RC25670 (WAT1710-015) October 9, 2017
Computer Science

IBM Research Report

Proceedings of the 11th Advanced Summer School on Service Oriented Computing

Johanna Barzen[1], Rania Khalaf[2],
Frank Leymann[1], Bernhard Mitschang[1], Editors

[1]University of Stuttgart
Germany

[2]IBM Research Division
One Rogers Street
Cambridge, MA 02142-1203
USA

The 11th Advanced Summer School on Service-Oriented Computing

June 25 - June 30
2017
Hersonissos, Crete, Greece

The 11th Advanced Summer School on Service Oriented Computing (SummerSOC'17) continued a successful series of summer schools that started in 2007, regularly attracting world-class experts in Service Oriented Computing to present state-of-the-art research during a week-long program organized in several thematic tracks: patterns and IoT, formal methods for SOC, computing in the clouds, data science, e-Health and emerging topics. The advanced summer school is regularly attended by top researchers from academia and industry as well as by PhD and graduate students.

During the different sessions at SummerSOC renowned researchers gave invited tutorials on subjects from the themes mentioned above. The afternoon sessions were also dedicated to original research contributions in these areas: these contributions have been submitted in advance as papers that had been peer-reviewed. Accepted papers were presented during SummerSOC and during the poster session. Furthermore, PhD students had been invited based on prior submitted and reviewed extended abstracts to present the progress on their theses and to discuss it during poster sessions. Some of these posters have been invited to be extended as a full paper, which are included in this technical report.

Johanna Barzen, Rania Khalaf, Frank Leymann, Bernhard Mitschang
- Editors -

Content

Applying IoT Patterns to Smart Factory Systems

Lukas Reinfurt[1,2], Michael Falkenthal[1], Uwe Breitenbücher[1], and Frank Leymann[1]

[1] University of Stuttgart, Institute of Architecture of Application Systems
Universitätsstr. 38, 70569 Stuttgart, Germany
`[firstname.lastname]@iaas.uni-stuttgart.de`
[2] Daimler AG
Epplestraße 225, 70567 Stuttgart, Germany
`lukas.reinfurt@daimler.com`

Abstract. Creating Internet of Things systems is a complex challenge as it involves both software and hardware, and because it touches on constrained devices and networks, storage, analytics, automation, and many other topics. This is further complicated by the large number of available technologies and the variety of different protocols and standards. To help with the ensuing confusion, we presented Internet of Things Patterns in several categories, such as device communication and management, energy supply types, and operation modes. These patterns describe abstract solutions to common problems and can be used to understand and design Internet of Things systems. In this paper, we show that these patterns can be applied to Smart Factory systems, which is one of the many domains where the Internet of Things is applicable.

Keywords: Internet of Things, Architecture, Patterns, Industry 4.0, Smart Factory, Industrial Internet

1 Introduction

Building Internet of Things (IoT) systems is a complex endeavor. It requires successfully combining both software and hardware across various domains. Sensing and actuation capabilities have to be brought into all kinds of environments using constrained devices and networks. Moreover, collected data has to be communicated and turned into usable information, sometimes fast and sometimes in huge quantities. Data and information has to be stored, has to be made accessible to others, and is used as basis for automation. Remote sensing and control offers great possibilities, but also high risks when it comes to security, privacy, and safety. This situation gets more complicated by the current state of the IoT field. As it is still relatively new and growing, a wide variety of technologies and solutions pushed by vendors from different areas are fighting for attention [2,15,18]. There is also an abundance of standard, as these solutions often have been created in silos [23]. This makes it confusing for IoT developers and architects to find appropriate technologies and solutions for their particular situation. To tackle these issues, we identified and collected *IoT Patterns*[3] in various categories that abstract from individual technologies [20,21,22]. During the design of IoT systems, architects can use IoT Patterns to solve specific problems they encounter. But applying the patterns is not limited to single problems. Rather, once one pattern has been applied, architects can now follow links to other related patterns. This enables them to build IoT systems step by step. As the IoT is applicable in many different domains, such as home automation, health care, logistics, and industrial fabrication, our IoT Patterns should also be applicable in these different domains. In this paper we will show that this is the case for industrial fabrication.

The remainder of this paper is structured as follows: In Section 2, we will present work related to IoT Patterns and their application. Section 3 introduces a motivating example from the domain of industrial fabrication which will be used in the remainder of the paper. Section 4 briefly describes the IoT Patterns which are relevant to this paper. Section 5 elaborates how these patterns can be used to describe and refine the system which was introduced in the motivating example. Finally, Section 6 ends the paper with a conclusion.

2 Related Work

We have published IoT Patterns for device energy supply and operation modes [22] and device communication and management [20,21]. Eloranta et al. presented patterns for building distributed control systems [5,6], which are concerned with reliability and fault-tolerance of large moving machines used for forrestry, mining, construction, etc. Qanbari et al. introduced four patterns for provisioning, deploying, orchestrating, and monitoring edge applications [19]. Another paper presents a pattern language for IoT applications [4], based largely on patterns from blog entries which are not comparable to our patterns in format or scope. Guth et al. compared several IoT Platform architectures to create an IoT reference architecture [16] to which our example system can be mapped. There are several publications that use patterns to design software architectures, such as [3,13]. But as the IoT includes a lot of physical things, our patterns are not only concerned with software, but also with the features of these things and how they shape and influence the IoT systems. There is also work by Falkenthal et al. which refines abstract patterns to patterns with more concrete, technology specific solution descriptions [9] or links them to concrete implementations [8,7]. This could be a next step to move our abstract IoT Patterns towards concrete solutions.

3 Motivating Example

One of the many domains where the IoT is applicable is industrial fabrication [10,11]. Here, as in many other domains, digitalization is advancing. This

[3] An up-to-date overview of all published IoT Patterns can be found on http://www.internetofthingspatterns.com

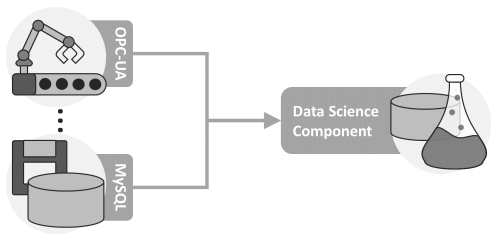

Fig. 1. Abstract high-level overview of the SePiA.Pro project.

has lead to the creating of various movement, such as Smart Factory, Industrial Internet, or Industry 4.0, which at their core have similar goals: to gain advanced insights and control of productions processes through wide spread digitalization and to further automate and optimize these processes.

Our example stems from the Industry 4.0 research project SePiA.Pro[4], which is situated in the domain of optimization [10]. Todays factories are comprised of many machines. By connecting several of these machines together, production lines are formed, which transform a part over several steps, for example by cutting, drilling, bending, welding, and painting. While single machines are often highly optimized, the overall optimization of production lines might be improved. The aim of SePiA.Pro is to build a self-service platform that enables analytics specialists to offer optimization services to large and small companies, which can instantiate these services on- or off-premise to optimize their existing machines and production lines with little or no technical expertise.

Figure 1 shows a very high level example of what this entails. On the left side, there are several data sources that provide the data which is used as input for the optimization services. These include data from the actual machines that should be optimized, but potentially also data from other sources, such as databases. The right-hand side of Figure 1 shows a *Data Science Component*, which takes this as input for analytical algorithms and produces a result that can be used to optimize machines and production lines. Of course there are several additional steps and obstacles between the left and right side that have to be looked at in more detail. For this we will use IoT Patterns.

4 IoT Patterns

The idea of patterns, which are abstract textual descriptions of proven solutions to reoccurring problems, goes back to Alexander's architecture patterns [1]. Since then, patterns have been published for all kinds of domains, also in the domain of IT [14,17]. In our previous work, we added to the already existing IT related patterns by publishing a collection of IoT Patterns [20,21,22]. Table 1 provides a brief overview of the IoT Patterns that are relevant in the context of this paper[3]. They include patterns concerned with device energy supply and device

[4] `http://projekt-sepiapro.de`

operation modes, as well as patterns concerned with communication and with data processing.

5 Applying IoT Patterns to Smart Factory Systems

Our motivating example shown in Figure 1 gives only a very general overview of what the SePiA.Pro project is trying to achieve. But by stepwise applying the existing IoT Patterns presented in Section 4, we can build this simple overview into a more full fledged architecture. We start on the left hand side of Figure 1 with the devices and other data sources.

5.1 Devices

The devices and potentially other data sources are the first point in the system where patterns can be applied. In some cases, these pattern can have a profound impact on the rest of the system architecture. There are two patterns that are applicable in this case, as shown in Figure 2.

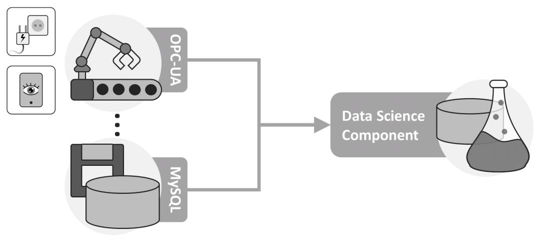

Fig. 2. Example with added Mains-Powered Device and Always-On Device.

One important question in IoT systems is how you provide energy to all the devices. In many systems you will have Period Energy-Limited Devices or Lifetime Energy-Limited Devices which are powered by batteries, or Energy-Harvesting Devices that gather their energy from their surroundings, for example with solar cells [22]. These kinds of devices often have a large impact on the overall system, as they are usually very constrained in their resources and only intermittently online [22]. In our case we are looking at large industrial machines that require a lot of power to operate and are therefore connected to the energy grid. Thus, they are Mains-Powered Devices (see Table 1) [22]. These kinds of devices have the advantage that they have all the power they need at their disposal and do not have to restrain themselves. On the other hand, they cannot be mobile and are depended on the power grid. In our case, this is not a problem and these devices do create no special problems regarding their energy supply.

Devices can use different operating modes to get the most out of the energy available to them. Device with limited energy are often Normally-Sleeping

Table 1. Short summary of the IoT Patterns used in this paper.

Icon	Description
	Mains-Powered Device Some devices have high energy requirements or are otherwise restricted so that powering them with batteries or energy harvesting is not an option. Connecting these devices to mains power provides them with plenty of energy [22].
	Always-On Device Some devices have to be constantly active and connected to fulfill their intended function or have virtually unlimited power available to them (e.g., Mains-Powered Devices). Leave these Always-On Devices connected and running at all times [22].
	Device Gateway Devices often differ in the communication technologies, protocols, or payload formats they use. Connect them to an existing network or system by using an intermediary Device Gateway that translates between the different communication methods [20].
	Device Shadow Devices go offline to save energy or because of network outages. Other components still want to interact with them. By storing a persistent virtual representation of devices and communicating through this copy only, other components can still work with offline devices [20].
	Rules Engine Throughout its operation a system receives a wide range of messages from devices and other components. A Rules Engine can react in different ways to these messages depending on their content, metadata, or additional external data sources. Each message is evaluated against a set of rules which trigger actions if they match [20].
	Remote Processing Some processing on the data produced by devices is very processing intensive, requires a lot of storage, or requires multiple data sources to be combined. Such processing steps may be too resource intensive for devices. Remote Processing runs the processing steps somewhere else, e.g., the Cloud, and returns the result to the originator.
	Local Processing Some situations require a fast reaction to sensor readings or other events. In such cases, first sending this data to Remote Processing components and then waiting for the answer may take too long. Local Processing integrates processing capabilities directly on or physically close to the devices where time-critical data is generated.

Devices which turn most of their components off for long periods of time in order to safe energy [22]. But in our case, as our industrial machines are used continuously to produce goods and are Mains-Powered Devices anyway, it makes little sense for them to sleep for long periods. Thus, they are Always-On Devices (see Table 1) [22] which, apart from high energy costs, bring only few disadvantages.

5.2 Communication

The patterns we applied so far had no impact on our overall system architecture. We continue with the middle part of our original overview in Figure 1, which is concerned with the communication between the data sources on the left and the *Data Science Component* on the right. Two patterns can be applied here as we described in the rest of this section.

As mentioned earlier, we can have all kinds of machines and other data sources which produce data in which we are interested for optimization purposes. But these data sources rarely use a single communication technology, protocol, or payload format. Industrial machines may use OPC-UA[5] or other industrial communication technologies, while other sources, for example databases, might be accessed via SQL[6]. This is the problem solved by the Device Gateway pattern (see Table 1) [20]. As shown in Figure 3, by adding a *Data Interface Unit*, which implements the Device Gateway pattern, we are able to translate different communication technologies so that they can be uniformly accessed by the *Data Science Component*.

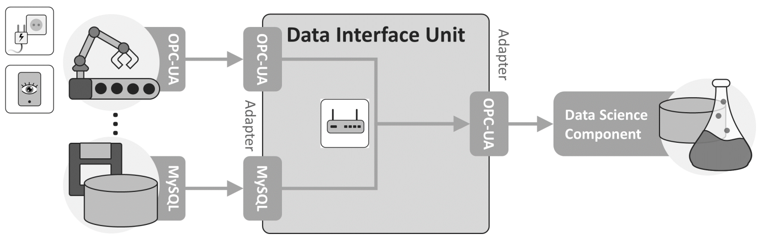

Fig. 3. Example with added Device Gateway pattern.

Although we have Mains-Powered Devices and Always-On Devices, there may be situations where they are unavailable, for example during maintenance or a power outage. Besides, some devices may not have a persistent storage for their data. In this case, a Device Shadow (see Table 1) helps as it stores the last known states and desired future states of all devices connected to it [20]. Thus, it is possible for other components, such as the *Data Science Component*

[5] https://opcfoundation.org/about/opc-technologies/opc-ua/
[6] https://www.iso.org/standard/63565.html

in our example, to access device data even if the device is currently not available. As can be seen in the middle of Figure 4, we added this functionality to the *Data Interface Unit* with the *Relay Service*.

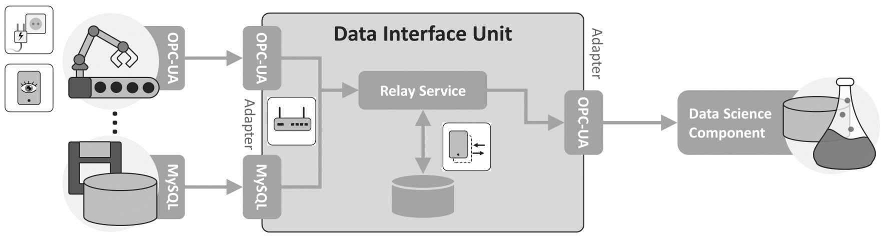

Fig. 4. Example with added DEVICE SHADOW pattern.

