure 6. This proposal by Guth et al. (2016) is noticeably abstract and has a firm focus on terminology since this is argued to be the key enabler for interoperability among systems. Its devices use drivers, which are software to process data either from sensors or to actuators. These devices can be either i) self-contained, hence depicted as a black box providing a functionality, or ii) connected to another system, e.g. to an IoT integration middleware. The gateway receives a proprietary binary message from the sensors in a device. The received information will then be translated into JSON or XML and the gateway transmits the data to a system in the world wide web. Vice versa, when the gateway receives commands directed towards the actuators. A device can communicate directly with the IoT integration middleware layer if it holds the following features; i) an appropriate communication technology, e.g., WiFi, or Bluetooth, ii) a corresponding transport protocol, e.g. HTTP or MQTT, iii) and a compatible payload format such as JSON or XML. If not, the device communicates through a gateway. Additional functionalities of the middleware layer include a rules engine, device- and user management as well as data aggregation and utilization. This layer is typically accessed through an API, e.g. HTTP-based REST APIs. The application layer provides the user insights by requesting sensor data, or to control the physical actions using

[4] http://www.open-mtc.org/

[5] https://www.fiware.org/

[6] http://www.sitewhere.org/

[7] https://aws.amazon.com/en/iot/

actuators. To allow for scalability, an application can also be another middleware layer to facilitate the integration of multiple systems (Guth, Breitenbücher, Falkenthal, Leymann, & Reinfurt, 2016).

With the Azure reference IoTA, Microsoft proposes an IoT solution that is cloud native, microservice, and serverless based (Microsoft, 2018). This architecture consists out of core subsystems, and optional subsystems, similar to the architecture discussed above. The core subsystems consist of the following layers: i) devices, which can securely transmit and receive data from the cloud; ii) a cloud gateway service, which accepts data and can provide device management capabilities; iii) stream processors, which consume data, integrate the data with BPs and store it; and finally, iv) a user interface to visualise data and facilitate device management. In the architecture by Guth et al. (2016) the functionality in (iii) was included in the IoT integration middleware layer. Next, the optional subsystems are: v) intelligent edge devices, which enable aggregation or transformation of data and on-premise processing, vi) data transformation, which allows restructuring, combination, or transformation of telemetry data received from devices, vii) machine learning, which enables predictive methods to be applied to historical data, and viii) user management to split functionality among different roles and users (Microsoft, 2018).

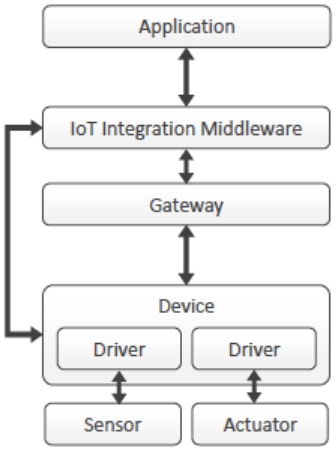


**Figure 6 Reference Architecture by Guth et al. (2016)**

WSO2[8] proposes a reference architecture, which is built based on their experience in developing IoT solutions (Fremantle, 2015). Its architecture consists of five layers, being i) client/external communications, ii) event processing and analytics, iii) an aggregation/bus layer, iv) relevant transport between devices and the system, and v) the devices themselves. In addition, there are two cross-cutting layers, being a device manager and identity and access management. The usefulness of the architecture is proven by mapping it onto a modular, open-source enterprise platform. The architecture is argued to be a useful, deployable, and effective reference architecture since it is based on real-world projects that have been deployed with customers to support IoT capabilities (Fremantle, 2015).

Finally, the Arrowhead[9] architecture is discussed where the dynamic nature of IoT is central. The architecture aims at enabling the creation of local automation clouds, which hold an advantage over global clouds as it allows for real-time performance and security together with simple engineering and scalability. The local clouds consist of three core components, i.e. service registry, authorization, and orchestration (Breivold, 2017). The framework is built on the SOA paradigm to enable the Industrial IoT. Interoperability is achieved through the loose coupling and discovery properties which come with a SOA (Breivold, 2017). Hence, components and functionalities are abstracted into services. In addition, there are no hardwired communications since these services are loosely coupled between service consumers and providers. Late binding enables to get information at any time, also at runtime, by connecting to a specific resource (Delsing, et al., 2017).

## Other architectures

Currently, the majority of IoT platforms use RESTful APIs to access sensor devices and to retrieve, store, update and delete data via the standard HTTP operations such as GET, POST, PUT and DELETE. Data transmission is furnished using the XML, JSON or CSV formats. Acquired data from the physical environment is stored in centralised, mostly cloud-based, databases for processing and accessing. These platforms have differences in some non-core functions, such as business model, data storage policy, data management, visualization, data analysis, event notification and access permission

[8] https://wso2.com/

[9] https://www.arrowhead.eu/

| Layer | Description |
|---|---|
| Device | This layer is integrated with existing hardware (RFID, sensors, actuators, etc.) to acquire data from the physical environment and to steer devices with actuators. |
| Network | This layer supports the required communication technologies and protocols in both directions and for translating data if necessary |
| Service | This layer supports following functionalities: i) receives data from devices, then ii) processes the data, iii) provides the information to connected applications and finally, iv) controls commands sent by devices to enact actuators. |
| Application | This layer provides interaction methods to users and other applications. It uses the middleware layer to gain insights or to control physical actions. |

**Table 1 Description of common layers in IoTA**

control (Wang, Lee, & Murray, 2017). Apart from the refence architectures discussed above, there are more IoTAs proposed in literature. The most basic approach consists of three layers, i.e. an application layer, a network layer and a sensing (or perception) layer (Zheng, et al., 2011). This model was proposed to create specific types of communication but does not cover all of the underlying technologies that transfer data to an IoT platform (Molano, Ródriguez, Bravo, & Santana, 2017). Several other architectures were built based on this three-layered architecture (Wu, Lu, Ling, Sun, & Du, 2010; Atzori, Iera, Morabito, & Nitti, 2012; Aazam, Khan, Alsaffar, & Huh, 2014). Further approaches of layered architectures based on SOAs introduced by Liu et al. (2014), and Li et al. (2013) continue on this three-layered approach and add a fourth layer, i.e. an abstraction layer based on middleware technology therefore called a middleware (or service) layer. A four-layered architecture can be applied in a business environment, i.e. Industry 4.0. Figure 7 shows a generic SOA, proposed by Li et al. (2013), where the interaction among all four layers is illustrated. The functionality of the layers in a four-layered architecture is briefly discussed in Table 1.

## Conclusion

Mostly, the service-oriented approach is applied for IoT systems. SOA-based techniques are widely supported and are classified into the WS-* family of standards, where SOAP is commonly accepted as standard communication protocol. Although, SOAP is not imperative for SOA. These techniques are focused on providing a uniform and structured way of communicating with low power IoT devices. On the other hand, the resource-oriented architecture (ROA) approach stems from the REST paradigm, and could also be classified under service-orientation, but since the capacity of services is not infinite, considering objects as resources can be useful. Regarding the functional components from which the IoTA is built, the level of granularity impacts the level of decomposition of the layers. In this thesis, a trade-off will be made between abstraction which enables interoperability and still preserving sufficient granularity which brings usefulness and clarity.