5.3 Processing

The communication patterns that we introduced in the last sections added new components to our example system which now allow us to communicate data from the different data sources to the *Data Science Component*. Now we turn our attention to the actual data processing. There are three patterns which can be applied in this area.

The algorithms and software needed for the analysis and optimization of machines and production lines vary depending on the use case. Thus, there is not only one *Data Science Component*, as shown in the previous figures. Instead, there are multiple different *Data Science Components* that can be selected. These are hosted on a common *Industrial Analytics Platform*, as shown in Figure 5. As an example, A RULES ENGINE (see Table 1) [20] could be one kind of *Data Science Component* or part of a *Data Science Component*.

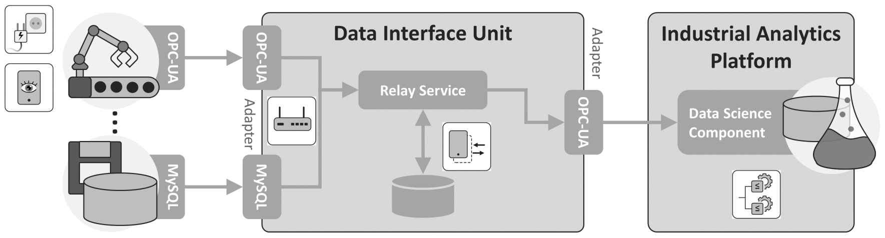

Fig. 5. Example with added RULES ENGINE pattern.

As mentioned earlier, the analytics capabilities provided by the SePiA.Pro platform should be usable for all kinds of small or large organizations. Some of these do not have the required IT infrastructure or technical knowledge to run the software provided to them. For such cases, SePiA.Pro offers the option of Remote Processing (see Table 1), where the analytics software is hosted remotely in the cloud and the data produced by the machines and other data sources is transferred into the cloud for analysis. To support this scenario, the *Data Interface Unit* and the *Industrial Analytics Platform*, which includes a *Data Science Component*, are packaged as a Smart Service [11]. This Smart Service is then provisioned by the *Smart Service Provisioning Engine* into a remote environment, as shown in Figure 6.

There may also be organizations using the SePiA.Pro platform for which sending all their data to a remote cloud is not an option for security and privacy reasons. Besides, in cases where a lot of data is produced and should be analyzed, sending all this data to a remote location may not be practical because of bandwidth limitations, high latency, or high costs. For such cases, SePiA.Pro also implements the Local Processing patterns (see Table 1). Here, the *Smart Service Provisioning Engine* is used to provision the Smart Service at the local premises of the company, where they are in full control of their data and IT infrastructure.

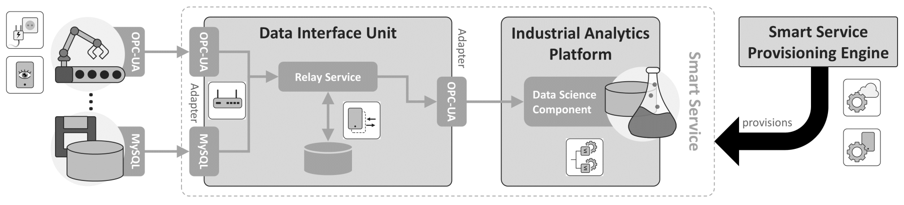

Fig. 6. The final example system after applying the Remote Processing and Local Processing patterns.

6 Conclusion

Industrial production and automation is one area where IoT will be relevant in the future. We have shown that our existing IoT Patterns can be applied in this area to better understand the consequences of choosing a particular solution, solve problems that appear in such systems, and provide an abstract architectural overview. In the future we work on adding more patterns to the already existing pattern catalog. One interesting area is security and privacy, where IoT Patterns would be of high relevance, especially in industrial scenarios such as the example described in this paper, where security and privacy is highly relevant. We are also planning to further refine the existing connections between the IoT Patterns into a pattern language, which gives IoT architects and designers more tools to find and apply the right patterns for their particular use case. Besides, we are working on methods that allow generic patterns to be refined to technology specific patterns [9] which can be linked into solution languages [12]. This could provide IoT architects additional support when implementing IoT systems based on IoT Patterns.

Acknowledgments. This work was partially funded by the project SePiA.Pro (01MD16013F) of the BMWi program Smart Service World.

References

1. Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language: Towns, Buildings, Construction. Oxford University Press, New York (1977)
2. Atzori, L., Iera, A., Morabito, G.: The internet of things: A survey. Computer networks 54(15), 2787–2805 (2010)
3. Buschmann, F., Meunier, R., Rohnert, H., Sornmerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns, vol. 1 (1996)
4. Chandra, G.S.: Pattern language for iot applications (2016)
5. Eloranta, V.P., Koskinen, J., Leppänen, M., Reijonen, V.: Designing distributed control systems: A pattern language approach. Wiley series in software design patterns, Wiley, Hoboken, NJ (2014)
6. Eloranta, V.P., Koskinen, J., Leppänen, M., Reijonen, V.: Patterns for the companion website (2014), `http://media.wiley.com/product_ancillary/55/11186941/DOWNLOAD/website_patterns.pdf`
7. Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F.: Efficient pattern application: Validating the concept of solution implementations in different domains. International Journal on Advances in Software 7(3&4), 710–726 (2014)
8. Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F.: From pattern languages to solution implementations. In: Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS 2014). pp. 12–21. IARIA, Wilmington, DE (2014)
9. Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F., Hadjakos, A., Hentschel, F., Schulze, H.: Leveraging pattern application via pattern refinement. In: Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change (PURPLSOC 2015)
10. Falkenthal, M., Breitenbücher, U., Christ, M., Endres, C., Kempa-Liehr, A.W., Leymann, F., Zimmermann, M.: Towards function and data shipping in manufacturing environments: How cloud technologies leverage the 4th industrial revolution. In: Proceedings of the 10th Advanced Summer School on Service Oriented Computing. pp. 16–25. IBM Research Report, IBM Research Report (2016)
11. Falkenthal, M., Breitenbücher, U., Képes, K., Leymann, F., Zimmermann, M., Christ, M., Neuffer, J., Braun, N., Kempa-Liehr, A.W.: Opentosca for the 4th industrial revolution: Automating the provisioning of analytics tools based on apache flink. In: Proceedings of the 6th International Conference on the Internet of Things. pp. 179–180. IoT'16, ACM (2016)
12. Falkenthal, M., Leymann, F.: Easing pattern application by means of solution languages. In: Proceedings of the Ninth International Conferences on Pervasive Patterns and Applications (PATTERNS) 2017. pp. 58–64. Xpert Publishing Services (2017)
13. Fernandez, E.B.: Security Patterns in Practice: Designing Secure Architectures Using Software Patterns. Wiley (2013)

14. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Massachusetts (1995)
15. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (iot): A vision, architectural elements, and future directions. Future Generation Computer Systems 29(7), 1645–1660 (2013)
16. Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., Reinfurt, L.: Comparison of iot platform architectures: A field study based on a reference architecture. In: Proceedings of the International Conference on Cloudification of the Internet of Things (CIoT). IEEE (2016)
17. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, Boston, Massachusetts (2004)
18. Ishaq, I., Carels, D., Teklemariam, G., Hoebeke, J., Abeele, F., Poorter, E., Moerman, I., Demeester, P.: Ietf standardization in the field of the internet of things (iot): A survey. Journal of Sensor and Actuator Networks 2(2), 235–287 (2013)
19. Qanbari, S., Pezeshki, S., Raisi, R., Mahdizadeh, S., Rahimzadeh, R., Behinaein, N., Mahmoudi, F., Ayoubzadeh, S., Fazlali, P., Roshani, K., Yaghini, A., Amiri, M., Farivarmoheb, A., Zamani, A., Dustdar, S.: Iot design patterns: Computational constructs to design, build and engineer edge applications. In: Proceedings of the First International Conference on Internet-of-Things Design and Implementation (IoTDI). pp. 277–282. IEEE (2016)
20. Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., Riegg, A.: Internet of things patterns. In: Proceedings of the 21st European Conference on Pattern Languages of Programs (EuroPLoP). ACM (2016)
21. Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., Riegg, A.: Internet of things patterns for communication and management. LNCS Transactions on Pattern Languages of Programming (2017)
22. Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., Riegg, A.: Internet of things patterns for devices. In: Proceedings of the Ninth International Conferences on Pervasive Patterns and Applications (PATTERNS) 2017. pp. 117–126. Xpert Publishing Services (2017)
23. Singh, J., Pasquier, T., Bacon, J., Ko, H., Eyers, D.: Twenty security considerations for cloud-supported internet of things. IEEE Internet of Things Journal 3(3), 269–284 (2016)

Towards Predicting Availability of IoT Cloud Solutions

Otto Bibartiu

University of Stuttgart
Universitätsstraße 38
70569 Stuttgart, Germany
Otto.Bibartiu@ipvs.uni-stuttgart.de

Abstract. The Internet of Things (IoT) requires highly scalable infrastructure to handle the large amount of data from IoT devices. Therefore, many IoT solutions use cloud computing to profit from the economy of scale. Besides scalability, many of these solutions require service level agreements with well-defined quality of service (QoS), which are challenging to define considering the complex environment of a cloud ecosystem. This paper gives a basic introduction to the authors's PhD topic tackling this problem. The main goal of this thesis is to provide the IoT solution architect with concepts and tools to predict QoS, and additionally optimize the architecture to utilize cloud services in a cost efficient manner while fulfill its QoS.

Keywords: IoT, Cloud, Availability, Prediction, Architecture

1 Introduction and Motivation

In the Internet of Things (IoT), virtually everything will be connected to the Internet, including, in particular, devices integrating sensors and actuators. Gartner estimates that 20 billion devices will be IoT enabled by the end of 2020[1]. In order to manage the huge population of IoT devices and the large amount of data stemming from these devices, a highly scalable infrastructure is required. Cloud computing promises to provide such a scalable infrastructure at low cost utilizing commodity hardware, virtualization techniques, and resource sharing concepts.

However, running IoT applications atop a virtualized and shared infrastructure makes it challenging to provide these applications with a well-defined quality of service (QoS). In particular, safety-critical IoT applications have stringent requirements with respect to response time (latency), and availability. For instance, an emergency call (eCall) IoT cloud solution, depicted in Figure 1, can automatically call for help after a car collision. The eCall solution needs to be highly available and guarantee a response time in the range of seconds from the

[1] Gartner, Inc. http://www.gartner.com/newsroom/id/3598917, Last Accessed: 27.04.2017

detection of the collision by a sensor embedded in the car until triggering further actions to help the driver. Consequently, we need concepts and tools to assist the IoT cloud solution architect to design and implement cloud applications fulfilling a well-defined QoS with respect to latency and availability.

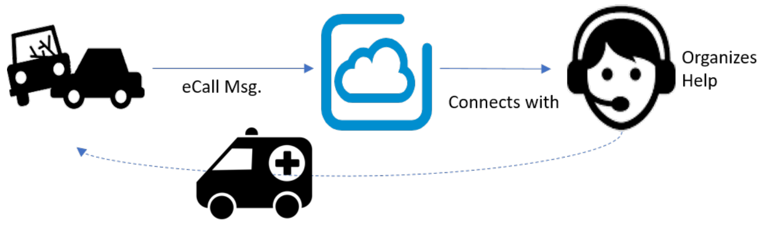

Fig. 1: Example of the emergency Call (eCall) IoT use case.

Thus, the goal of this PhD project is to design concepts to predict the QoS of a given solution architecture and more importantly optimize the architecture to utilize cloud services in a cost efficient manner and still fulfill its QoS. This paper is providing a broad overview on this PhD topic and also discuss about related work and research gaps.

The rest of this paper is structured as fallows, Section 2 presents preliminary evaluations of related work. Finally, Section 3 provides a detailed description of the PhD topic and the main research gaps.

2 Related Work

The early approaches to predict software availability, as proposed for instance by Cheung [1], sought to predict availability by modeling dependencies and failure rates manually as (discrete-time) Markov Chains as depicted in Figure 2. A software system is modeled as a control flow graph where each node N_i is a software module, and each edge (N_i, N_j) a possible transfer of control to module N_j. In case of a failure, every node might transit to an absorbing failure state F. The software system is available when the control flow reaches a successful state C.

However, such approaches do not take response time into account and each component needs prior availability annotations, which makes large cloud applications hard to model. In order to avoid complex modeling of Markov Chains, concepts have been proposed of mapping use case scenarios or UML diagrams to some Markov Chain structure [2]. Such concepts ease the design process, but still requires a domain expert to annotate the model correctly.

Recent research in availability prediction has begun to include performance analysis and offer better abstractions to model complex distributed systems. For example, the Palladio Component Model (PCM), which is widely used in the

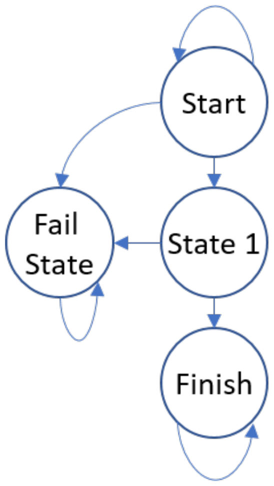

Fig. 2: Example of a user-Oriented reliability software model by Cheung [1].

field of software engineering to analyze the performance of distributed systems, has been extended by Brosch et al. [3] with the feature of availability prediction. For example, Figure 3 depicts a MongoDB instance with one CPU and one hard drive (HD); additional components can be added and it is even possible to model use case flows within the models. In particular the PCM concept might also be useful to model Micro Services.

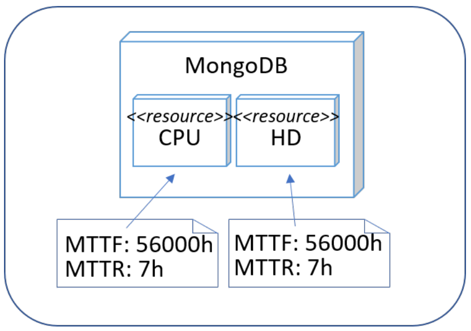

Fig. 3: Example of a basic MongoDB instance with one CPU and one hard drive (HD)

Nevertheless, PCM still requires manual availability annotations of components during the design phase, and it cannot model cloud specific mechanics like auto scaling or VM (Virtual Machine) migration.

Dedicated approaches to model cloud availability have only focused on the Infrastructure as a Service (IaaS) layer. Jammal et al. and Kim et al. provide availability prediction models based on Stochastic Petri Nets [4, 5] which help to model multi-tier applications and include physical servers combined in clusters, VMs, and load balancing into their models.

However, these approaches are missing interaction with cloud services offered as Platform as a Service (PaaS), where clients can utilize cloud services like databases, message brokers, etc. Most importantly they solely focus on dedicated services, whereas most clouds also provide shared hardware and software services to their clients.

Another research trend in availability prediction is to avoid manual annotation of availability of software components. To this end, Pitakarat et al. provide methods to predict the availability of distributed systems by applying machine learning methods to monitor data [6], but this requires the IoT application to be already deployed and running in the cloud, rather than evaluating an undeployed architecture during development. A possible solution might be to combine the architecture of an IoT application together with a online prediction model of the cloud. In general, current software reliability solutions are not sufficient to model the high complexity of cloud applications.

3 Research Goals

Cloud concepts like VM migration, auto scaling of service instances, fail-over techniques, influence of shared services, interaction between the cloud layers (IaaS, PaaS, Software as a Service), etc. have not been investigated deeply enough. Therefore, this PhD project focuses its research on providing solutions which help the IoT solution architect to predict the availability of his architecture before deploying the application to the cloud, and more importantly optimize the architecture to be cost efficient and still fulfill its availability requirements when running in the cloud.

Figure 4 gives an overview of the project goals. A *Three-Tier Cloud Pattern* architecture (left) is used as an example IoT solution [7]. This architecture uses two RabbitMQ nodes as messaging middleware to decouple the presentation tier from the business tier . Additionally, the business tier is connected to a dedicated MongoDB service.

First, this project aims to provide concepts to predict QoS, e.g., availability and performance, of the IoT architecture before deploying the architecture to the cloud. Therefore, the cloud ecosystem – here modeled with the basic three layer; IaaS, PaaS, SaaS – and the additional management interface, need to be taken into consideration since QoS depends on running services, service dependencies, complex cloud patterns [7], and how they are utilized by other tenants.

In the second step, these concepts and tools needs be enhanced to also propose a cost effective architecture which still fulfills the clients's QoS requirements. For example, the architecture on the left of Figure 4 might provide a higher availability than expected. To reduce cost, design tools might suggest

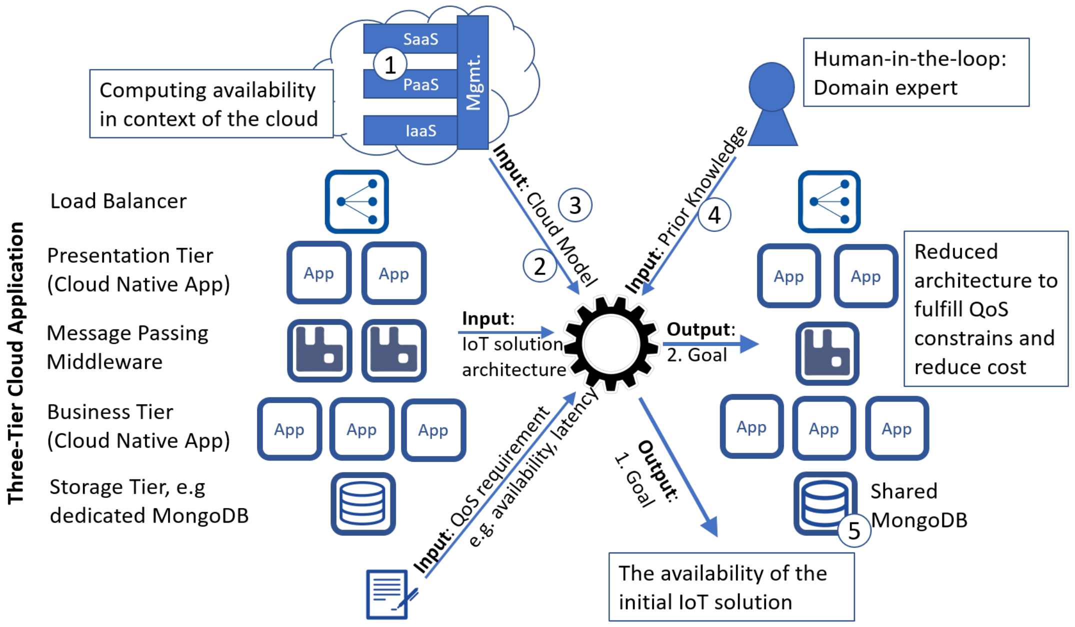
Computing availability in context of the cloud
SaaS
PaaS
IaaS
Mgmt.
1
Human-in-the-loop: Domain expert
Three-Tier Cloud Application
Load Balancer
Presentation Tier (Cloud Native App)
Message Passing Middleware
Business Tier (Cloud Native App)
Storage Tier, e.g dedicated MongoDB
App
Input: Cloud Model
3
2
Input: Prior Knowledge
4
Input: IoT solution architecture
Output: 2. Goal
Reduced architecture to fulfill QoS constrains and reduce cost
Input: QoS requirement e.g. availability, latency
Output: 1. Goal
The availability of the initial IoT solution
Shared MongoDB
5

Fig. 4: Research and project goals.

other service configurations (right side of Figure 4), e.g., using one RabbitMQ node instead of two, and utilizing a shared MonogoDB instated of a dedicated MongoDB instance.

Generally, proving these tools requires to tackle the following main research gaps (see numerical labels in Figure 4):

1. Mapping of cloud structure to availability model.
2. Influence of shared services on availability.
3. Dependency interaction between cloud layers (IaaS, PaaS, SaaS).
4. Infer availability with little or without estimates of domain experts avoiding human in the loop.
5. Modeling complex virtualization concepts and cloud computing patterns, like service migration between datacenters, deployment chains, container nesting.

In a first attempt to solve these research gaps, we focus on Bayesian approaches [8, 9] and probabilistic graph models [10], like Bayesian Networks [11]. Bayesian Networks are helpful to model uncertainty, compute inference, and to handle missing values. Additionally, Object Oriented Bayesian [12] Networks help to manage software components, like Micro Service, in an hierarchical manner.

Finally, the IoT solution architect should receive tools which abstract away the underlining availability model, by providing an environment similar to PCM, or offering an interface where the architect combines cloud patterns [7] to complex IoT solutions.

4 Conclusion

Holding IoT applications in the cloud requires a high focus on availability, i.e. QoS, for IoT solutions. To this end, this PhD project is investigating solutions to predict availability of an IoT architecture before deploying the solution to the cloud and more importantly optimize the architecture to be cost efficient and still fulfill its availability requirements when running in the cloud. Our preliminary evaluation of related work has shown that many cloud specific concepts and patterns have not been taken into account when modeling availability, especially, the interaction between cloud layers and influence of shared service. One promising method to model availability are probabilistic graph models, like Bayesian Networks. Therefore this PhD project focuses on concepts to combine cloud patterns with mathematical models, in order to deliver QoS predictions which help the IoT solution architect in designing his IoT solution.

5 Acknowledgement

We gratefully acknowledge the support of this PhD project by Robert Bosch GmbH within the Doctoral Academy.

References

1. R. C. Cheung, "A user-oriented software reliability model," *IEEE transactions on Software Engineering*, 1980.
2. S. Yacoub, B. Cukic, and H. H. Ammar, "A scenario-based reliability analysis approach for component-based software," *IEEE transactions on reliability*, vol. 53, no. 4, pp. 465–480, 2004.
3. F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner, "Architecture-based reliability prediction with the palladio component model," *IEEE Transactions on Software Engineering*, 11 2012.
4. M. Jammal, A. Kanso, P. Heidari, and A. Shami, "A formal model for the availability analysis of cloud deployed multi-tiered applications," in *Cloud Engineering Workshop (IC2EW), 2016 IEEE International Conference on*, pp. 82–87, IEEE, 2016.
5. D. S. Kim, F. Machida, and K. S. Trivedi, "Availability modeling and analysis of a virtualized system," in *2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*, pp. 365–371, 11 2009.
6. T. Pitakrat, D. Okanovic, A. Van Hoorn, and L. Grunske, "Hora: Architecture-aware online failure prediction," in *Quality of Software Architectures (QoSA), 2016 12th International ACM SIGSOFT Conference on*, 2016.
7. C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications.* Springer, 2014.
8. H. Pham, *System software reliability.* Springer Science & Business Media, 2007.
9. V. Cortellessa, H. Singh, and B. Cukic, "Early reliability assessment of uml based software models," in *Proceedings of the 3rd International Workshop on Software and Performance*, WOSP '02, (New York, NY, USA), pp. 302–309, ACM, 2002.

10. D. Koller, N. Friedman, L. Getoor, and B. Taskar, "Graphical models in a nutshell," *Introduction to statistical relational learning*, pp. 13–55, 2007.
11. R. E. Neapolitan, N. I. University, and I. Chicago, *Learning bayesian networks*. Pearson Prentice Hall Upper Saddle River, NJ, 2004.
12. O. Bangso and P.-H. Wuillemin, "Object oriented bayesian networks: A framework for top-down specification of large bayesian networks with repetitive structures," 2000.

Declarative vs. Imperative: How to Model the Automated Deployment of IoT Applications?

Uwe Breitenbücher, Kálmán Képes, Frank Leymann, and Michael Wurster

Institute of Architecture of Application Systems, University of Stuttgart
Universitätsstr. 38, 70569 Stuttgart, Germany
`[firstname].[lastname]@iaas.uni-stuttgart.de`

Abstract. The Internet of Things (IoT) has become an increasingly important domain, which more and more requires application deployment automation as manual deployment is time-consuming, error-prone, and costly. However, the variety of available deployment automation systems also increases the complexity of selecting the most appropriate technology. In this paper, we discuss how the deployment of complex composite IoT applications can be automated and discuss the conceptual strengths and weaknesses of declarative and imperative deployment modelling.

Keywords: Deployment Modelling, Declarative, Imperative, TOSCA

1 Introduction

The Internet of Things (IoT) has become an increasingly important domain as more and more IoT applications influence our life. IoT applications typically consist of physical devices that are connected to software, which is often deployed in cloud environments. Thus, IoT applications are *cyber-physical systems* that consist of one or more physical parts and virtual parts. However, especially the combination of physical devices and cloud-based software deployments quickly leads to complex architectures as multiple physical as well as virtual components have to be deployed, configured, and wired. As a result, the deployment of such *complex composite IoT applications* is a serious challenge that requires immense technical expertise [28]: Physical devices must be installed, scripts deployed, sensors configured, and backend software provisioned. Due to this complexity manually deploying IoT applications is time-consuming, error-prone, and costly. However, although various deployment approaches exist to automate the deployment of IoT applications, the variety of available technologies makes it very difficult to select the most appropriate technology for a certain use case.

In this paper, we discuss how the deployment of such complex composite IoT applications can be automated by analyzing the conceptual strengths and weaknesses of declarative and imperative deployment approaches based on our experiences we gained in the BMWi project SmartOrchestra[1]. Thus, we do not focus on individual technologies but on the general *deployment modelling concepts* and discuss their suitability for different IoT deployment use cases.

[1] `http://www.smartorchestra.de`

2 Fundamentals & Related Work

The declarative deployment modelling approach is based on *declarative deployment models* that describe the structure of the application to be deployed including all components, their configuration, and their relationships. A declarative deployment model is consumed by a declarative deployment system that interprets the model, derives all required technical tasks to deploy the described application, and executes these tasks [11]. Thus, declarative deployment models specify only *what* has to be deployed, while the actual deployment logic gets calculated by the deployment system and is, therefore, not contained in the model. There are multiple scientific works that support the declarative approach for modelling the deployment of applications, for example, by Eilam et al. [9], Maghraoui et al. [10], Hewson et al. [13], and Breitenbücher et al. [6, 5]. Moreover, configuration management technologies such as Puppet [27] often support declarative deployment modelling, too. The *Topology and Orchestration Specification for Cloud Applications (TOSCA)* [22, 21, 24, 3] is a standard that enables automating the deployment of cloud applications. TOSCA also supports the declarative approach as it provides a metamodel for modelling the topology of the application to be deployed including all components, their configurations, and their relationships.

In contrast, the imperative deployment modelling approach is based on *imperative deployment models* that describe the actual deployment logic to be executed in the form of a process [11]. Thus, imperative deployment models are process models that explicitly describe all technical deployment tasks to be executed, their order, and the data flow between these tasks. For application deployment and management automation, often the workflow technology [18] is used to describe the corresponding processes in an executable manner. For example, the approaches presented by Mietzner et al. [20], Bellavista et al. [1], Keller et al. [14], and Breitenbücher et al. [5, 8] are based on the workflow technology. Moreover, there are domain-specific extensions for workflow languages that focus on deployment, for example, BPMN4TOSCA [15, 17] or the approach presented by Weerasiri et al. [30]. In practice, low-level shell scripts are often used as well to describe software installation and configuration tasks in an imperative manner. Imperative deployment models are executed by a corresponding process engine, for example, a workflow engine, or by an imperative deployment system such as OpenTOSCA [2], which is an open-source runtime for TOSCA. Thus, TOSCA also supports the imperative deployment modelling approach by the concept of *Management Plans*, which are executable process models that can be used to automate the deployment of the modelled application.

We documented both deployment modelling approaches in our previous work [11] in the form of *Application Deployment Modelling Patterns.* This work also categorizes some available deployment automation technologies based on the two modelling approaches. In this paper, we discuss the general suitability of the two approaches for the deployment of complex composite IoT applications with respect to requirements of the IoT domain. We first describe the conceptual strengths and weaknesses of the declarative modelling approach in the next Section 3, which is followed by the imperative part discussed in Section 4.

3 Declarative IoT Deployment Modelling

In this section, we discuss the strengths and drawbacks of declarative deployment modelling with respect to the domain of IoT. To provide a conceptual overview, the discussion does not compare concrete technologies but discusses the general suitability of declarative IoT deployment modelling. Each following subsection discusses one certain strength or drawback. To support understanding and to better illustrate problems, we provide examples based on the TOSCA standard.

3.1 Creation and Comprehensibility of the Deployment Model

Declarative deployment models capture the structure of the system to be deployed as they describe the components that shall be deployed as well as their relationships, thus, the *topology* of the application. This directly reflects the application's structure developers have in mind during development, which provides an intuitive modelling approach and directly shows the final result. Moreover, especially for IoT deployments directly capturing the involved physical devices as well as their connections to software components eases the creation and understanding of the deployment model. In addition, for declarative deployment languages typically graphical modelling tools are available that enable a fast creation of the respective models. For example, the open-source TOSCA modelling tool *Winery* [16] supports graphically modelling TOSCA-based declarative deployment models based on the visual notation Vino4TOSCA [7].