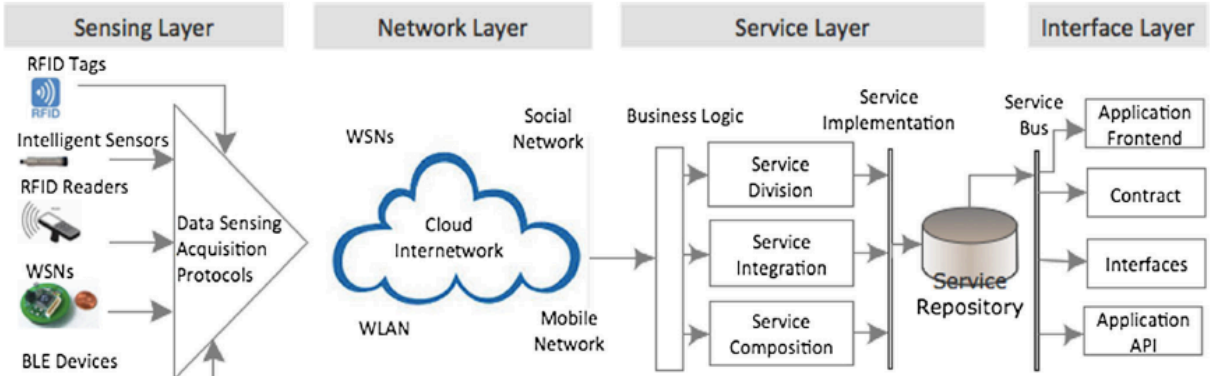


**Figure 7 Generic SOA by Li et al. (2014)**

## 3.3 IoT-aware Business Process Architecture

The aim of enhancing BPs with IoT technology is to eliminate the discrepancy between the digital world of a BPMS and the real world with IoT devices. This can be achieved by enabling an efficient and effective integration of IoT devices and exposing them in such a way that the IoT data can easily be integrated into the BPMS. In order to accomplish this goal, a uniform architecture needs to be in place to facilitate seamless access to these IoT devices (Kashif et al., 2015). On the other hand, BPs need to increase adaptability and be enhanced to exploit the real-time data provided by the IoT devices. This will lead to intelligent cutting-edge systems with substantial self-configuration, self-monitoring, and self-healing properties, which are required to manage the large and fast-growing number of devices (Haller & Magerkurth, 2011). In this thesis, an architecture is called IoT-aware when the IoT data is collected and used for insights and improvements on the BP. An IoT-enhanced BPA, however, is when the system not only interprets the data collected from the IoT objects, but also actively steers the IoT objects to act upon the collected data.

In literature, IoT-aware BPAs are increasingly gaining attraction. For instance, the generic ROA by Dar et al. (2015) enables an end-to-end integration of enterprise-level BPs and tiny IoT devices. A resource-oriented approach was opted for because they are convinced the REST principles will form the foundation of smart environments (sensing, computations and communication can easily be mashed up). This architecture is built on design principles which can manage the features IoT brings. These principles are: i) unified integration, i.e. enabling services over the Web through unique URIs in order to integrate them into enterprise-level BPs; ii) event-based interaction, i.e. supported by a device-push interaction to ensure resource efficiency, which implies that BPs should not actively detect device-level events; iii) a smooth service replacement, which can be managed by retaining the current state of a BP, such that it can resume its execution after the service replacement; and lastly, iv) a decentralised business process (Dar, Taherkordi, Baraki, Eliassen, & Geihs, 2015). This resource-oriented approach is also being adopted by Guinard et al. (2010), where RESTful architectures are considered the most effective solution for IoT-aware systems.

Although there is an increase in this matter, there are still many open challenges to be tackled which will allow for a better integration and align both IoT and BPM technology (Janiesch, et al., 2017). A subset of these challenges is tackled by Schönig et al. (2018) who propose an architecture that addresses i) the placement of IoT sensors in a process-aware way; ii) connection of analytical processes with IoT, i.e. data needs to be up-to-date, current, qualitative, and it needs to be clear where the data stems from and where it has been used; iii) visualization support for managing manually executed tasks, i.e. responsible users must be notified on mobile devices in real time; and iv) improving

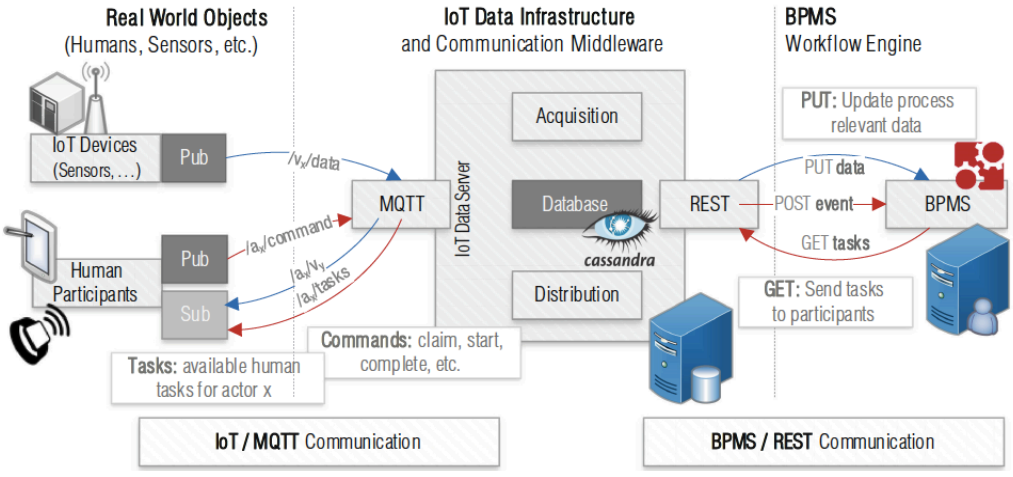


**Figure 8 Integrated communication architecture by Schönig et al. (2018)**

resource utilization optimization, hence provisioning context-specific knowledge that is relevant for the user's particular needs.

The architecture is shown as Figure 8, consisting of three layers: i) IoT objects; ii) IoT infrastructure and communication middleware; and iii) the BPMS. The communication between layer (ii) and the BPMS, is furnished by the Camunda REST API, i.e. HTTP requests GET, PUT and POST. Communication between the IoT objects, the middleware layer (ii) and the BPMS at layer (iii) is established with a Message Queue Telemetry Transport[10] (MQTT) broker. MQTT is a simple and lightweight messaging protocol for machine-to-machine (M2M) connectivity. It works in a pub/sub fashion and is well-suited for IoT networks since it is designed for constrained devices and low-bandwidth, high latency or unreliable networks. Next, collected data is sent from the acquisition application to a NoSQL database, i.e. the Apache Cassandra database. The BPMS remains updated with the latest IoT values because of the distribution application at layer (ii). Evaluating this architecture in practice showed that "the application of the IoT enhanced BPMS leads to less machine stops because users need less time to recognize work that needs to be done." (Schönig, Ackermann, Jablonski, & Ermer, 2018).

### Conclusion

Although the field of IoT-aware BPs is rapidly gaining attention, no mature reference in terms of architectural solutions has been commonly acknowledged. Currently, open challenges have been identified to achieve the benefits which are ready for exploitation. Possible solutions to subsets of these challenges are being proposed, e.g. the discussed integration architecture by Schönig et al. (2018). The current architectural trend is prone to a layered architecture where communication is established based on the REST paradigm. This is argued to be the most effective solution for IoT-aware systems in terms of communication. The IoT is decentralised and heterogenous by nature, and hence a decentralised, event-driven architecture is required to cope with this property. Although this field is not yet mature enough for industry-wide adoption, it yields very promising features such as intelligent cutting-edge systems with substantial self-configuration, self-monitoring, and self-healing properties which are required to manage the large and fast-growing number of devices (Haller & Magerkurth, 2011).

## 4 Common requirements

The Rational Unified Process®[11] has the following definition for *any* requirement: "a requirement describes a condition or capability to which a system must conform; either derived directly from user needs, or stated in a contract, standard, specification, or other formally imposed document." More explicit, an architectural requirement can be defined as any requirement that is "architecturally significant, whether this significance be implicit or explicit" (Eeles, 2005). This section will enumerate the common architectural requirements for a reference architecture to enhance BPs with IoT. In addition, it will provide insights on how these requirements are addressed in current literature. In Section 5, the reference architecture will be built, where the requirements are to be embedded into a reference architecture, whilst providing traceability between the architectural features and the common requirements. A clear overview on how the requirements are addressed by the proposed architecture is provided in Table 2 at the end of Section 5. In the following paragraph all relevant requirements for a reference architecture for IoT-BPs are collected and discussed. In this thesis, four different categories of requirements are identified, i.e. i) IoT specific requirements, e.g. device management ii) BP specific requirements, e.g. process adaptability, iii) cross-cutting requirements which apply to both fields, e.g.

[10] http://mqtt.org/

[11] https://www.ibm.com/

scalability and lastly iv) specific requirements which are not essential in IoTA nor in BPA, but they appear necessary in the combination of both domains, e.g. a decentralized process execution. Although these categories have been identified, it is hard to determine which requirements entirely belong to a certain category since they often overlap, and hence a hard border cannot be drawn.

## 4.1 Interoperability

For architectures it is generally not a question of whether interoperability should be supported, but rather the extent to which the architecture must support for interoperability between applications, systems and components. In case of no physical connections at all, the systems are fully isolated and thus interoperability is inexistent. On the other side of the spectrum, there are universal systems, i.e. enterprise-wide shared systems. Typically, the aimed level of interoperability is in between the isolated and the universal systems, i.e. distributed systems where heterogeneous product exchange is possible, or integrated systems where applications and data can be shared (Tolk & Muguira, 2003).