3.2 Suitability of Declarative Technologies

Many IoT applications are composed of common components such as Raspberry Pis and IoT middlewares[2] such as the Mosquitto message broker. Moreover, typically standardized communication protocols, such as MQTT [23], are used that are explicitly defined regarding their technical details. As declarative deployment models are interpreted by the deployment system, the components to be provisioned and their relationships must be processable by the system. In a previous work [28], we have shown that automatically deploying IoT applications that are composed of such common components and protocols is possible based on declarative TOSCA models. However, if customization and application-specific deployment logic is required, the declarative approach reaches its limits as the system needs to interpret the declarative model that specifies only *what* has to be provisioned, but not *how*. To tackle this issue, declarative deployment technologies typically provide plug-points, which can be used to specify custom deployment logic for individual components. For example, the *TOSCA Lifecycle Interface* [21, 24] enables to provide an own install implementation for a certain type of component. However, this is typically limited to lifecycle operations and does not allow to customize the deployment arbitrarily [4]. Thus, the declarative modelling approach is mainly suited for common and non-complex deployments, but is limited regarding individual customizations and application-specific details.

[2] An overview of different IoT Integration Middlewares is provided by Guth et al. [12]

3.3 Required Technical Deployment Expertise

Declarative deployment models specify only *what* has to be deployed, but not *how* the deployment shall be executed. Therefore, modellers only need to specify the application's structure including the components, their wiring, and the desired component configurations. Thus, only little or even no technical expertise is required about the actual deployment execution: Neither scripting languages have to be understood nor API calls must be orchestrated, which is typically required for the deployment of complex systems [8]. Especially in the domain of IoT, this characteristic becomes of vital importance as deploying and configuring software on (remote) physical IoT devices is typically more complex than solely deploying software in cloud environments as more technical deployment and configuration tasks have to be executed [28]. The additional technical tasks range from, for example, connecting to (remote) physical devices for installing software to configuring gateways in order to establish the communication between devices and backend. The immense heterogeneity regarding IoT middleware systems [12] additionally increases this complexity. Therefore, only modelling the structure of the system to be deployed requires significantly less technical expertise than the imperative modelling approach, which has to specify all technical deployment tasks, remote communications with devices, API calls, script executions, etc.

3.4 Deployment Customization

Declarative deployment models are interpreted by the deployment system, which derives and executes the technical deployment tasks [11]. This interpretation is typically based on (i) known types of components and relationships and (ii) known management interfaces. For example, the OpenTOSCA deployment system provides a plug-in system for the deployment of different component types [5]. Many declarative technologies also support mechanisms to inject custom deployment logic into the model based on known management interfaces. For example, TOSCA not only provides a metamodel for declarative deployment models but also standardizes the *TOSCA Lifecycle Interface*, which defines the operations that are called during the deployment of a component, e.g., *install* and *start*. Moreover, TOSCA enables to provide own implementations for these operations on a per-component basis in the model. Thus, based on such interfaces, the deployment logic of a component can be influenced even if a declarative deployment technology is used. However, if more complex tasks must be executed that do not follow such predefined operations, the declarative approach reaches its limitations: The deployment can be customized only using such plug-points, but not in an arbitrary manner. For example, if two different components must be installed before both can be started, this cannot be realized using the TOSCA Lifecycle Interface. Especially in the domain of IoT this is a critical limitation as the configuration of physical devices often needs a special deployment execution order. For example, often two devices must be physically connected before software can be installed. Thus, the relationship between these two components must be established before software can be deployed on the devices, which breaks the typical deployment execution order of components and their relationships [5].

3.5 Integration of Human Tasks

IoT applications are cyber-physical systems and, thus, consist of one or more virtual parts and one or more physical parts. For automating the deployment of the virtual part, there are many declarative deployment technologies available that are capable of executing arbitrary deployment tasks, for example, creating virtual machines, installing Web-based applications, and instantiating a database on a Storage as a Service offering such as Amazon RDS[3]. All these deployment tasks have in common that they can be executed fully automatically without the need for human intervention as the components that have to be deployed as well as the components on which they have to be deployed can be accessed by software, e.g., via HTTP-based APIs of hypervisors and cloud service offerings or low-level communication protocols to access virtual machines. In contrast, deploying the physical part of IoT applications often requires humans, e.g., for installing devices, soldering sensors, etc. However, such *human tasks* are typically not natively supported by declarative modelling approaches and are, therefore, very hard to integrate into the available deployment systems and the corresponding models. For example, in TOSCA it would be possible to implement the *install* operation of a physical device component by a script that sends a message to a human to install the device. This means that typical workflow features such as staff resolution, work item management, and role management [18] would have to be re-implemented in such solutions as they are typically not supported natively by declarative deployment systems. However, the reliability and robustness of workflow management systems [18] cannot be achieved using such workarounds.

3.6 State-preserving and State-Changing Deployment Tasks

Deployment and management tasks can be abstractly classified into (i) *state-changing tasks* and (ii) *state-preserving tasks* [4]. State-changing task change the state of one or more components or relationships of the application, for example, modifying the HTTP port of a Webserver changes the state of the component. In contrast, state-preserving tasks do not change the state of one or more components or relationships, for example, exporting data from a database does not change the state or configuration of the database or of its stored data. Declarative models support state-changing deployment tasks natively: Components are transferred from state *uninstalled* to state *installed*, for example. However, this type of deployment models does not support state-preserving tasks very well as this kind of tasks cannot be modelled by specifying a component, a relationship, or a configuration [4]. Especially for deploying IoT applications this is a serious problem as state-preserving tasks are often required. For example, before connecting a physical device to the backend system, its physical functionality and also non-functional requirements such as its battery level may have to be checked. In particular, if a device has an actuator that triggers some physical action, a declarative model cannot specify that the physical functionality has to be verified manually after the successful installation of the device or its software.

[3] `https://aws.amazon.com/rds/`

4 Imperative IoT Deployment Modelling

In this section, we discuss the general strengths and drawbacks of imperative deployment modelling with respect to the domain of IoT. Where possible we refer to the declarative strengths and drawbacks to compare the both approaches.

4.1 Deployment Customization

Imperative deployment models are process models that specify a set of activities to be executed as well as their order and the control flow between them. Thus, they specify exactly *how* a deployment has to be executed [11]. This enables influencing the deployment execution arbitrarily as each detail can be described and customized in the process model, for example, the deployment order of components can be changed or application-specific customizations can be implemented by additional tasks. Thus, in contrast to the declarative approach, imperative deployment models support state-preserving as well as state-changing tasks. In particular, for deploying complex composite IoT applications also tasks that are hard to describe declaratively can be realized, for example, installing a physical device at a certain place or testing a device before connecting it to the backend. TOSCA also supports the imperative deployment modelling approach in the form of so-called *Management Plans*, which are executable process models that automate the execution of a certain management function for the application, e.g., its deployment. Thus, the TOSCA standard supports the declarative as well as the imperative approach for deploying applications, which therefore provides a suitable basis to choose the right modelling approach for a certain IoT deployment use case. In particular, there are previous works that show how the TOSCA standard can be used for IoT deployment automation [19, 28, 29].

4.2 Integration of Human Tasks

The integration of human tasks in the automated declarative deployment of an IoT application is hard to realize and may misuse concepts provided by the deployment technology (cf. Section 3.5). Using imperative deployment models this is much easier as especially many workflow languages and workflow management systems support the integration of human tasks in automatically executed processes [18]. For example, the Business Process Model and Notation (BPMN) [25] defines a task type for integrating manual human task executions in an overall automated workflow. Thus, this kind of deployment model is suited for IoT application deployments in which manual task executions by humans are required, for example, to install or configure physical devices at a certain place. However, the available standards-based domain-specific workflow extensions for application deployment such as BPMN4TOSCA [15, 17] currently do not support the integration of human IoT deployment tasks, which is therefore part of our future work.

4.3 Required Technical Deployment Expertise

Imperative deployment models describe each detail about the deployment tasks to be executed, for example, technical details of script invocations, API calls, and file transfers to virtual machines. In addition, also the control flow of these tasks must be specified as well as the data flow between them. Moreover, when deployment tasks shall be executed in parallel, this quickly leads to complex process models that must be developed and maintained carefully. Thus, manually creating imperative deployment models is a complex and technically error-prone challenge [8]. Especially when multiple deployment technologies must be orchestrated, for example, for multi cloud or hybrid cloud deployments, different API designs, data formats, invocation mechanisms, and security concepts of the different deployment technologies and provider APIs also increase the complexity of imperative deployment models [8]. IoT applications additionally increase this complexity as also physical devices must be considered in the deployment process, which often requires establishing connections to devices, transferring files, and executing scripts—all these tasks must be reflected in the imperative process model. As a result, the development and maintenance of complex imperative IoT deployment models requires immense technical deployment expertise of possibly multiple different deployment systems and APIs that have to be combined. Therefore, the imperative approach is complex, error-prone, and time-consuming [8]. The TOSCA standard enables to reduce the complexity of imperative deployment models as deployment logic can be hidden by the lifecycle operation implementations of the components (cf. Section 3.4). Thus, a deployment plan can invoke these operations, which wrap technical details. Moreover, in previous work [31], we also developed a TOSCA-based *management bus* that encapsulates deployment technologies such as Chef [26] from plans. As a result, plans only invoke this bus in a standardized manner to execute a deploying task using a certain technology without the need to care about the technical invocation details.

4.4 Comprehensibility of the Deployment Model

Declarative deployment models specify the structure of the application including all components and relationships. Especially when graphical, graph-based representations are used, declarative models provide a comprehensive overview on the application to be deployed (cf. Section 3.1). In contrast to this, imperative deployment models specify the process of the deployment. Thus, the final result is not immediately visible in these imperative models and must be derived by analyzing the modelled tasks, their semantics, and their order. This quickly gets complex if an IoT application is complex and consists of various physical as well as virtual components. Moreover, in IoT applications often multiple devices of the same kind are involved, for example, devices with temperature sensors in a smart home. Thus, in such scenarios not only the deployment tasks and possibly their labels in the process model must be analyzed, but also their parameters must be understood to recognize which component is affected by a certain task.

5 Conclusion and Future Work

In this paper, we discussed the suitability of the declarative and the imperative deployment modelling approaches for automating the deployment of IoT applications. We analyzed the major conceptual strengths and weaknesses of both approaches and mainly compared them in terms of complexity for modellers and the required technical expertise. The paper shows that the major drawbacks of each approach is solved by the other one, which leads to the conclusion, that a hybrid IoT deployment modelling approach is required: A declarative deployment model eases the specification of the desired deployment but is limited to common and non-complex deployments. Thus, transforming declarative models into imperative deployment models enables customizing the IoT application deployment arbitrarily as any additional task can be added and even human tasks and state-preserving tasks can be included in the generated process model. It has been already shown that this transformation is possible for software-based application deployments [10, 9, 5] and also that simple declarative IoT application deployment models can be transformed into imperative workflows [29]. Therefore, in future work, we focus on this transformation—especially on the integration of human tasks into the automated deployment process. This requires a detailed analysis of existing imperative deployment modelling languages such as BPMN4TOSCA and new concepts that support humans in executing such manual IoT deployment tasks. For example, to install a certain software via USB stick on a device that cannot be managed remotely due to missing connectivity to the network.

Acknowledgments. This work was partially funded by the project SmartOrchestra (01MD16001F) of the BMWi program Smart Service World.

References

1. Bellavista, P., Corradi, A., Foschini, L., Pernafini, A.: Towards an Automated BPEL-based SaaS Provisioning Support for OpenStack IaaS. Scalable Computing 14(4), 235–247 (2013)
2. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: OpenTOSCA - A Runtime for TOSCA-based Cloud Applications. In: Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013). pp. 692–695. Springer (Dec 2013)
3. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: Portable Automated Deployment and Management of Cloud Applications, pp. 527–549. Advanced Web Services, Springer (Jan 2014)
4. Breitenbücher, U.: Eine musterbasierte Methode zur Automatisierung des Anwendungsmanagements. Dissertation, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik (2016)
5. Breitenbücher, U., Binz, T., Képes, K., Kopp, O., Leymann, F., Wettinger, J.: Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA. In: International Conference on Cloud Engineering (IC2E 2014). pp. 87–96. IEEE (Mar 2014)

6. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F.: Pattern-based Runtime Management of Composite Cloud Applications. In: Proceedings of the 3rd International Conference on Cloud Computing and Services Science (CLOSER 2013). pp. 475–482. SciTePress (May 2013)
7. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Schumm, D.: Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA. In: On the Move to Meaningful Internet Systems: OTM 2012 (CoopIS 2012). pp. 416–424. Springer (Sep 2012)
8. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Wettinger, J.: Integrated Cloud Application Provisioning: Interconnecting Service-Centric and Script-Centric Management Technologies. In: On the Move to Meaningful Internet Systems: OTM 2013 Conferences (CoopIS 2013). pp. 130–148. Springer (Sep 2013)
9. Eilam, T., Elder, M., Konstantinou, A.V., Snible, E.: Pattern-based Composite Application Deployment. In: Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011). pp. 217–224. IEEE (May 2011)
10. El Maghraoui, K., Meghranjani, A., Eilam, T., Kalantar, M., Konstantinou, A.: Model Driven Provisioning: Bridging the Gap Between Declarative Object Models and Procedural Provisioning Tools. In: Proceedings of the 7th International Middleware Conference (Middleware 2006). pp. 404–423. Springer (Nov 2006)
11. Endres, C., Breitenbücher, U., Falkenthal, M., Kopp, O., Leymann, F., Wettinger, J.: Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications. In: Proceedings of the 9th International Conference on Pervasive Patterns and Applications (PATTERNS). pp. 22–27. Xpert Publishing Services (Feb 2017)
12. Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., Reinfurt, L.: Comparison of IoT Platform Architectures: A Field Study based on a Reference Architecture. In: Cloudification of the Internet of Things (CIoT). IEEE (Nov 2016)
13. Hewson, J.A., Anderson, P., Gordon, A.D.: A Declarative Approach to Automated Configuration. In: Proceedings of the 26th Large Installation System Administration Conference (LISA). pp. 51–66. USENIX (Dec 2012)
14. Keller, A., Badonnel, R.: Automating the Provisioning of Application Services with the BPEL4WS Workflow Language. In: Proceedings of the 15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2004). pp. 15–27. Springer (Nov 2004)
15. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: BPMN4TOSCA: A Domain-Specific Language to Model Management Plans for Composite Applications. In: Proceedings of the 4th International Workshop on the Business Process Model and Notation (BPMN 2012). pp. 38–52. Springer (Sep 2012)
16. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery – A Modeling Tool for TOSCA-based Cloud Applications. In: Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013). pp. 700–704. Springer (Dec 2013)
17. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F., Michelbach, T.: A Domain-Specific Modeling Tool to Model Management Plans for Composite Applications. In: Proceedings of the 7th Central European Workshop on Services and their Composition, ZEUS 2015. pp. 51–54. CEUR Workshop Proceedings (May 2015)
18. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall PTR (2000)

19. Li, F., Vögler, M., Claeßens, M., Dustdar, S.: Towards automated iot application deployment by a cloud-based approach. In: 6th International Conference on Service-Oriented Computing and Applications (SOCA). pp. 61–68 (Dec 2013)
20. Mietzner, R., Leymann, F.: Towards Provisioning the Cloud: On the Usage of Multi-Granularity Flows and Services to Realize a Unified Provisioning Infrastructure for SaaS Applications. In: Proceedings of the International Congress on Services (SERVICES 2008). pp. 3–10. IEEE (Jul 2008)
21. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0. Organization for the Advancement of Structured Information Standards (OASIS) (2013)
22. OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0. Organization for the Advancement of Structured Information Standards (OASIS) (2013)
23. OASIS: Message Queuing Telemetry Transport (MQTT) Version 3.1.1. Organization for the Advancement of Structured Information Standards (OASIS) (2014)
24. OASIS: TOSCA Simple Profile in YAML Version 1.0. Organization for the Advancement of Structured Information Standards (OASIS) (2015)
25. OMG: Business Process Model and Notation (BPMN) Version 2.0. Object Management Group (OMG) (2011)
26. Opscode, Inc.: Chef Official Site, `http://www.opscode.com/chef`
27. Puppet Labs: Puppet Official Site, `http://puppetlabs.com/puppet/what-is-puppet`
28. da Silva, A.C.F., Breitenbücher, U., Hirmer, P., Képes, K., Kopp, O., Leymann, F., Mitschang, B., Steinke, R.: Internet of Things Out of the Box: Using TOSCA for Automating the Deployment of IoT Environments. In: Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER). pp. 358–367. SciTePress Digital Library (Jun 2017)
29. da Silva, A.C.F., Breitenbücher, U., Képes, K., Kopp, O., Leymann, F.: OpenTOSCA for IoT: Automating the Deployment of IoT Applications based on the Mosquitto Message Broker. In: Proceedings of the 6th International Conference on the Internet of Things (IoT). pp. 181–182. ACM (Nov 2016)
30. Weerasiri, D., Benatallah, B., Barukh, M.: Process-driven Configuration of Federated Cloud Resources. In: Database Systems for Advanced Applications, pp. 334–350. Springer (2015)
31. Wettinger, J., Binz, T., Breitenbücher, U., Kopp, O., Leymann, F., Zimmermann, M.: Unified Invocation of Scripts and Services for Provisioning, Deployment, and Management of Cloud Applications Based on TOSCA. In: Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER 2014). pp. 559–568. SciTePress (Apr 2014)

All links were last followed on 28.07.2017.

Requirements and Enforcement Points for Policies in Industrial Data Sharing Scenarios

Michael Falkenthal[1], Felix W. Baumann[2], Gerd Grünert[2], Sebastian Hudert[2], Frank Leymann[1], and Michael Zimmermann[1]

[1] University of Stuttgart, Institute of Architecture of Application Systems Universitätsstr. 38, 70569 Stuttgart, Germany `[firstname].[lastname]@iaas.uni-stuttgart.de`

[2] TWT GmbH Science & Innovation Industriestr. 6, 70565 Stuttgart, Germany `[firstname].[lastname]@twt-gmbh.de`

Abstract. Industry 4.0 endeavours often integrate and analyze a multitude of data, such as data about machinery, production steps, and environmental conditions, in order to optimize manufacturing processes. Thereby, they aim to reveal information hidden in formerly isolated data silos via holistic analytics approaches. However, the integration of such data silos is often accompanied by challenges according legal regulations, organizational obstructions, and technical implementations, among others. Therefore, in this work we present a list of key challenges, which have to be commonly overcome in integration projects dealing with essential data from production processes. They can be used as a check list to address recurring challenges in future Industry 4.0 projects. Finally, we identify several plug-points in an abstract integration architecture, which have to be considered in concrete projects at hand to enforced the requirements.

Keywords: Requirements, Policies, Data Aggregation, Industrial Data, Data Integration, Industry 4.0

1 Introduction

The 4th industrial revolution, respectively known as Industry 4.0 [13], is facilitated by developments in the fields of data analytics, which evolve in this context to a new research field of so-called smart services [2]. Besides the availability of easily accessible cloud computing resources and advances in the miniaturization of sensors and Internet of Things (IoT) devices, the need for smarter factories and dynamic production processes are main drivers for manufacturing companies to foster new analytics approaches targeting the automated optimization of production lines. Thereby, different technologies, e.g., IoT, shall be leveraged in order to capture data about manufacturing environments and supporting processes in a very fine-grained manner. For instance, sensors are brought out into factories, which allow to enrich already present monitoring data about production processes with additional environmental parameters, such as temperature or humidity. The major goal of such endeavours is to identify previously hidden auxiliary conditions influencing production processes in a specific manufacturing facility. In ideal cases, conclusions about changing degrees of incorrectly produced parts can be deduced and machinery can be automatically adjusted appropriately to compensate such changing parameters. Another example is to integrate data silos from different production units to enable holistic analyses inferring new insights and optimization potentialities of production processes to make factories more adaptable to drastically increasing variations in the product portfolio.

However, while such developments promise to align whole industries for the upcoming era of dynamically and rapidly changing productions, actual Industry 4.0 projects are typically faced with different kinds of challenges that have to be considered in order to attain success. On the one hand, the integration of many different data sources and the analysis of big amounts of data both require immense expertise in terms of the development of analytics algorithms and the operation of integration middleware. On the other hand, the technical perspective of such an data analysis project is commonly not the most substantive one if it is about to prosper. Further requirements regarding law constraints, organizational obstacles, or qualities and semantics of analysis results have to be managed, which are often unapparent in the course of an Industry 4.0 project.

Therefore, we present findings by the research project *SePiA.Pro* [1, 16], which is located in the context of Industry 4.0. We describe and structure ascertained requirements concerning the integration and processing of business-critical data about manufacturing processes and production steps. Among the discussion of these requirements we also locate different enforcement points in an abstract and, thus, generic integration architecture that can be considered as technical hooks allowing to ensure compliance according the identified requirements.

The remainder of this paper is structured as following: we motivate the challenges of Industry 4.0 projects in more detail and give deeper background information in Section 2. We explain and structure key challenges in this context in Section 3 and discuss possible enforcement points to assure the compliant usage and security of business critical data in Section 4. Related work that supports and extends the presented findings in this work is discussed in Section 5. We conclude this work in Section 6 by summing up results that are expected from this work and the project and identify relevant future work.

2 Motivation and Background

Typical Industry 4.0 projects have to deal with the integration of formerly disconnected data silos, be it because of different production units, departments, or even legal entities. All these data sources are from the production or supply chain. Such a scenario is depicted in Figure 1, which illustrates the isolation of two production units on the right. In many companies, hierarchical organizations or the distribution of manufacturing among different production facilities lead to the emergence of data silos, i.e., technically disconnected databases. Each data

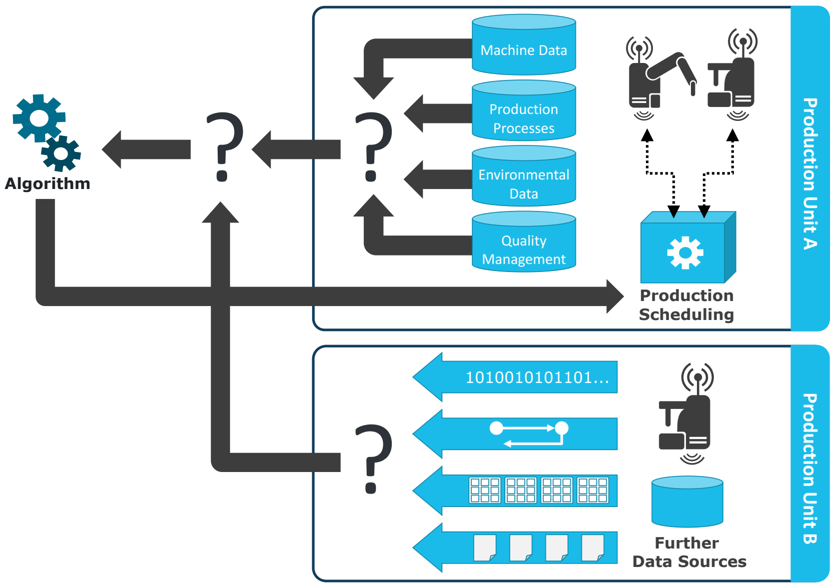

Fig. 1. Typical Industry 4.0 integration scenario: (i) different types of data are integrated as exemplarily depicted in *Production Unit A*, while (ii) different technical characteristics, such as data streams, data batches, and transmission via file are exemplarily illustrated in *Production Unit B*. Further, data from the different production units is further integrated and exposed to an external analytics *smart service*.

silo typically comprises different types of data, such as exemplarily illustrated in *Production Unit A* by machine-data, data about production steps and processes, data about environmental conditions, and data from quality management monitoring production lines. The combination of this data can enable the creation of additional knowledge that can benefit the data owner. Specific about this setup is that the different data types are usually disconnected even if they are technically available in a production unit. This is often due to different backend systems, diverging data formats and semantics, as well as missing adapters and integration middleware. Besides these technical impediments also organizational structures and responsibilities can cause such inhomogeneous system landscapes.

The situation is typically even more complex, since besides different types of data, also the technical characteristics of how it is captured and provided can differ greatly. This circumstance is illustrated in *Production Unit B*, where data is (i) available via *data streams*, i.e., continuous streams of bits and bytes, (ii) a *request – response* model, where an inquiring system sends specific requests for data, (iii) *batches of data*, such as results from executed sets of pooled SQL queries, or as (iv) a number of *file* exports. Therefore, also the heterogeneity if these characteristics adds to the complexity of integrating data from different systems. Thus, from a technological point of view, the integration of these data is a non-trivial task, as indicated by the question marks in the depicted production units in Figure 1. The integration is non-trivial because immense expertise about the source systems, as well as suitable integration middleware is required.

However, once data shall be integrated among different departments, production units, or even legal entities there often arise also obstacles in terms of legal issues and organizational caveats, which have to be handled by integration projects. Thereby, the integration of data from different legal units might cause violations of country-specific law, e.g., laws and acts intending to protect the economy from monopolistic consolidations of companies, consumer protection laws or laws against insider trading. This applies especially to integration scenarios, where competitors strive to cooperate in non-business-critical areas. Also, if data shall be integrated just internally, different departments or organizational units might inhibit the integration of their data, since analyzing data more holistically often allows for more transparencies and comparisons with other units, thus fueling rivalry. All these non-technical issues have to be clarified in order to provide proper boundary conditions for developing and applying algorithms to analyze connected data holistically, which is illustrated by the third question mark where inputs from the two production units are joined.

Based on this motivating scenario, a list of key challenges and requirements is described in Section 3. In combination with identified policy enforcement points in Section 4, they can be used as a starting point to identify, refine, and address such obstacles in future projects working in the context of Industry 4.0.

3 Requirements for Industrial Data Sharing Platforms and Policies

In this section, requirements for integration systems and data protecting policies, respectively, are described and categorized. The requirements can be classified into legal requirements, which are mainly dependent upon the geographic or legal location in which the data acquisition and processing is performed. The legal requirements are not entirely harmonized among different countries, making it important to involve legal experts early on in such a project.

Organizational requirements are usually defined and enforced by the company or companies employing such a smart service project. There is an overlap between organizational and regulatory requirements with regulatory requirements influencing and creating organizational requirements. Organizational requirements can, furthermore, be different within one single business entity, e.g., in a company which is operating in different countries. These requirements can further depend upon the customers affected by such smart services, e.g., the requirements can differ for corporate, private or governmental customers.

The third category are technical or structural requirements, which mainly stem from the underlying technical landscape. The fourth category comprises of the logical or principal requirements, and, therefore, groups those that are enforced to align with certain objectives or goals for which a particular smart service project is created. The requirements are identified in the following sections and described by respective examples and explanations.

3.1 Legal Requirements

In general, legal requirements arise from the country-specific legal situation. Thus, it is important to learn which laws have to be considered for elaborating a compliant data integration solution. This can be important in scheduling Industry 4.0 projects, because law dictates the fundamental frame about which data is allowed to be integrated. Thus, legal restrictions can prohibit to integrate data in order to be analyzed holistically, although there are no technical limitations.

The Data Privacy Act protects individual-related data in Germany. Data related to people is specifically protected to be processes arbitrarily. Transfered to data integration scenarios, e.g., data about bank accounts must not be combined with data about the purchasing behaviour of customers without being explicitly approved by them. Another example affects the processing of data about production processes. In such cases, it is often prohibited, or at least critical, e.g., to combine data about production processes, downtimes of conveyor belts, and staff for calculating the overall efficiency of personnel. Such purposes mostly have to be clarified with and approved by the employee organization of the company.

Sharing data between different legal entities or internal units can lead to legal issues. On the one hand, if data is shared with competitors in the same or in equal business areas, this can violate law against the suppression of competition, i.e., anti-cartel law. On the other hand, integrating data of distinct internal units can violate law and internal compliance policies if, e.g., the resulting integrated dataset and the automated analysis is not consistent with four-eyes-principles. Therefore, overcoming data silos might cause the circumvention of formerly established compliance processes because due to such regulations data must be distributed among separate units with different management responsibilities.

Anti-discriminatory algorithm design must be enforced in case of legal requirements to avoid biased algorithms. This can be of importance when algorithms and analyses involve person-attributable factors such as gender, religion or race. For industrial data scenarios this applies in cases when data about personnel, e.g., from processes, is combined with other data to draw conclusion's about their performance or qualification for specific tasks. Such analysis scenarios then typically have to deal with requirements detailed in *The Data Privacy Act* above.