Interoperability is defined by The Open Group[12] in three-part, i.e. "interoperability is i) the ability to share information and services; ii) the ability of two or more systems or components to exchange and use information; and iii) the ability of systems to provide and receive services from other systems and to use the services so interchanged to enable them to operate effectively together" (The Open Group, 2018). If systems are able to communicate with each other by means of standard data formats and communication protocols, as in (i), it is called syntactic interoperability. Beyond merely communicating systems, semantic interoperability, defined by The Open Group in (ii), means systems can automatically interpret data meaningfully by accurately determining the meaning of the data such that it holds useful information for the end-user. Semantic integration can be achieved by converting entity description into a system-readable representation via customized adaptors to then interpret the correct meaning. However, this imposes high design complexity to the adaptors, since the interpretation patterns of an adaptor are pre-defined. This challenge can be tackled by building ontologies which can describe fine-grained entities and can still be personalized by third-parties (Wang, Lee, & Murray, 2017). In other words, the reference architecture must provide a clear ontology since this allows for third parties to customise entities whilst maintaining the (globally consistent) semantic meaning of the described entities. This way all systems can share information and interpret it the same way.

In the Arrowhead architecture, discussed in Section 3.2, three solutions are proposed to support interoperability in an architecture, namely: i) an interoperability layer, ii) a translator system and iii) a translator as a service (Delsing, et al., 2017). The layer (i) is a contact communication hub which defines a suitable technology and semantics. It translates each service from/to the appropriate technology and semantics. The translator system (ii) can consume a service from one technology and translate the same service to provide the service to another technology. The third option (iii) is very similar to the solution in (ii) but allows the translator to be consumed by other systems as a service. Wang et al. (2017) address the issue of connecting heterogeneous devices in a similar fashion as described above in (i). Interoperability is supported by integrating the distributed proxies in the middleware technology layer, which is independent of platforms and networks.

## 4.2 Security

"Corporate information is not only a competitive asset, but it often contains information of customers, consumers and employees that, in the wrong hands, could create a civil liability and possibly criminal charges" (Rimal, Jukan, Katsaros, & Goeleven, 2011). Hence, building a secure system must not be regarded as a marketing attribute, but rather as an imperative requirement of the system to assure the wellbeing of all stakeholders. Developing a system with resource-constrained devices inherently

[12] https://pubs.opengroup.org/

introduces security risks since the devices itself cannot support adequate security protocols, e.g. proper asymmetric encryption (Fremantle, 2015). In addition, legacy systems are not originally developed to cope with such weak access points and often lack internal security measures. Traditionally, with pure web solutions there is often no available code to attack (i.e. completely server-side implementation), but smart IoT devices contain a small amount of code, which is vulnerable for attacks with the aim to reverse engineer its security (Fremantle, 2015). The architecture must support devices in preventing unauthorized access to the system or the data. In order to provide trust and privacy to all stakeholders of the system, security must be assured system-wide, hence security needs to be embedded within all layers of the architecture. Devices which are powerful enough should be encrypted such that its data is only useful for the intended recipient of the data, and constrained devices should be protected from the system, e.g. with an adequate firewall in between the devices and the system.

A novel approach that is rapidly gaining popularity is the principle of DevSecOps. This approach ensures better-secured information systems by combining the fields of development, security and operations from the early stages of development. This approach ensures a system-wide security *layer* where security is repeatedly considered whilst developing the system. The reference architecture by Fremantle (2015) implies a modular architecture that supports extensions, which copes with the specific demands of certain concrete architectures. Identity and access management are considered essential security requirements that must be met by, a.o. encrypting devices that are powerful enough, an identity model based on tokens (instead of userIDs/paswords) and policy-based and user-managed access control for the system based on XACML (Fremantle, 2015).

A promising technology for safeguarding security is blockchain. However, for networks with constrained devices, the technology was not lightweight nor scalable enough. As a response to this challenge, Dorri et al. (2017) propose a lightweight application of blockchain technology to secure the entire system, adapted to the specific features of constrained devices in IoT networks. Security is assured on three aspects being confidentiality, integrity and availability. Confidentiality is achieved using symmetric encryption, integrity of data is safeguarded by lightweight hashing to detect changes in a transaction's content during transaction. And finally, highly available services to the user are achieved by limiting acceptable transactions by devices and the miner (Dorri, Kanhere, Jurdak, & Guaravaram, 2017). By simulating their blockchain method, the findings concluded that the overheads incurred by their method are low and manageable for low resource IoT devices. They argue that these overheads are worth their weight given the significant security and privacy benefits on offer (Dorri, Kanhere, Jurdak, & Guaravaram, 2017). A blockchain architecture for IoT is also proposed by Sharma et al. (2017). Hence, also blockchain can be considered as a viable solution to secure a system working with constrained devices. Although, a trade-off will need to be made for each specific application. On an architectural basis, this security system requires a smart miner which centralises incoming and outgoing transactions to and from the system. This miner can be integrated into the local Internet gateway or as a stand-alone device (Dorri, Kanhere, Jurdak, & Guaravaram, 2017).

## 4.3 Quality of Service (QoS)

QoS typically relates to networks, i.e. a measure of network service performance (Duan, Chen, & Xing, 2011). Common applications of IoT, like delay-sensitive real-time applications or low data rate monitoring, are very sensitive to the quality of its consumed services. Therefore, QoS is considered as one of the most important architectural requirements for IoT (Yaqoob, et al., 2017). IoTAs are required to provide qualitative services to its users and consumers. However, QoS assurance is cumbersome with constrained devices and unreliable connections. Two measures are to be considered with QoS, i.e. i) prioritization of services and ii) retrieval of required information. By prioritizing services of for example real-time processes, the performance of the service can be guaranteed. In addition, by solely retrieving the required information as response to a query, unnecessary exchange of data is limited to the minimum leading to increased QoS (Yaqoob, et al., 2017). This can be enabled through intermediate data processing and filtering (Dar, Taherkordi, Baraki, Eliassen, & Geihs, 2015).

QoS can be measured based on specific thresholds with clearly defined measurement techniques. For example, the service delay in the application layer, being the lead time between the consumer's send request for a service to receive a response, can be measured in an exact time and hence a threshold can be used to determine the quality. For each layer, several measurements can be set up providing a clear overview of the overall QoS and where the weak links are. More measurements can be found in more detail in (Duan, Chen, & Xing, 2011).

An architecture can support the QoS requirements by having a cross-cutting QoS management layer in place which communicates to QoS brokers, located on each layer to guarantee QoS in each layer. This QoS management layer then manifests itself as a control mechanism for transferring and translating QoS requirements across layers (Duan, Chen, & Xing, 2011).

## 4.4 Device management

IoT services are by nature more volatile than traditional Web services (WSs), because devices in an IoT network may be out of range, low on battery, etc. Hence, broken communication links may lead to service unavailability (Dar, Taherkordi, Baraki, Eliassen, & Geihs, 2015). Adding or removing objects and services to or from a network, should never influence the existing connections of other objects and services. In addition, an unavailable service should be replaced by another (if any) available service offering a similar functionality (Dar, Taherkordi, Baraki, Eliassen, & Geihs, 2015). This cannot be established with WS-* standards as these imply predefined and thus hard-coded mechanisms (Wang, Lee, & Murray, 2017). The reference architecture is expected to support reconfiguration of virtual objects and services at runtime. Wang et al. (2017) propose a service composer which can dynamically coordinate interactions, such as coupling or decoupling connections between objects and services in order to meet specific application needs (Wang, Lee, & Murray, 2017).

## 4.5 Process adaptability

BPs are currently deterministic, centralised and designed top-down. The latter means that process changes originate from the original process model and need to be deployed in the runtime enactment environment (Hens, Snoeck, Poels, & De Backer, 2014). With the big data that is generated by IoT devices and the associated insights, BPs can constantly be upgraded for more efficient solutions. In addition, exceptional situations might occur at runtime, which were not planned at design-time. In order to create this flexibility, ad hoc process changes for adapting running process instances must be allowed, whilst being aware of the situation and the context (Marrella & Mecella, 2017). Marella et al. (2017) introduce Smart Process Management which enables a process to deviate from the execution path at run-time without changing its process model. Changing a BP bottom-up implies that the change will be performed locally at a specific fragment. This change consists of adapting an event rule and propagating this change to other runtime fragments (Hens, Snoeck, Poels, & De Backer, 2014). This flexibility can only be achieved by enabling runtime reconfiguration, model-driven development instead of hard-coded BPs and ensuring the data at hand is reliable. The extent to which processes ought to be flexible, depends on many factors including the nature of the processes, the timeline over which the change would be applicable, etc.