3.2 Organizational Requirements

Organizational requirements typically stem from responsibilities of management staff, hierarchies, and segmentation of companies into departments, divisions, and units. Besides this, also sociological connections can influence data integrations among different departments. In any case, a integration project can benefit from a sponsor with wide-ranging responsibilities regarding these influencing factors.

New continuous transparencies of business unit data arise by enabling holisitc analyses. This is due to the fact, that formerly isolated data silos are connected and integrated, which can cause suspiciousness at any affected employee. Therefore, the intended transparencies as a result of the analysis of integrated data can, e.g., lead to unpleasant comparisons of the performance of different departments. Thus, people might refuse collaboration with integration projects because they fear being bad in comparison with others.

Data ownership is often dedicated to specific management responsibilities in a company. This implies that there are managers in a company, who must be enabled to enforce rules, i.e., policies about how specific data can be used and processed. However, data integration scenarios typically integrate datasets to allow comprehensive analysis. Thus, it has to be clarified how data policies can be enforced in the integrated dataset and if new data responsibilities are added.

Inter-company analytics scenarios are often not the major aim of Industry 4.0 projects. Nevertheless, abstracting and aggregating data until they no longer contain business critical information can still open up analysis scenarios along with further companies, which can lead to overall results and value adds for all participants. For instance, if companies operating machines share data about their processing environments, environmental conditions, and machine parameters with the machine vendors, this can enable completely new business models. Of course, then data has to be shared in a way that no business critical information, such as information about the produced parts, is captured. However, this can then enable the analysis of overall machine fleets by machine vendors resulting in suggestions about how to optimize the operation of machines.

3.3 Technical or Structural Requirements

The technical or structural requirements involve issues and obstacles, which can occur due to implications based on technological restrictions or implementation-specific difficulties. These restrictions typically have to be managed in the development and implementation phase of an integration project, while requirements as presented in the sections above have to be carefully considered and incorporated.

Different semantics of data from different data sources can lead to immense integration efforts. For instance, machinery from different vendors can be technically integrated based on compatible protocols but usually provide a vendor-specific data model. So, the different data models have to be compared and mapped to each other in order to assure precise semantics of the resulting integrated set of data. Often, a normalized data model has to be derived and additional data transformations have to be introduced for source systems to match with the integrated data model. Such transformations typically lead to more complexity in the overall integration system due to additional processing components but also because of additionally required processing infrastructure.

Different formats, quality of data, and acquisition rates are also obstacles, which have to be managed in order to enable the holisitc analysis of different data sources. Thereby, semantically equal values have to be adjusted, e.g., the fractual part of floating-point numbers. Another problem arises, if data with different accuracies is collected. Some data sources can provide a higher degree of uncertainity with some data than others, which must be reflected for decisions based on them. Finally, sensors and other data sources often provide data by different rates, which has to be considered by normalizing such data streams.

Data policies have to be inseperable from the data to be protected and integration middleware has to enforce them. This assures, especially in the field of industrial data, the required degree of security for business critical data. For instance, if a specific set of data is classified, i.e., it is defined that it must not leave the company, this has to be attached to the data in the form of a data policy, which can be processed and enforced by integration systems [15].

Arbitrary data transfers have to be secured by data policies. Hence, data policies have to be attached to data independently from the communication channel, be it the transfer of data via data streams, batch jobs, or ordinary files.

Policies have to be combinable on all aggregation steps as motivated in Section 2. Integrating data among different departments, business units, or even companies often implies that data is integrated on a cascade of different integration systems. Each integration step can require to initiate aggregations and obfuscation of data in order to enforce attached policies. However, in such scenarios policies also have to be applied in combination, i.e., policy aggregations have to be conceptually possible and must also technically be enforced.

3.4 Logical or Principal Requirements

The requirements presented in this section add general aspects to the above presented ones. They influence the quality of analysis results and the protection of data by adding general properties to be incorporated into integration systems.

Data results must yield a specific format or data-range. Thus, specific expectation checks should be applied to data at the different integration and aggregation steps as presented above. For instance, data input and data output at a particular processing step must conform to specific rules, which have to be defined. This assures that data does not get corrupted during different manipulations.

Enforceability of data policies has to be assured twofold: firstly, policies must be enforceable under specific conditions, such as in accordance with time. So the relevance of a policy can be restricted via time constraints in a way, that it only applies for data with specific time stamps. Secondly, policy checking and enforcement must be automated to assure performance of integration scenarios.

4 Policy Enforcement Points in Industrial Settings

Policies must be enforced in any system. With the proposed and described system a distributed data network is created. By the nature of this distribution, different logical points are possible and reasonable for enforcement operations. By enforcing policies in different locations, different results and implications are manifested. In the current scenario, data is acquired and handled within *DataHubs*, software components that are equipped with control and access logic, thus, managing the access and acquisition of data for the stakeholders. They can be recursively stacked within enterprises and locations as the following example depicts.

A DataHub is placed logically near the data producing machine and unifies the access to this device such that it can be used in the resulting smart service. Another DataHub is placed within one factory building, aggregating data from and unifying access to several downstream DataHubs located at various machines. Furthermore, a DataHub is placed at a business unit that controls various factory buildings and individual machines with associated DataHubs. The location and placement of the DataHubs can be categorized as follows:

– Directly at the data-producing machine. This placement requires knowledge about access control structures, based on employees or groups, which might not be available in this level as the management is usually a few layers up.
– Aggregating data within a physical location, e.g., within a factory building.
– Within a business unit, responsible for data acquisition of a various number of physical and logical locations.

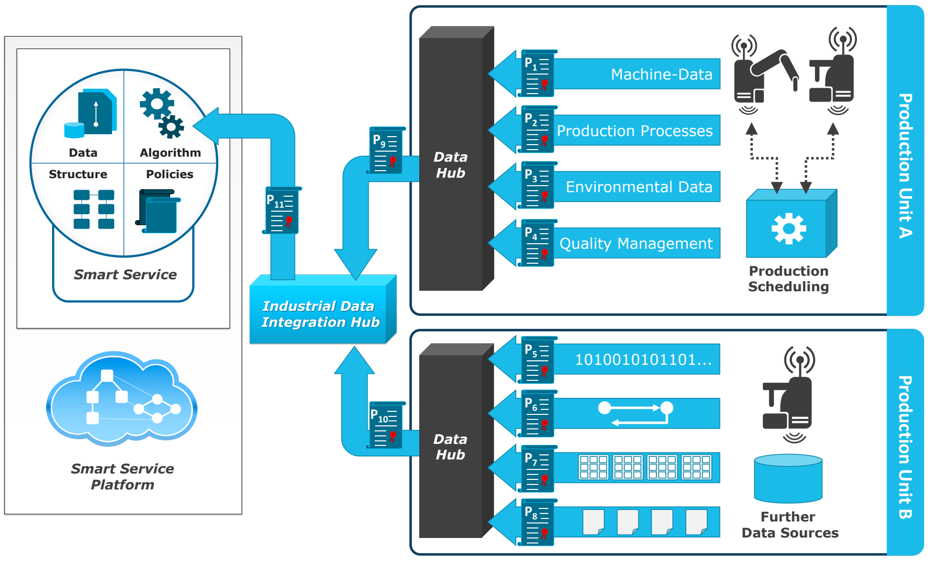

Fig. 2. Data policy enforcement points along different aggregation steps as indicated by eleven policy documents in an abstract integration scenario.

– At company borders, for inter-company connection of data processing tasks. This placement requires, like all others too, detailed knowledge of the connected machinery to enforce certain access restrictions on specific data fields, which might be not available this far from the data production.

All placements vary in the degree of implications caused. To summarize it can be stated, that for the data access enforcement, knowledge about the allowed and disallowed groups and persons, knowledge about company structure, aims, and targets, and, finally, knowledge about the data structure available is required. The different involved parties have varying knowledge about this.

The DataHub is intended to be data-agnostic, meaning that the access is unified, but for the implementation this knowledge is still required. The individual DataHubs do not have any knowledge of the structure further upstream and are only aware of their immediate downstream level. The logic is imbued into the system through the smart services, which have a complete picture of the connectedness of the machines and DataHubs. The implementation scenario of the DataHubs and the machinery within the *SePiA.Pro* project is depicted in Figure 2. In this figure, the *Industrial Data Integration Hub* is also a DataHub component, that is named to indicate its purpose.

The policy enforcement can further be divided by being upstream or being downstream. Upstream denotes the case, where all data access is propagated from the user and the data is acquired accordingly and only in the situation of the transmission of the data back to the user, it is checked if the data is allowed to be procured to the user. The downstream policy checking works by testing the data acquisition or processing requests prior to their execution and acquisition. Aggregation of data might result in new data that the user is not allowed to process which is possible to filter in the upstream processing. In the downstream processing, the user might be wrongfully inhibited to generate data requests that would, eventually, through aggregation or pre-processing, result in data the user would be allowed to process. As an example, a policy could prohibit the usage of personal information, such as which operator of a machine is working with a machine for how long. In this example and with downstream policy application, it is possible to query the machine operating hours and separately shift plans for the respective machine. Both these queries do not yield the forbidden information, so it would be allowed. Further combination of these data would yield the forbidden information but could not be prevented in this case. To prevent this, upstream policy application is required.

The policies, indicated by the scroll symbol and named P_x in Figure 2, can be attached to raw data and processed data of various types. The policies are ensured to be enforced by the DataHub. In the reverse direction, sending instructions towards the data sources, e.g., for the addition of additional sensors or for restructuring of the data, are also possible with the system. This direction is not depicted in the figure. These instructions are also enhanceable with policies as described.

5 Related Work

In the following, related work is discussed, which extends the context of this work. Thereby, we especially point out work presenting details and approaches about the presented challenges and starting points to solve them in particular projects.

According to Weyer et al. [24], one additionally identified and important challenge for the advent and success of smart factories is the standardization of protocols, technologies, and data formats. They identify that production systems, nowadays, are still only vendor-specific ecosystems, which are not driven by open standards. This hinders their interplay and integration to foster automatic control and adjustment of production processes. Thus, they provide *SmartFactory*^{KL} as an exemplary reference for a modular and adaptable production system.

Regarding the communication and connectivity of machinery, devices, and further data sources, Varghese and Tandur [23] describe the key role of wireless communication networks to enable Industry 4.0 systems. They discuss current key challenges in the field of wireless communication and argue how the 5th generation of wireless networks can tackle these. The identified key challenges concerning wireless communication extend the presented technical challenges in this work. Wollschläger et al. [25] further underline these challenges by identifying IoT as the leading technological evolution, which enables the development of smart platforms to access and orchestrate industrial data and devices.

To overcome the technical challenges in terms of the design and implementation of integration architectures as abstractly depicted in this work, there are different approaches presented. General IoT reference architectures to identify main system components are given, such as presented by Guth et al. [12]. The enterprise integration patterns by Hohpe and Woolf [14] provide best practices to design and implement integration systems. Implementations of these patterns are already available via different middleware technologies, such as Apache ActiveMQ [21], Apache Camel [22] or Spring Integration [17]. Further, the cloud computing patterns by Fehling et al. [11] provide knowledge about integration of private and public clouds, which can get necessary in Industry 4.0 endeavours if local processing power is not sufficient to execute analytics algorithms contained in smart services. To specifically deal with architectures and characteristics of IoT-related systems and devices, Reinfurt et al. [18, 19] provide a collection of *Internet of Things Patterns*, which they plan to develop towards a pattern language for IoT. The presented requirements in this work can be mapped to their patterns in order to find proper solution concepts. Finally, to ease and guide design and implementation of IoT systems and integration scenarios and, thus, to efficiently overcome the identified implementation challenges in this work, Falkenthal et al. [5, 4, 6, 9] describe approaches to connect concrete implementations to patters using pattern and solution repositories as introduced by Fehling et al. [10].

Finally, technologies from the domain of cloud computing have been identified to be drivers of the 4th industrial revolution in terms of automating the provisioning and management of analytics stacks [8] and to enable function and data shipping scenarios based on situational conditions [7], such as legal and organizational requirements as identified in this work. Application of cloud technology is also considered essential in manufacturing concepts, such as cloud manufacturing as described in Baumann et al. [3], where the connection of additive manufacturing technology to the Internet is described, thus enabling collaborative work.

6 Conclusion and Future Work

We presented in this work findings from the project SePiA.Pro [1], which investigates the issues and challenges of Industry 4.0 projects in terms of the research program Smart Service World of the federal ministry of economics and energy of Germany. Thereby, we elaborated requirements for protecting industrial data in the context of Industry 4.0 endeavours via requirements or data policies, respectively. Such data policies are means to specify constraints, restrictions or instructions that apply to the data, taking into account aspects such as data accessibility, utilisation, processing, obfuscation, storage or generation. The policies extend common access control rules and restrictions to incorporate concepts such as temporal, logical and organisational triggers. An exemplarily scenario for enabling trust and enforcing implementations was analysed within this work, which can be used as a coarse-grained overview to attach data policies to relevant data sources and plug-points in data integration architectures. The rationale for such an attachment of policies is to secure and protect data from manufacturing environments in standards-based deployment models such as cloud computing. These models can be used to provision smart services and wiring them with arbitrary data sources, such as databases, data aggregation services, industry specific machine to machine or IoT related data streaming endpoints.

In future work, we plan to further investigate, how and through which means, i.e. systems and parties, the identified challenges, requirements, and policies can be enforced at several points in time of the lifecycle of smart services — specifically at modelling time, deployment time and runtime — to overcome the above mentioned obstacles. Based on these investigations we plan to extend the open-source provisioning engine OpenTOSCA to enable the enforcement of data policies in Industry 4.0 deployment scenarios as presented by Falkenthal et al. [8] and also or more general IoT integrations such as presented by [20].

Acknowledgments. This work is partially funded by the project SePiA.Pro (01MD16013F) of the BMWi program Smart Service World.