## 4.6 Efficient connectivity and communication

In order to connect the devices with the system and to establish a reliable and efficient communication between the devices and the system, contemporary communication protocols which are better suited for the needs of constrained devices need to be considered. Existing communication protocols such as HTTP are mostly designed for people-to-people communication and hence simple and uniform. This allowed for third parties to easily provide and consume services. However, the price for these useful traits is a high overhead for the devices. This high overhead implies higher memory usage and a higher

usage of energy, which are both limited on IoT devices. In addition, human-readable protocols do not hold much value since most IoT applications are based on object-to-object (O2O) communication. Hence, simple binary protocols are preferred over traditional communication protocols (Wang, Lee, & Murray, 2017; Fremantle, 2015). However, not only binary protocols are required to be offered to the constrained devices, there should still be the possibility for third parties to share data over traditional communication protocols. A key communication requirement for IoT-BPA, is that not only the device should send data to the cloud or the server, but also the reverse (Fremantle, 2015).

In the reference architecture, these communication features are to be integrated into a middleware layer. This layer will act as an intermediary to link different protocols and communication technologies supporting both unconstrained devices by means of WS-* standards, together with constrained devices which are bound to other, more parsimonious communication protocols. The data transmission with these constrained devices can be achieved through XML, JSON or CSV data formats which are considered best practice to describe objects in IoT networks (Wang, Lee, & Murray, 2017; Yaqoob, et al., 2017).

## 4.7 Modular services and loose coupling

Service modularization is virtualizing physical objects or services as primitive middleware components. A reusable and modularized service makes providing and consuming services for third-party users easier. Combining these modularized services into composed services requires a loosely coupled architecture, since this enables the logical separation of virtual objects and services. Hence, primitive virtual objects and services can be independently added, removed or reconfigured (Wang, Lee, & Murray, 2017). Modular services and loose coupling are key SOA principles, which improve the interoperability between devices and the integration of services provided by these devices (Li, Shancang, Xu, & Zhao, 2015; Breivold, 2017). Therefore, the reference architecture will be service-oriented.

## 4.8 High availability

The entire IoT area network must ensure high reliability in order to achieve high availability of data communications, considering the vitality for the actual execution of operations (ITU-T, 2016). The increasing dependence on software-intensive systems drives the need for dependable, robust, continuously available systems. Runtime reconfiguration is a critical step in order to achieve this goal.

High availability is required for an IoT-BPA because often it is applied where real-time interaction between BPs and IoT devices in both directions is critical. For example, a robotised surgery where a surgeon is in another physical location than the patient. In this case the entire system is required to be highly available, approaching .99999 uptime which translates into a weekly downtime of 0.1 seconds. The system cannot afford to be down for even the smallest period, because this imposes life-threatening situations. In case of a network failure, the applications running on those networks must be able to continue operating by using any resources still functioning (Byers, 2017). Important steps to achieve high availability are argued to be i) the provisioning of priority redundancies, ii) the identification of failures, and iii) the invocation of alleviation mechanisms (Sharma, Chen, & Park, 2017).

## 4.9 Event-based

An essential benefit that IoT brings when integrated with BPs, is that the processes are better aligned with what happens in the real world. Inherently, processes are based on events which are either detected directly or by real-time analysis of sensor data (Hens, Snoeck, Poels, & De Backer, 2014). Therefore, services deployed on IoT devices must support event-based asynchronous interaction with BPs. To limit battery consumption of IoT devices, they should interact with the BPs in a device-push

fashion. This precludes BPs to steadily poll for device-level events, meaning lavish use of energy. Instead, the BP is only notified in case of an event (Dar, Taherkordi, Baraki, Eliassen, & Geihs, 2015).

In addition, for an event-based architecture to fully support the integration of IoT with BPs current BP modelling tools need to be extended with capabilities to handle e.g. unreliable data, unreliable resources and highly distributed processes which are specific IoT-BPs (Haller & Magerkurth, 2011). Event-based communication is argued to facilitate flexibility for the execution environment allowing for better control over load balancing and replication needs (Hens, Snoeck, Poels, & De Backer, 2014).

## 4.10 Scalability

With the proliferating number of connected IoT devices and substantial differences in the interaction patterns and behaviours of BPs, supporting elastic scalability of a developed information system is an essential requirement for a reference architecture. Scalability can be defined as “managing the connectivity among a voluminous amount of network devices without causing any performance degradation issues” (Yaqoob, et al., 2017). In other words, adding more capacity to a system should bring a (quasi-) linear increase in performance. Prior systems were mostly created and developed for the current needs, without reckoning the possible future needs of the system. Therefore, it is crucial to keep scalability in mind from early stages in developing the architecture, so that the developed system can be scaled-up, scaled-down and new functionalities can be extended, or obsolete functionalities can be decommissioned with ease. Scalability can be achieved horizontally, vertically, or both. Vertical scalability means increasing the resource capacity of a single node, e.g. increasing the processing capacity of a server. This is considered out of scope since the reference architecture ought to be independent of infrastructure. Horizontal scalability is typically addressed by load balancing process instances across multiple nodes. Load balancing is the mechanism of self-regulating the workloads across available resources. In other words, the load balancer provides ways by which application instances can be provisioned and de-provisioned to the node (Rimal, Jukan, Katsaros, & Goeleven, 2011). In this process, the service components are under constant supervision, and when a failover occurs, i.e. the component becomes non-responsive, the system instantly gets notified and sends the instances to another component (Rimal, Jukan, Katsaros, & Goeleven, 2011; Li, Muthusamy, & Jacobsen, 2010). As discussed in Section 3.1, Camunda provides a BPM platform which enables clustering of execution engines. By clustering execution engines, the application instances can be distributed among several nodes, i.e. replicated execution engines. Camunda defines a cluster as “a set of network nodes that all run the Camunda BPM platform against the same database”.
Another way to provide scalability is, instead of distributing the instances over different clusters, fragmenting the whole BP and executing these fragments on different (possibly unique) engines or servers (Hens, Snoeck, Poels, & De Backer, 2014). BP model fragmentation means splitting a process model into logically different, smaller model fragments. Hens et al. argue that finding the event rule for each process model fragment is the most important part in the transformation towards a partitioned and distributed BP. Other design choices influencing scalability include, but are not limited to prioritization of processing, reducing computational complexity, distributing processing over time, minimising the use of shared resources, etc.

## 4.11 Decentralised process execution

Another shortcoming of standard web protocols for IoT applications, in addition to the high overhead discussed earlier, is that accessing sensor data indirectly in remote web servers increases access

latency. Also, a centralized (orchestration) architecture imposes security, privacy, trust, responsibility and data ownership issues (Vermesan, et al., 2011); ergo, BPs should be organised for a decentralised (choreography) execution. Decentralised execution enables the domain expert to design and simultaneously define the communication between the different BPs (Dar, Taherkordi, Baraki, Eliassen, & Geihs, 2015). Hens et al. (2014) argue that the global process flow needs to be transformed into a distributed event-based process. This would enable the event-based architecture, as discussed in R9, to exploit the loose coupling benefits it imposes.

### 4.12 Actuation of devices

Lastly, there is the architectural requirement to enable the system to steer IoT devices for physical actuation. Operational processes happen in real-time, and thus BPs are expected to be executed in real-time as well. Thus, devices need to be able to analyse and act on real-time data. Powerful devices can utilise engines for event processing and action. However, most devices will only possess simple embedded logic to execute tasks (Fremantle, 2015). An important aspect in actuating devices, is online conformance checking, i.e. analysing an existing process model and comparing it with an event log of the same process (Janiesch, et al., 2017). This improves the alignment of BPs with the actual process in the real world.

## 5 Proposed reference architecture

### 5.1 Building the architecture

A survey by Breivold (2017) identified common driving forces for the development of IoT reference architectures. These include i) the increased complexity and size of systems with the proliferating number of IoT devices, ii) the need for shorter time-to-market and rapid development, iii) collaborative solutions that demand integrated and coordinated information systems, iv) need for interoperability and compliance among devices and systems, and finally v) increased focus on optimising assets and operations in and across different plants (Breivold, 2017). These driving forces are in line with (Cloutier, et al., 2009). Building a reference architecture holds several advantages. Because of the rapid evolvement of new technologies and hence application requirements, a concrete architecture would only be valid for a limited amount of time. Therefore, developing a reference architecture holds more value over time since it can be adapted to the current needs of the application field, reducing time and cost to suit to the changes. Also, a reference architecture provides interoperability by abstracting from the specificities which come with heterogenic platforms, devices, etc. Moreover, a reference architecture enables third-parties to provide and consume services on public platforms in a simple way (Wang, Lee, & Murray, 2017). At the end of this section, Table 2 provides traceability by clearly depicting which architectural features address which requirements. For the reference architecture to be deployed in various environments, not all layers are required and can be omitted depending on the needs of the concrete architecture. For example, if the business process does not require to physically impact the environment, actuators do not have to be included. Additionally, if the system only consists of unconstrained devices in any way, the devices can directly communicate with the middleware layer, and there is no need for a network layer.

### 5.1.1 SOA principle

Building an architecture in a service-oriented fashion brings several essential features, including extensibility, scalability, modularity, and interoperability among heterogeneous devices. IBM[13] defines a SOA as “an application architecture within which all functions are defined as independent services with well-defined invokable interfaces, which can be called in defined sequences to form business processes” (Channabasavaiah, Holley, & Tuggle, 2003).

By developing an architecture according to the SOA paradigm, already many of the collected architectural requirements are addressed (Breivold, 2017). Building a SOA yields several advantages, which include i) the reduction in development time and cost, i.e. services are built using existing services to form composite applications; ii) lower maintenance cost, i.e. reusing services decreases the amount of services and internal complexity of enterprise services; iii) lower integration cost because standardised services enable various applications to connect easily; and iv) reduce risk since fewer reusable services provide a better overview and hence better control of the overall compliance risk of an enterprise (Xiao, Guo, Xu, & Gong, 2014; Karande, Karande, & Meshram, 2011).

## 5.2 A reference architecture for executing IoT-BPs

### 5.2.1 Device layer

A device is a hardware component that connects wirelessly to a network and is capable of transmitting data. A device is connected to a sensor or an actuator, or these components are integrated into the device. A sensor senses the environment and gathers information about the environment. An actuator is able to automate a process activity by acting upon the data it receives. Actuators are key enablers towards full automation of processes, i.e. robotization. Examples of IoT devices can range from smartphones, smart heartrate monitors, connected traffic lights, a thermostat in a reefer container to even autonomous vehicles. Devices in IoT networks are heterogenic by nature because the characteristics of different devices vary strongly and hence the requirements the heterogeneity imposes on for example connectivity and communication vary as well. The key traits to distinguish devices are its capabilities, i.e. battery capacity, available processing power, accessible networks, etc. These factors categorise a device to be either constrained or unconstrained. In line with Guth et al. (2016), another distinction can be made between devices being either self-contained or connected to another system. Connected devices communicate with the middleware layer, while self-contained devices create a black box of functionality to the system. An example of a self-contained device could be a thermostat which processes its own sensed data and then manages the cooling or heating of a reefer container to assure the temperature remains within a certain threshold. An active interaction with the IoT environment can be achieved with BPMN service tasks provided by the HTTP

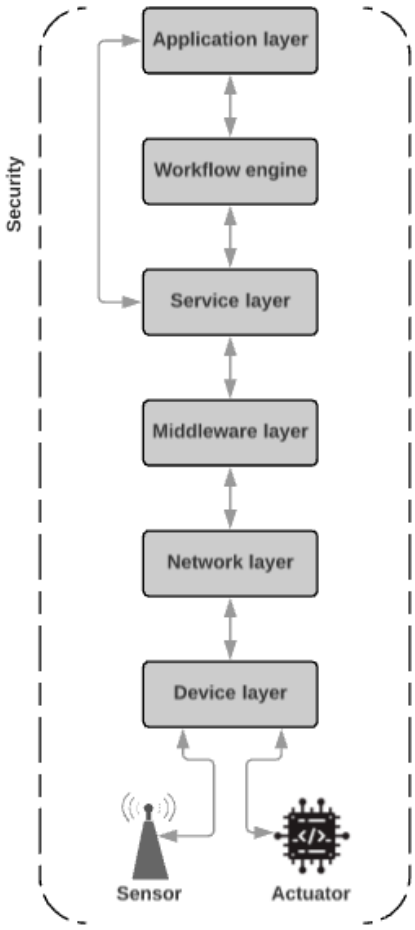


**Figure 9 Reference architecture for IoT-enhanced BPs**

[13] https://www.ibm.com/

connectors of a BPMN execution engine. These connectors establish either a direct communication over protocols such as HTTP or REST, or an indirect communication over a platform, e.g. Node-RED[14]. In case a robot (i.e. an IoT device) would receive a service task, the system would send a HTTP-command to the IoT platform, where the corresponding URL parameter or POST-body is issued to the robot in a supported protocol. In this way a fully bidirectional communication between IoT devices and process management systems can be realised, as shown in (Schönig, Jablonski, & Ermer, 2019).

### 5.2.2 Network layer

The goal of the network layer is to furnish a connection between (constrained) devices and the system, i.e. the middleware layer. This can be achieved either directly in case of IP-enabled devices, or over a gateway in case of constrained devices. Hence, a gateway is required to compensate the limitations specific to constrained devices by providing technologies and functionalities such as translating between different protocols and forwarding the intended communication to or from the system. Constrained devices connect over a gateway, provided in the network layer. Unconstrained devices can be IP-enabled and thus connect directly to the middleware layer over the Internet. The Azure platform by Microsoft acknowledges the need for two types of gateways in order to limit de data transmitted over the network and to improve latency. One gateway is provided at the edge of the network, which is capable of data filtering, aggregation and buffering of data, protocol translation, event rules processing and device provisioning. The second gateway is a cloud gateway which allows for remote devices to communicate with the system (Microsoft, 2018). By intermediate data processing and filtering on a gateway at the edge of the network, the required information to a query can be determined more easily and limits the network load, ensuring better QoS. In addition, an adequate QoS-level also ensures high availability of the system. Note that SOAP is not imperative for SOA. Only for providing services to third parties over the Internet, WSs are considered the best solution. Unconstrained devices can use full HTTP client libraries to properly implement the whole protocol, while constrained devices could only partially support the protocol, e.g. only enough code to POST or GET a resource (Fremantle, 2015). Hence, systems with constrained devices achieve a better performance with binary, rather than text-based, protocols such as MQTT or CoAP since they generate much smaller overhead and they lower the data stream (Wang, Lee, & Murray, 2017).

### 5.2.3 Middleware layer

The middleware layer is mainly responsible for receiving and processing data, in addition to facilitating the transmission of data between IoT objects and the system, i.e. the workflow engine and the connected applications. Communication patterns can be designed as pull, push or pub/sub, to fulfil different application needs without concerning the underlying devices or networks (Wang, Lee, & Murray, 2017). The required bidirectional transmission of data between IoT objects and the system can be established by working with a brokered communication model, where clients can connect an outbound connection to the broker, regardless if the device is acting as a publisher or a subscriber. In addition, a brokered communication model is also required to support an event-driven architecture (Li, Muthusamy, & Jacobsen, 2010). One of the benefits of an MQTT specification is that it enables this type of communication (Fremantle, 2015).

In line with the integration architecture proposed by Schönig et al. (2018), the middleware is responsible for three main tasks, i.e. i) acquiring the IoT data, including the meta information to ensure clear data provenance, and then ii) storing the acquired data in a well-suited database (e.g. a NoSQL database such as Apache Casandra[15]). Subsequently, iii) a distribution application in the layer provides the service layer with all the necessary functionalities to accommodate the workflow engine with the latest

[14] https://nodered.org/

[15] http://cassandra.apache.org/

IoT values. Further, real-time information can be provided to the service layer such that the application layer has an easy access to the latest insights on the acquired data. Typically, the middleware layer is accessible through APIs, e.g. HTTP-based REST APIs (Guth, Breitenbücher, Falkenthal, Leymann, & Reinfurt, 2016).

Another functionality of this layer is hiding the heterogeneity of hardware and software to provide uniformity among objects, hence improving interoperability. All communication must be translated into the correct communication protocol such that the intended message reaches its recipient, and no ambiguity is formed. Subsequently, it allows the system to make a connection between the IoT platform and the (business) applications, but also with customers and partners. An important element for this layer is that the hardware and software should be reusable.

### 5.2.4 Service layer

This layer provides the necessary functionalities to connected applications on one hand and sends data to the workflow engine on the other hand. To properly facilitate a SOA, the service functionalities are considered in a distinct layer, whereas other architectures may embed these functionalities into the middleware layer. The service layer facilitates a seamless integration of the system with the applications and the BPs, in combination with the middleware layer. Atomic objects and service components can be combined to provide a composed service. By then reusing primitive services, this lowers the development cost and improves the ease of deployment. The service layer ought to identify common requirements from applications and the workflow engine, and provide the necessary functionalities to meet these requirements. These functionalities are provided over APIs and protocols. The principle functionalities of the service layer are: i) service discovery, ii) service composition, iii) trustworthiness management, iv) service APIs, and v) service replacement management (Li, Xu, & Zhao, 2015).