References

1. Service Plattform for the Inteligently Optimization of Manufacturing Environments, `http://projekt-sepiapro.de/en`, last accessed on 28th July 2017
2. Allmendinger, G., Lombreglia, R.: Four strategies for the age of smart services. Harvard Business Review 83(10), 131 (2005)
3. Baumann, F.W., Eichhoff, J., Roller, D.: Collaborative cloud printing service. In: Cooperative Design, Visualization, and Engineering - 13th International Conference, CDVE 2016, Sydney, NSW, Australia, October 24–27, 2016, Proceedings. pp. 77–85. Springer (Oct 2016)
4. Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F.: Efficient Pattern Application: Validating the Concept of Solution Implementations in Different Domains. International Journal On Advances in Software 7(3&4), 710–726 (Dec 2014)
5. Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F.: From Pattern Languages to Solution Implementations. In: Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS 2014). pp. 12–21. Xpert Publishing Services (May 2014)
6. Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F., Hadjakos, A., Hentschel, F., Schulze, H.: Leveraging Pattern Application via Pattern Refinement. In: Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change (PURPLSOC 2015). epubli (Jun 2015)
7. Falkenthal, M., Breitenbücher, U., Christ, M., Endres, C., Kempa-Liehr, A.W., Leymann, F., Zimmermann, M.: Towards Function and Data Shipping in Manufacturing Environments: How Cloud Technologies leverage the 4th Industrial Revolution. In: Proceedings of the 10th Advanced Summer School on Service Oriented Computing. pp. 16–25. IBM Research Report, IBM Research Report (Sep 2016)
8. Falkenthal, M., Breitenbücher, U., Képes, K., Leymann, F., Zimmermann, M., Christ, M., Neuffer, J., Braun, N., Kempa-Liehr, A.W.: OpenTOSCA for the 4th Industrial Revolution: Automating the Provisioning of Analytics Tools Based on Apache Flink. In: Proceedings of the 6th International Conference on the Internet of Things. pp. 179–180. IoT'16, ACM (2016)
9. Falkenthal, M., Leymann, F.: Easing Pattern Application by Means of Solution Languages. In: Proceedings of the 9th International Conference on Pervasive Patterns and Applications. pp. 58–64. Xpert Publishing Services (2017)
10. Fehling, C., Barzen, J., Falkenthal, M., Leymann, F.: PatternPedia – Collaborative Pattern Identification and Authoring. In: Proceedings of PURPLSOC (Pursuit of Pattern Languages for Societal Change). The Workshop 2014. pp. 252–284 (Aug 2015)
11. Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications. Springer (Jan 2014)
12. Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., Reinfurt, L.: Comparison of iot platform architectures: A field study based on a reference architecture. In: 2016 Cloudification of the Internet of Things. IEEE (2016)
13. Hermann, M., Pentek, T., Otto, B.: Design Principles for Industrie 4.0 Scenarios. In: Proceedings of the 49th Hawaii International Conference on System Sciences (HICSS). pp. 3928–3937 (2016)
14. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley (2004)

15. Pearson, S., Casassa-Mont, M.: Sticky policies: An approach for managing privacy across multiple parties. Computer 44(9), 60–68 (Sept 2011)
16. Pfeil, M., Odefey, U., Ritter, Y., Schuermann, M., Fäßler, V.: Smart services - the smart implementation of Industry 4.0. In: NAFEMS Seminar Simulation von Composites – Bereit für Industrie 4.0? (10 2016), `https://www.researchgate.net/publication/317687064_Smart_services_-_the_smart_implementation_of_Industry_40`, last accessed on 28th July 2017
17. Pivotal Software: Spring Integration, `https://spring.io/spring-integration`, last accessed on 28th July 2017
18. Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., Riegg, A.: Internet of things patterns. In: Proceedings of the 21st European Conference on Pattern Languages of Programs. ACM (2016)
19. Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., Riegg, A.: Internet of things patterns for devices. In: Ninth international Conferences on Pervasive Patterns and Applications. pp. 117–126. Xpert Publishing Services (2017)
20. da Silva, A.C.F., Breitenbücher, U., Hirmer, P., Képes, K., Kopp, O., Leymann, F., Mitschang, B., Steinke, R.: Internet of Things Out of the Box: Using TOSCA for Automating the Deployment of IoT Environments. In: Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER). pp. 358–367. SciTePress Digital Library (Jun 2017)
21. The Apache Software Foundation: Apache ActiveMQ, `https://activemq.apache.org`, last accessed on 28th July 2017
22. The Apache Software Foundation: Apache Camel, `http://camel.apache.org`, last accessed on 28th July 2017
23. Varghese, A., Tandur, D.: Wireless requirements and challenges in industry 4.0. In: 2014 International Conference on Contemporary Computing and Informatics. pp. 634–638 (Nov 2014)
24. Weyer, S., Schmitt, M., Ohmer, M., Gorecky, D.: Towards industry 4.0 - standardization as the crucial challenge for highly modular, multi-vendor production systems. IFAC Symposium on Information Control Problems in Manufacturing-PapersOnLine 48(3), 579 – 584 (2015)
25. Wollschlaeger, M., Sauter, T., Jasperneite, J.: The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. IEEE Industrial Electronics Magazine 11(1), 17–27 (March 2017)

Towards A Reference Architecture for (Big) Data Pipelines

Ekhtiar Syed[1,2], Willem-Jan van den Heuvel[2, 3], and, Pierluigi Casale[1,2]

[1] Philips Lighting Research, High Tech Campus, Eindhoven, The Netherlands
[2] Jheronimus Academy of Data Science (JADS), s'-Hertogenbosch, The Netherlands
[3] Tilburg University, Warandelaan 2, Tilburg, The Netherlands
Primary contact: ekhtiar.syed@{philips.com, gmail.com}

Abstract. Over the past years, (big) data and analytics have made a significant impact in nearly all domain. Organizations have commenced progressively on capitalizing their data assets in larger volumes, and with more varieties and varying velocities than ever before. Data pipelines constitute the de-facto vehicle for transforming these data assets into value. Although there have been many publications in recent years regarding concrete architectures of data pipelines for specific use cases and technologies, a literature on technology and domain independent, high level, and unified reference architecture is absent. In this paper, we first review and analyze multiple industry cases on the development and management of data pipelines. Based on this thorough analysis we propose and partially validate a reference architecture for robust and scalable data pipelines, which is the main contribution of this paper. By defining the functionalities and data flow between the main components of a big data pipeline, we hope to facilitate implementation of a concrete architectures for big data pipelines.

Keywords: (Big) Data Pipeline, (Big) Data Pipeline Reference Architecture

1 Introduction

In 2006 Michael Palmer, a marketing commentator, was the first to compare Data with crude oil; and just like crude oil, value from data needs to be extracted through proper processing [1]. (Big) Data pipelines constitute the refineries of data: they efficiently collect, process, and transform raw data into actual value. These are challenging tasks as the volume, variety, velocity and veracity of data is continuously increasing. In this paper, we make a first step towards the definition of a high-level reference architecture for (big) data pipelines. Therefore, we seek to answer the following three research questions:

1. What are the components of a reference architecture for (big) data pipelines?
2. What are the functionalities of these components?
3. How does the data flow between these components?

We envision that this reference model will provide multiple benefits, both at theoretical and practical level. In primis, a reference architecture facilitates the design of concrete architectures. This also facilitates standardization of concrete architectures, which enables reuse of standard system functionalities and configurations, and resulting shorter time to market and cost reduction [2]. The definition of a true reference architecture is far from a trivial task. We will use a bottom-up approach: starting from industry cases, we will abstract key features, including functional and non-function requirements (desiderata) of current real-life pipelines.

In this work, we will adopt the design science research methodology, using empirical research to gather requirements, with design and validation (again based on empirical studies). Firstly, we have conducted an in-depth literature survey to localize case studies reporting on the development of data pipelines. From the cases, we have carefully selected three case studies that proved to have sufficient objectivity, quality, and depth for further analysis in Section 3. From the case studies, we have subsequently analyzed and abstracted key design desiderata for data pipelines also in Section 3. Based on these desiderata, we then have proposed a generic reference architecture in Section 4. In Section 5, we use the framework provided in [3] to analyze the design and relevance of our reference architecture. Finally, in Section 6 completes this article with conclusions and future work.

2 Reference Architecture Definitions and Design Approach

In the software domain, *reference architecture* is associated with numerous definitions [3]. For our work, we have selected the definition used in [3], where a software reference architecture is defined as a generic architecture for a class of software systems. It is a high-level abstraction *of all the components, and it defines or describes the functionalities and data flow in between each component*. Furthermore, reference architecture lays the foundation for creating concrete architectures.
This paper proposes a reference architecture for big data systems based on the empirically-grounded design framework for reference architecture; which is originally proposed in [3] and [4]. In Section 3.1 three use cases of big data pipelines are studied from literature. The framework proposed in [3] comprises of multi-dimensional space for classification of five different types of reference architectures. The work in [3] also displays that a reference architecture is *congruent* if a reference architecture can be classified as one of the five reference architectures. In this paper, we also examine the proposed reference architecture using the multi-dimensional space and validate its congruency.

3 Study of Real World Data Pipelines

To understand the essential characteristics and design desiderata of data pipelines, in this section we reviewed three case studies from literature. After studying these pipelines, we superimpose these case studies and refine common serving as the natural substrate of the proposed reference architecture.

In [5] authors design a data pipeline to process ‘activities’ of customers for Groupon, an e-commerce company. These activities include web-based interactions such as product view, clicks, and purchases on Groupon’s touchpoints (website, mobile application, and marketing email) from a 100 million+ customer base. To handle such large volumes of data, the data pipeline is designed in two segments or parts. In the first part, the data pipeline accommodates products and customer activity data from separate sources. Product information such as price and locations are collected and stored in Hadoop Distributed File System (HDFS) [6] and various log files containing customer activity information are put, in real-time, to Apache Kafka [7]. Apache Storm [8] consumes these logs from Kafka in real-time, processes it in a canonical format and writes the output into Apache HBase [9]. The second segment joins and processes product and customer data from HDFS and HBase with MapReduce [10] in batch or offline to derive valuable and actionable insights.

In [11], authors proposed a big-data pipeline for carrier payments, a service enabled via PayPal to use mobile as a mean to pay for digital goods. A verification PIN is sent to the mobile phone and the amount is charged at the end of the month along with the mobile phone bill. As with any payment method, dealing with fraudulent transactions is a key challenge. Authors implement a data pipeline to bring relevant data into a platform to enable a data-driven method to tackle fraudsters. Data is pushed from various sources into the data pipeline through a single REST interface, which accepts JSON data with just a simple application reference ID. To handle ingestions at various pace with scalability, this entry point is built as a micro service and the ingested data are pushed into Kafka. Each source is submitted to a separate 'topic' or queue in Kafka. Apache Storm picks up ingested data from Kafka, does simple processing and stores the output in Hive [12].

[13] presents a data pipeline for IoT applications where the authors address the challenge of capturing and processing huge amounts of IoT data in an efficient way. To achieve such efficiency in a data pipeline, a combination of various suitable tools is required. The authors perceive scalability and load balancing as the key requirements in selecting each component for building the IoT (sensor) data pipeline. They suggest Apache Flume [14], Fluentd [15], ZeroMQ [16], RabbitMQ [17] as great for data ingestion. The authors also lists Apache HBase [9], Apache Cassandra [18] and InfluxDB [19] as options for implementing a highly scalable high-speed data store. Furthermore, the literature suggests Apache Flink [20], Apache Storm [8], Apache MapReduce [10], and Apache Spark [21] as data processing framework; and R [22], SciPy [23], NumPy [24], Hive [12] as preferred data analytics toolkit.

Comparative Case-Study Analysis: For all the data pipelines reviewed above, system scalability and real-time capability is of top priority. Furthermore, in [5] and [11], ingesting unstructured data from various sources with a high degree of schema flexibility is a necessity. To ensure system scalability and increase schema flexibility, we see a common trend of implementing a publish/subscribe mechanism, where ingested data is fed into a central queueing system. However, the data pipelines vary on the requirement of processing complexity of ingested data. While [5] requires complex data processing, [11] and [13] requires simple data processing. A summary of the design considerations underpinning the case studies is summarized in Table 1.

Table 1. Data Pipeline Design Considerations

Architectural Patterns	Mentioned In
Ingestion with Schema Flexibility	[5] [11]
Real-Time Capabilities	[5] [11] [13]
Complex Processing	[5]

System Scalability	[5] [11] [13]
Pub/Sub Mechanism	[5] [11] [13]

4 Towards a Reference Architecture

Traditionally, a pipeline is a collection of data processing tasks connected in a series, where the output of one task is the input of the next task [25]. *As we have observed in Section 3, (big) data pipelines in modern-world settings typically consist of multiple dependent tasks leveraging different technologies to meet required design goals or considerations.*

The reference architecture proposed in Figure 1 is grounded on the design desiderata reported in Section 3. The *data collection* (see left hand side in this figure) ingests data from external sources into the data pipeline. Data sources are the end-points from where the data pipelines consume data, and is further categorized as *streaming* or *stationary sources*. Data collection has a streaming layer to accommodate data from streaming sources and a batch layer to gather data from stationary sources. Streaming data sources push data into the data pipeline when new data is available. On the contrary, data collection gather data from stationary sources in a batch or periodic manner. The *data bus,* which is essentially a message queuing system, acts as a buffer for the incoming messages. The data collector is responsible for sorting the data into the right queue. Once the data processor picks up incoming data, it transforms the data and writes the result to a data store or sink.

The data queue enables the data collection and data processing to operate asynchronously and scale independently. Typically processing the data and writing it to an output requires more resource. By implementing a publish/subscriber pattern, multiple data processing instances can consume data from a single point. This simplifies complicated tasks such as governing, routing, and managing data. Furthermore, the data queue enables the data collection to push data with high throughput, without having to worry about the availability of data processing.

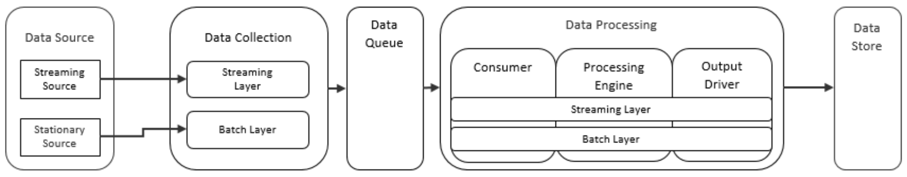

Fig. 1. (Big) Data Pipeline Reference Architecture

The next component of our data pipeline entails the *data processing*. It consists of three sub components: the *consumer*, *processing engine*, and *output driver*. The consumer is responsible for picking up the right set of data from the data bus. Once the right data is retrieved, the processing engine transforms or processes the data. After the data is processed, the result is stored for further consumption. The output driver connects and writes the output of the processing engine to the data store. Like the data collection, the data processing could be configured to operate in a streaming or batch style.

A summary of every component, identifying the functionalities and data flow is provided in the Table 2.