### 5.2.5 Workflow engine

The middleware layer generates information which the service layer offers to the workflow engine in order to execute the process. Communication between the middleware layer and the workflow engine typically happens by means of REST APIs, i.e. HTTP requests PUT, POST and GET. Based on the available information stemming from IoT data, the engine can calculate available activities.

The workflow engine is capable of executing and adapting the processes, which can be regarded as two layers in the workflow engine. The execution implies managing and coordinating the process enactment, i.e. in a distributed way. BPMN is commonly accepted as modelling language for BPs. BPMN provides an XML-file which can be interpreted for execution by the engine. The engine is also responsible for identifying discrepancies or changes, which it notifies to the adaptation layer. The workflow engine sends its commands to the IoT devices over a specific platform to translate the communication from the engine into device-readable protocols, e.g. over the NodeRED platform. The automatic communication between the execution engine and the IoT device, i.e. the actuator, enables the system to fully automated processes. For such processes, down-time of the system is critical, hence the workflow engine must be capable of handling high workload efficiently and assure high availability by for instance load balancing mechanisms as discussed in Section 4.11. In addition, the workflow engine plays an important role in designing an architecture which is scalable to meet the growing process instances to be executed (e.g. by clustering the workflow engine).

Compared to the WfMC reference architecture, the workflow engine is also a stand-alone component of the architecture. However, the gravity of this component in the proposed architecture is less dominant since it is not only focused on the execution of processes, but also on the transmission of data from the sensors through the system, to the actuators.

### 5.2.6 Application layer

The application layer provides interaction methods to users and other applications. The service layer provides services which enable the applications to gain insights or to control physical actions. An application can also be another middleware layer, in order to ensure extensibility of the system and to integrate multiple systems. Through this layer the IoT data is linked with the IT and operational technology (OT) applications such as ERP, CRM, EMS, etc. Therefore, it is a crucial layer to link the business with the IoT platform.

| Requirement | | Architectural feature | Also considered by |
|---|---|---|---|
| **R1** | Interoperability | Service layer / middleware layer | (Wang, Lee, & Murray, 2017); (Xu, He, & Li, 2014); (Grefen & de Vries, 1998); (Xiao, Guo, Xu, & Gong, 2014); (Yaqoob, et al., 2017); (Dar, Taherkordi, Baraki, Eliassen, & Geihs, 2015); (Fremantle, 2015); (Li, Xu, & Zhao, 2015); |
| **R2** | Security | Cross-layer security | (Yaqoob, et al., 2017); (Byers, 2017); (Weyrich & Ebert, 2016); (Fremantle, 2015) |
| **R3** | QoS | Network layer | (Yaqoob, et al., 2017); (Li, Xu, & Zhao, 2015); |
| **R4** | Device management | Device layer | (Weyrich & Ebert, 2016); (Dar, Taherkordi, Baraki, Eliassen, & Geihs, 2015); (Fremantle, 2015); |
| **R5** | Process adaptability | Workflow engine / middleware layer | (Yaqoob, et al., 2017); |
| **R6** | Efficient connectivity & communication | Middleware layer / network layer | (Fremantle, 2015); (Weyrich & Ebert, 2016); |
| **R7** | Modular services and loose coupling | Service layer | (Wang, Lee, & Murray, 2017); (Byers, 2017); (Li, Xu, & Zhao, 2015); |
| **R8** | High availability | Cross-layer | (Fremantle, 2015); |
| **R9** | Event-based | Workflow engine | (Dar, Taherkordi, Baraki, Eliassen, & Geihs, 2015); |
| **R10** | Scalability | Cross-layer | (Fremantle, 2015); (Li, Xu, & Zhao, 2015); |
| **R11** | Decentralised process execution | Workflow | (Dar, Taherkordi, Baraki, Eliassen, & Geihs, 2015); |
| **R12** | Actuation of devices | Actuators / workflow engine | (Schönig, Jablonski, & Ermer, 2019); |

**Table 2 Overview table on selected requirements**

# 6 Evaluation and discussion

The proposed architecture is evaluated by comparing its capabilities with other architectures, which have been discussed in Section 3. In Table 3, the identified requirements for a reference IoT-BPA are listed. For each requirement it is shown whether it is a capability of the architecture (✓), or not (✗). Requirements which were identified in other works, but not considered for this architecture can be found in Appendix A including a brief motivation for not considering the requirement.

By comparing the capabilities, i.e. requirements which are met by the architecture, some insights can be gained. Interoperability is commonly acknowledged as an architectural requirement. In contrast, there appears to be a lack of attention for security in (reference) architectures. Nevertheless, security is considered as an important architectural requirement in this thesis, which is in line with the WSO2 architecture by Fremantle (2015). For BPAs, the S3 appears to be the most comprehensive reference architecture. However, in Section 3.1, this was the least popular BPA to serve as a basis for concrete architectures. In contrast to IoTAs, the BPAs are rather simple in terms of architectural requirements. For instance, efficient connectivity and communication are mostly considered for IoTA or IoT-BPA, ditto for device management. Many of the requirements appear to be necessary for an architecture to cope with IoT, whereas the requirements for BPAs are rather limited.

Notably, only two architectures have the capability to steer actuators, i.e. i) the reference IoTA by Guth et al. (2016) and ii) the IoT-BPA by Schönig et al. (2019). The architecture proposed in this thesis, however, differs from (i) by including the BPs and facilitating the full integration of both domains, not only focusing on IoT. Moreover, it differs from (ii) by being a reference architecture and not a concrete architecture. The IoT-BPA by Dar et al. (2015) is a comprehensive and useful architecture for IoT-BPs, however it lacks the ability to steer actuators. Therefore, the proposed reference architecture is believed to be useful, serving as a basis for concrete architectures or future research.

However, the outcome and findings from this evaluation should be regarded with due consideration of the limitations of this thesis. By reason of time constraints, the evaluation was limited to a comparison with 9 other architectures. Hence, the reference architecture is presumed to be more useful than current alternatives, but this cannot be generalised. By performing a more extensive comparison of architectures, in addition to a (systematic) literature review, the obtained findings can be nuanced. In order to assure the quality of the reference architecture, it should be evaluated more thoroughly, in addition to the comparison, e.g. by means of a case study. In such case study, various scenarios are mapped onto the reference architecture and the reference architecture can prove its value by identification of its capabilities and shortcomings. This method was executed by Sharpe et al. (2019) in an Industry 4.0 reference architecture. Still, I believe the selected architectures to which the proposed architecture is compared are prominent and representative for the majority of (reference) architectures and thus the findings are presumed to be useful.

| | BPA | | | | IoTA | | | IoT-BP | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | WfMC | Mercurius | S3 | Kumaran et al. (2008) | Guth et al. (2016) | WSO2 | Mircosoft Azure | Schönig et al. (2019) | Dar et al. (2015) | **A Reference Architecture for IoT-enhanced Business Processes** |
| **Interoperability** | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ |
| **Security** | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✓ |
| **QoS** | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✓ |
| **Device management** | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ |
| **Process adaptability** | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ |
| **Efficient connectivity & communication** | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Modular services and loose coupling** | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ |
| **High availability** | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ |
| **Event-based** | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Scalability** | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Decentralised process execution** | ✓ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ | ✓ | ✓ |
| **Actuation of devices** | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ | ✓ |

**Table 3 Comparison of architectures; (✓) enabled by architecture, (✗) not enabled by architecture**

# 7 Conclusion and future work

With the steadily maturing IoT platforms and the demand for businesses to enhance its operations with this technology, a reference architecture is becoming increasingly relevant for the global implementation of IoT. However, there still is a lack in (reference) architectures to facilitate the execution of IoT-BPs. The objective of this thesis was to narrow down the architectural gap for the amalgamation of IoT devices and BPs.

First, the state of the art was reviewed in all three domains (i.e. IoTA, BPA and IoT-BPA). This allows researchers to quickly gain insight into the matter and serves as a basis for more elaborate research. Next, the relevant requirements for a reference architecture for IoT-BPs were identified and discussed, also showing how these are addressed in current literature. Subsequently, merging the listed requirements lead to an architecture which, in contrast to all other examined architectures, is capable to not only integrate the acquired IoT data into BPs, but also enhance the system to physically impact the environment by steering actuators. To the best of my knowledge, there is no existing reference architecture that utilises the IoT capabilities as extensively (i.e. both sensing and actuating) as the proposed reference architecture in this thesis. Respective to the evaluation in Section 6, the proposed reference architecture demonstrated its added value in comparison to the discussed alternatives.

With the introduction of this architecture, it is intended to provide basis for future research or concrete architectures. The proposed architecture was focussed on BP execution. Hence, it would be interesting to see how the architecture should be extended to support other phases in BPM, for instance conformance checking. In addition, ontology is not discussed in detail in this thesis, since this is reckoned to be examined more thoroughly in order to be standardised. Guth et al. (2016) argue that there is no common terminology on IoT keywords, such as *things* or *devices* and that a reference architecture must provide clarity in this matter. As also argued by Xu et al. (2014), will the successful global use of IoT strongly depend on standardisation, as this enables interoperability, compatibility, reliability, and effective operations.

# 8 References


Aazam, M., Khan, I., Alsaffar, A. A., & Huh, E.-N. (2014). Cloud of Things: Integrating Internet of Things and Cloud Computing and the Issues Involved. *11th International Bhurban Conference on Applied Sciences & Technology (IBCAST)* (pp. 414-419). Islamabad: IEEE.

Arsanjani, A., Zhang, L.-J., Ellis, M., Allam, A., & Channabasavaiah, K. (2007). S3: A Service- Oriented Reference Architecture. *IEEE Computer Society*, 10-17.

Atzori, L., Iera, A., Morabito, G., & Nitti, M. (2012). The Social Internet of Things (SIoT) – When social networks meet the Internet of Things: Concept, architecture and network characterization. *Computer Networks*, 3594–3608.

Barros, O. (2007). Business process patterns and frameworks. *Emerald Insight Business Process Management Journal, Vol. 13 Iss 1*, 47-69.

Breivold, H. P. (2017). A Survey and Analysis of Reference Architectures for the Internet-of-things. *The Twelfth International Conference on Software Engineering Advances* (pp. 132-138). Västerås, Sweden: ICSEA.

Breivold, H. P. (2017). Internet-of-Things and Cloud Computing for Smart Industry: A Systematic Mapping Studdy. *5th International Conference on Enterprise Systems* (pp. 299-304). Västerås, Sweden: ABB Corporate Research.

Byers, C. C. (2017). Architectural Imperatives for Fog Computing: Use Cases, Requirements, and Architectural Techniques for Fog-Enabled IoT Networks. *IEEE Communications Magazine, 55*(8), 14 - 20.

Channabasavaiah, K., Holley, K., & Tuggle, E. (2003). Migrating to a service-oriented architecture. *IBM DeveloperWorks, 16,* , 727-728.

Cloutier, R., Muller, G., Verma, D., Nilchiani, R., Hole, E., & Bone, M. (2009). The Concept of Reference Architectures. *Systems Engineering, 13*(1), pp. 14-27.

Dar, K., Taherkordi, A., Baraki, H., Eliassen, F., & Geihs, K. (2015). A resource oriented integration architecture for the Internet of Things: a business process perspective. *Pervasive and Mobile Computing*, pp. 145-159.

Delsing, J., Varga, P., Ferreira, L., Albano, M., Pereira, P. P., Eliasson, J., . . . Derhamy, H. (2017). The Arrowhead Framework architecture. In J. Delsing, *IoT automation Arrowhead framework* (pp. 43-86). CPC Press.

Dijkman, R., Vanderfeesten, I., & Reijers, H. A. (2011). *Designing a Business Process Architecture: An Overview of Approaches and their Use.* Eindhoven, the Netherlands: Eindhoven University of Technology.

Dorri, A., Kanhere, S. S., Jurdak, R., & Guaravaram, P. (2017). Blockchain for IoT Security and Privacy: The Case Study of a Smart Home. *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 1-6.

Duan, R., Chen, X., & Xing, T. (2011). A QoS Architecture for IOT. *IEEE International Conferences on Internet of Things, and Cyber, Physical and Social Computing*, 717-720.

Dumas, M., La Rosa, M., Mendling, J., & Reijers, H. (2013). *Fundamentals of Business Process Management Vol.1, p. 2.* Heidelberg: Springer.

Eeles, P. (2005). *Capturing Architectural Requirements*. Retrieved from https://www.ibm.com/

Fremantle, P. (2015). *A Reference Architecture For The Internet Of Things.* DOI: 10.13140/RG.2.2.20158.89922

Glombitza, N., Ebers, S., Pfisterer, D., & Fischer, S. (2011). Using BPEL to Realize Business Processes for an Internet of Things. *Ad-hoc, Mobile, and Wireless Netowrks* (pp. 294-307). Berlin, Heidelberg: Springer.

Green, S., & Ould, M. (2005). A framework for classifying and evaluating process architecture methods. *John Wiley & Sons*.

Grefen, P., & de Vries, R. R. (1998). A reference architecture for workflow management systems. *Data & Knowledge Engineering 27*, pp. 31-57.

Guinard, D., Trifa, V., & Wilde, E. (2010). A resource oriented architecture for the Web of Things. *2010 Internet of Things (IOT)* (pp. 1-8). Tokyo: IEEE.

Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., & Reinfurt, L. (2016). Comparison of IoT Platform Architectures: A Field Study based on a Reference Architecture. In *2016 Cloudification of the Internet of Things (CIoT)* (pp. 1-6). Stüttgart: IEEE.

Haller, S., & Magerkurth, C. (2011). The real-time enterprise: iot-enabled business processes. *IETF IAB Workshop on Interconnecting Smart Objects with the Internet*, pp. 1-3.

Helmut, P. (2006). *Service-oriented architecture (soa) vs. component based architecture.* Vienna: Vienna University of Technology.

Hens, P., Snoeck, M., Poels, G., & De Backer, M. (2014). Process fragmentation, distribution and execution using an event-based interaction scheme. *The Journal of Systems and Software*, 170-192.

Hollingsworth, D. (1994). *Workflow Management Coalition The Workflow Reference Model.* The Workflow Management Coalition.

ITU-T. (2016). Internet of things and smart cities and communities – Requirements and use cases. *Series Y: global information infrastructure, internet protocol aspects and next-generation networks, internet of things and smart cities*, 1-10.

Janiesch, C., Koschmider, A., Mecella, M., Weber, B., Burattin, A., Ciccio, C. D., . . . Kannen, U. (2017). The Internet-of-Things Meets Business Process Management: Mutual Benefits and Challenges. *arxiv.org/abs/1709.03628*, 9.

Josey, A., & Franken, H. (2012). *ArchiMate 2.0: an introduction.* The Open Group.

Karande, A., Karande, M., & Meshram, B. (2011, March). Choreography and Orchestration using Business Process Execution Language for SOA with Web Services. *International Journal of Computer Science Issues, Vol. 8, Issue 2,*, pp. 224-232.

Krco, Srdjan, Pokric, B., & Carrez, F. (2014). Designing IoT architecture(s): a european perspective. *2014 IEEE World Forum on Internet of Things* (pp. 79-84). Seoul: IEEE.

Kumaran, S., Liu, R., Dhoolia, P., Heath, T., Nandi, P., & Pinel, F. (2008). A RESTful Architecture for Service-Oriented Business Process Execution. *IEEE International Conference on e-Business Engineering* (pp. 197-204). Xi'an: IEEE Computer Society.

Li, G., Muthusamy, V., & Jacobsen, H.-A. (2010). A distributed service-oriented architecture for business process execution. *ACM Trans. Web, 4, 1, Article 2*, 33.

Li, Q., Wang, Z.-y., Li, W.-h., Li, J., Wang, C., & Du, R.-y. (2013). Applications integration in a hybrid cloud computing environment: modelling and platform. *Enterprise Information Systems 7:3*, 237-271.

Li, S., Xu, L. D., & Zhao, S. (2015). The internet of things: a survey. *Information Systems Frontiers, 2*, 243-259.

Li, S., Zhang, Y., Raychaudhuri, D., Ravindran, R., Zheng, Q., Dong, L., & Wang, G. (2015). IoT Middleware Architecture over Information-Centric Network. *IEEE*.

Li, Shancang, Xu, L. D., & Zhao, S. (2015). The internet of things: a survey. *Information Systems Frontiers, 2*, 243-259.

Liu, C. H., Yang, B., & Liu, T. (2014). Efficient naming, addressing and profile services in Internet-of-Things sensory environments. *Ad Hoc Networks 18*, 85-101.

Marrella, A., & Mecella, M. (2017). Adaptive Process Management in Cyber-Physical Domains. *Intelligent Systems Reference Library 123*, 15-48.

Microsoft. (2018, September 26). *Microsoft Azure IoT Reference Architecture.* Retrieved from azure.microsoft.com: https://azure.microsoft.com/nl-nl/blog/azure-iot-reference-architecture-2-1-release/

Milojevic, M. (2017). *Digital Industrial Transformation with the Internet of Things.* CXP Group.

Molano, Ródriguez, J. I., Bravo, L. E., & Santana, E. R. (2017). Data architecture for the internet of things and industry 4.0. *International Conference on Data Mining and Big Data*, 283-293.

Pourmirza, S., Peters, S., Dijkman, R., & Grefen, P. (2017). A systematic literature review on the architecture of business process management systems. *Information Systems 66*, 43-58.

Rimal, B. P., Jukan, A., Katsaros, D., & Goeleven, Y. (2011). Architectural Requirements for Cloud Computing Systems: An Enterprise Cloud Approach. *Grid Computing*, 3-26.

Schönig, S., Ackermann, L., Jablonski, S., & Ermer, A. (2018). An Integrated Architecture for IoT-Aware Business Process Execution. *Springer Nature 2018*, 19–34.

Schönig, S., Jablonski, S., & Ermer, A. (2019). IoT-basiertes Prozessmanagement. *Mobile Benutzerung in er digitalen Fabrik*.

Serral, E., Valderas, P., & Pelechano, V. (2013). Addressing the evolution of automated user behaviour patterns by runtime model interpretation. *Springer*, 1387–1420.

Sharma, P. K., Chen, M.-Y., & Park, J. H. (2017). A Software Defined Fog Node Based Distributed Blockchain Cloud Architecture for IoT. *Special section on intelligent systems for the Internet of Things, 6*, pp. 115 - 124.

Sharpe, R., Lopik, K. v., Neal, A., Goodall, P., Conway, P. P., & West, A. A. (2019). An industrial evaluation of an Industry 4.0 reference architecture demonstrating the need for the inclusion of security and human components. *Computers in Industry*, 37–44.

Shirer, M., & Torchia, M. (2017). *IDC Media Center*. Retrieved from IDC.com: https://www.idc.com/getdoc.jsp?containerId=prUS43295217

Swenson, K. D., & Shapiro, R. M. (2008). *BPM in Practice: A Primer for BPM & Workflow Standards.*

The Open Group. (2018). *pubs.opengroup.org*. Retrieved from https://pubs.opengroup.org/architecture/togaf9-doc/arch/chap03.html

Tolk, A., & Muguira, J. A. (2003). The levels of conceptual interoperability model. *Proceedings of the 2003 fall simulation interoperability workshop* (pp. 1-11). Norfolk: Citeseer.

Torkaman, A., & Seyyedi, M. (2016). Analyzing IoT Reference Architecture Models. *International Journal of Computer Science and Software Engineering (IJCSSE)*, pp. 154-160.

van der Aalst, W. (2011). Process Discovery: An Introduction. In *Process Mining* (pp. 125-156). Berlin Heidelberg: Springer.

van Kranenburg, R. (2007). The Internet of Things: A critique of Ambient Technology and the All-Seeing Network of RFID. Amsterdam, The Netherlands: Institute of Network Cultures.

Vermesan, O., Friess, P., Guillemin, P., Gusmeroli, S., Sundmaeker, H., Bassi, A., . . . Doody, P. (2011). Internet of things strategic roadmap. *Internet of things - global technological societal trends, vol 1, 1*, 9-52.

Wang, W., Lee, K., & Murray, D. (2017). A global generic architecture for the future Internet of Things. *Service oriented computing and applications*, 329-344.

Wang, W., Lee, K., & Murray, D. (2017). A global generic architecture for the future Internet of Things. *Service Oriented Computing and Applications 11, 3*, 329-334.

Watson, M. (2014). Better, faster stronger: Smart manufacturing gains momentum. *IHS Quarterly/Technology*.

Weske, M. (2012). Business Process Management Architectures. In *Business Process Management* (pp. 333-371). Berlin: Springer.

Weyrich, M., & Ebert, C. (2016). Reference Architectures for the Internet of Things. *IEEE COMPUTER SOCIETY*, 112-116.

Wu, M., Lu, T.-l., Ling, F.-Y., Sun, l., & Du, H.-Y. (2010). Research on the architecture of Internet of things. *3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)* (pp. V5-484 - V5-487). IEEE.

Xiao, G., Guo, J., Xu, L., & Gong, Z. (2014). User interoperability with heterogeneous IoT devices through transformation. *Transactions on Industrial Informatics*, 1486 - 1496.

Xu, L. D., He, W., & Li, S. (2014). Internet of things in industries: A survey. *IEEE Transactions on industrial informatics 10, no. 4*, 2233-2243.

Yaqoob, I., Ahmed, E., Abaker, I., Hashem, T., Ibrahim, A., Ahmed, A., . . . Guizani, M. (2017). Internet of Things Architecture: Recent Advances, Taxonomy, Requirements, and Open Challenges. *IoT: Protocol Stack, Cross-Layer, and Power Consumption Issues*, pp. 10-16.

Zachman, J. A. (1987). A framework for information systems architecture. *IBM Systems Journal, Vol. 26, No.3*, 454-470.

Zheng, L., Zhang, H., Han, W., Zhou, X., He, J., Zhang, Z., & Wang, J. (2011). Technologies, Applications, and Governance in the Internet of Things. In O. Vermesan, & P. Friess, *Internet of Things - Global Technological and Societal Trends.* Aalborg: River Publishers.

Zimmermann, O., Doubrovski, V., Grundler, J., & Hogg, K. (2005). Service-Oriented Architecture and Business Process Choreography in an Order Management Scenario: Rationale, Concepts, Lessons Learned. *20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, (pp. 301-312). San Diego.

## ***Appendix A: Other requirements***

| Other requirements | Source | Motivation |
|---|---|---|
| Multipoint communication | (Wang, Lee, & Murray, 2017) | Part of efficient connectivity and communication. |
| Simplified deployment | (Wang, Lee, & Murray, 2017) | Considered too specific for reference architecture, since the reference architecture is technology independent. |
| Virtualization management | (Rimal, Jukan, Katsaros, & Goeleven, 2011)<br>(Byers, 2017) | The architecture provides services which can be consumed by the applications. Specific virtualization is considered too specific for a reference architecture. |
| Fault tolerance | (Rimal, Jukan, Katsaros, & Goeleven, 2011) | Considered part of the high-availability requirement |
| Data collection & processing | (Weyrich & Ebert 2016),<br>(Rimal, Jukan, Katsaros, & Goeleven, 2011)<br>(Fremantle, 2015) | Data is not explicitly considered a requirement for the reference architecture, can be mapped on middleware layer. |
| Predictive analysis | (Fremantle, 2015) | Part of processing of data in middleware layer. |
| Provider service delivery model | (Rimal, Jukan, Katsaros, & Goeleven, 2011) | Not relevant for a reference architecture focusing on the execution of BPs. |
| Service centric issues | (Rimal, Jukan, Katsaros, & Goeleven, 2011) | Addressed by SOA, process adaptability and device management. |
| SOA principle | (Wang, Lee, & Murray, 2017) | Considered as a principle, rather than a requirement. |
| Dynamicity and runtime reconfiguration | (Wang, Lee, & Murray, 2017) | Considered too specific for reference architecture. |
| Interference management | (Yaqoob, et al., 2017) | Considered too specific for reference architecture, but partially addressed by efficient connectivity and communication. |
| Low latency | (Byers, 2017) | Addressed by efficient connectivity and communication. Concrete solution is considered too specific for reference architecture. |
| Reduced bandwidth | (Byers, 2017) | Considered too specific for reference architecture. |
| Geographic locality of control | (Byers, 2017) | Considered too specific for reference architecture. |
| Data rich mobility | (Byers, 2017) | Considered too specific for reference architecture. |
| Supporting advanced analytics and automation | (Byers, 2017) | Partially considered in process adaptability at workflow engine layer. |
| Hierarchical organization | (Byers, 2017) | Considered too specific for reference architecture. |

| | | |
|---|---|---|
| Energy efficiency | (Byers, 2017) | Part of efficient connectivity and communication. |
| Multi tenancy | (Byers, 2017) | Considered too specific for reference architecture. |

**Table 4 Other requirements**