Table 2. (Big) Data Pipeline Reference Architecture Summary

Component	Functionality	Data Flow
Data Source	Provide Data	To Data Collection
Data Collection	Ingest data, simple processing, validation, and sorting the data into the right queue or topic.	To Data Queue
Data Queue	Enable asynchronous exchange of data between data collection and data processing.	To Data Processing
Data Processing	Simple and complex data processing to transformation	To Data Store
Data Store	Store the result produced by the Data Processing	-

5 Reference Architecture Classification and Analysis

We have used the multi-dimensional space provided in [3] to analyze the congruency of our reference architecture. In [3] , congruency is defined as alignment between architectural goals, design context and intended application context, and architecture design and specification of a reference architecture. The value for each dimensions of the multi-dimensional space is provided below in Table 3.

Table 3. Classification of Reference Architecture

Dimension	Value
Why is it defined?	Facilitation
Where will it be used?	Multiple organizations
Who defines it?	Research
When is it defined?	Preliminary
What is described?	Components
How detailed is it described?	Semi detailed
How Concrete is it described?	Abstract elements
How is it represented?	Semi-formal element specifications

As summarized in Table 3, the reference architecture proposed in this article was designed with the goal of facilitation of the design of concrete data pipeline architectures in research groups within Philips Lighting and Jheronimus Academy of Data Science. This is a preliminary reference architecture, which will mature and develop with further research and innovation of this topic. The values specified in the multi-dimensional space aligns with the type 5 reference architecture proposed in [3]. The work in [3] shows that if a multi-dimensional space can be aligned with one of the five types of reference architectures, the reference architecture proposed can be said to be congruent. Therefore, we can conclude that with respect to the purpose we have defined our reference architecture it is congruent.

5 Conclusion and Future Work

In this article, we lay the foundation for a reference architecture for data pipelines with the goal of facilitating concrete data pipeline architectures and further research in this field. The three research questions regarding the reference architecture for (big) data pipelines was answered in Section 4. Furthermore, the congruency of the reference architecture proposed in Section 4 was validated using the multi-dimensional space from [3] in Section 5. However, the reference architecture reported in this article are clearly core research results in nature. We understand that only three data pipelines were studied in design of this reference architecture, and further research is required to validate and improve the proposed reference architecture. Our ambition is to follow up this effort with additional research toward better management of data pipelines.

Furthermore, the amount of data available to companies is exponentially growing and diversifying. There is an increasing and urgent need to have more effective and efficient mechanisms, tools, and techniques to develop and manage data pipelines. Currently, data engineers typically design, develop and maintain data pipelines on a rather ad-hoc, case per case base. What is lacking is a robust, predictable and repeatable approach to develop and maintain data pipelines, promoting reuse, embracing change and resulting in shorter development time, less efforts and more standardized solutions.

We envision to exploit model management operators to further formal underpin and operationalize our approach. In addition, more (tool) support is needed to “instantiate” the reference architecture. We strongly believe that patterns have proven to be an efficient instrument for similar software engineering problems in the past. Our future work will be directed toward developing patterns to cater for a systematic, repeatable and rigorous development (and maintenance) of data pipelines.

References

1. P. Rotella, "Is Data The New Oil," Forbes, 02 April 2012. [Online]. Available: http://www.forbes.com/sites/perryrotella/2012/04/02/is-data-the-new-oil/. [Accessed 04 April 2016].
2. S. M.-F. P. A. F. M. Marques, "Benefits and Drawbacks of Reference Architectures," in *European Conference on Software Architecture*, Montpellier, 2013.
3. P. G. D. G. Samuil Angelov, "A framework for analysis and design of software reference architectures," *Information and Software Technology,* vol. 54, no. 4, pp. Pages 417-431, April 2012.
4. M. a. A. P. Galster, "Empirically-grounded Reference Architectures: A Proposal," in *Proceedings of the joint ACM SIGSOFT conference -- QoSA and ACM SIGSOFT symposium -- ISARCS on Quality of software architectures -- QoSA and architecting critical systems -- ISARCS*, Boulder, Colorado, USA, 2011.
5. V. D. a. N. P. Kang Li, "Big Data Gathering and Mining Pipeline for CRM using Open-source," in *IEEE International Conference on Big Data*, Dalian, 2015.
6. Apache Software Foundation, "Apache Hadoop," [Online]. Available: http://hadoop.apache.org/.
7. Apache Kafka, "Documentation - Apache Kafka," Apache Software Foundation, [Online]. Available: http://kafka.apache.org/documentation.html. [Accessed 6 June 2016].
8. Apache Software Foundation, "Apache Storm," [Online]. Available: http://storm.apache.org/. [Accessed 10 March 2017].
9. The Apache Software Foundation, "Apache HBase," [Online]. Available: https://hbase.apache.org/. [Accessed 10 March 2017].
10. Hadoop MapReduce, "Hadoop MapReduce," The Apache Software Foundation, [Online]. Available: http://hadoop.apache.org/. [Accessed 8 June 2016].
11. M. Mahendru, "PayPal Engineering," PayPal , 2016 November 15. [Online]. Available: https://www.paypal-engineering.com/2016/11/15/carrier-payments-big-data-pipeline-using-apache-storm/. [Accessed 10 March 2017].
12. Apache Hive, "Apache Hive," The Apache Software Foundation, [Online]. Available: https://hive.apache.org/. [Accessed 25 May 2016].
13. A. I. Jussi Ronkainen, "Designing a data management pipeline for pervasive sensor communication systems," in *The 12th International Conference on Mobile Systems and Pervasive Computing*, Belfort, 2015.
14. The Apache Software Foundation, "Apache Flume," [Online]. Available: https://flume.apache.org/. [Accessed 10 March 2017].

15. Cloud Native Computing Foundation (CNCF), "Fluentd - Open Source Data Collector," [Online]. Available: http://www.fluentd.org/. [Accessed 10 March 2017].

16. iMatix Corporation and Contributors, "Distributed Messaging - ZeroMQ," [Online]. Available: http://zeromq.org/. [Accessed 10 March 2017].

17. Pivotal Software, Inc, "RabbitMQ - Messaging That Just Works," [Online]. Available: https://www.rabbitmq.com/. [Accessed 10 March 2017].

18. The Apache Software Foundation, "Apache Cassandra," [Online]. Available: http://cassandra.apache.org/. [Accessed 10 March 2017].

19. InfluxData, "InfluxData," [Online]. Available: https://www.influxdata.com/open-source/. [Accessed 10 March 2017].

20. Apache Flink, "Apache Flink," The Apache Software Foundation, [Online]. Available: https://flink.apache.org/. [Accessed 8 June 2016].

21. Apache Spark, "Apache Spark," The Apache Software Foundation, [Online]. Available: http://spark.apache.org/. [Accessed 8 June 2016].

22. The R Foundation, "R: What is R?," [Online]. Available: https://www.r-project.org/about.html. [Accessed 10 March 2017].

23. S. developers, "SciPy," [Online]. Available: https://www.scipy.org/. [Accessed 10 March 2017].

24. N. developers, "NumPy," [Online]. Available: http://www.numpy.org/. [Accessed 10 March 2017].

25. in *Encyclopedia Of Information Technology*, New-Delhi, Atlantic Publishers & Dist, 2007 , p. 382.

A Configurable Sensor Platform for Cloud-Based IoT Application Scenarios

Gustavo Aragón[1], René Reiners[1] and Zsolt Kemény[2]

[1] Fraunhofer FIT, Schloss Birlinghoven, 53754 Sankt Augustin, Germany
{gustavo.aragon, rene.reiners}@fit.fraunhofer.de
[2] MTA SZTAKI, H-1111 Budapest, Kende u. 13-17, Hungary
zsolt.kemeny@sztaki.mta.hu

Abstract. This work presents the architecture of a sensor platform which is conceived as a sensor and data management system in IoT sensor networks. The main advantage of the platform is its flexibility regarding the design and management of sensor configurations. Sensor configurations are enriched by semantic annotations for fitting the sensor platform to specific application contexts. The sensor platform manages the connection with a variety of heterogeneous sensors, incorporates analytics and data persistence, energy management and allows the transmission of the information to middlewares or cloud computing systems. The sensor platform is designed to be integrated easily in different domains and was already successfully used in food tracking scenarios as well as in the field of ambient assisted living.

Keywords: Sensor Platform, Sensor management, Wireless Sensor Networks, Semantic Annotations, Data Management, Energy Management, Flexible Configuration, Resuability, IoT Application Scenarios

1 Motivation

The numbers of sensors used in the Internet of Things worldwide is constantly growing. According to IHS [1], 30.7 billion devices are expected to be interconnected wirelessly and as integrated devices in the year 2020. Their application fields are enormously diverse such as logistics, food tracking, ambient assisted living, energy, environmental monitoring, emergency cases, etc. The heterogeneity of domains implies the need for a high variety of sensor deployment and access. Usually in the typical application scenarios, a large amount of small sensors is distributed and therefore hard to access for energy consumption and its data. For such kinds of applications, sensors require a battery and low energy consumption, in order to increase the usage time and ensure precise sensing for longer time-periods. Bluetooth and ZigBee constitute two technologies that seem to fill this gap [2][3]. Taking into account these requirements and the high distribution of sensors, reading the information on demand becomes a challenge. The sensor platform presents itself as a solution for acquiring and managing the different sensors' data and making it avaible in convenient ways such as scanning or touchlessly reading-on-the-fly while passing by.

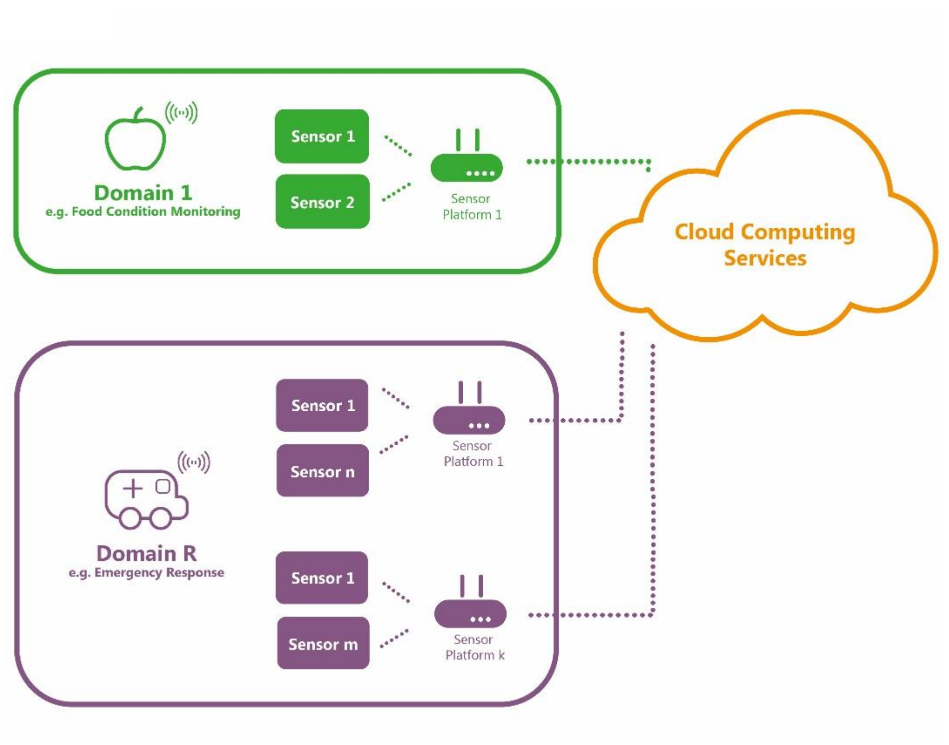

Fig. 1. Sensor platform application

A high number of sensors results in huge amounts of data depending on the number of relevant sensors and their acquisition interval. On the one hand, cloud computing constitutes a powerful development to manage and analyze big amounts of information. On the other hand, application design should consider if all single data points actually need to be transmitted or whether it makes sense to omit or pre-process sets of measurement directly within the sensor platform. The sensor platform allows designing for both alternatives and then establishes a common output interface for transferring the information to cloud services or middlewares (see Fig. 1).

2 Sensor Platform Architecture

The architecture of the sensor platform is composed of modules that incorporate different services and functionalities (see Fig. 2). The high level of abstraction allows the modules to work independently providing flexibility and dynamics to the platform, as it allows to add new sensors or to change the application's behaviour. The modules' responsibilities are divided into Communication Interfaces, Sensor Management Service, Energy Management, Data Management and Data Persistence that are described in the following.

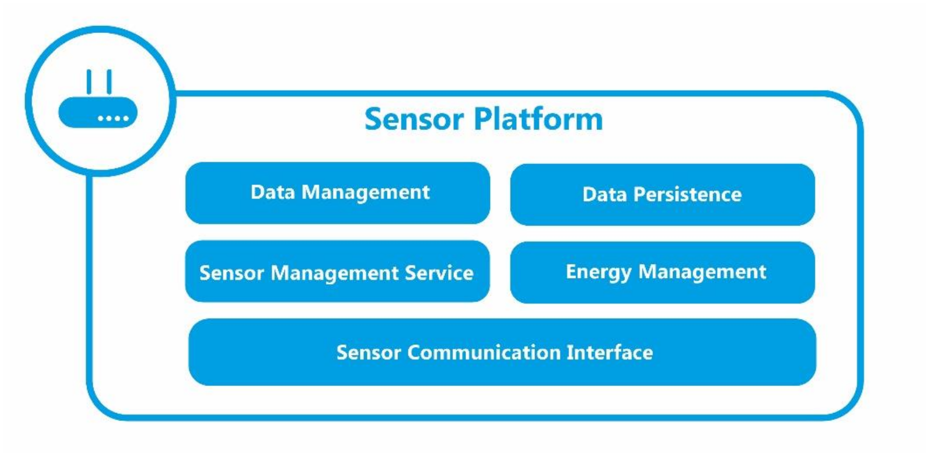

Fig. 2. Sensor Platform Architecture.

The **Communication Interface** provides the abstract connectors for the communication with the sensors and with the cloud services. Individual connectors for combining different protocols and technologies, such as Bluetooth, ZigBee, MQTT, WLAN, are integrated in this module. This gives the possibility of integrating the platform to cloud computing services.

The **Sensor Management Service** manages the configuration of different kinds of wireless sensors within the sensor platform network by entering their information via a web interface. The web interface uses an internal framework to configure the wireless sensors that will connect to the sensor platform. It defines physical characteristics (such as MAC-Address) and other functional characteristics of the sensor (sensor name, type, id, etc.). The framework uses the lightweight JSON format for introducing the information, which is later used by the Data Management Service. This service also establishes and ensures the connection to the set of configured sensors.

The **Energy Management** is responsible for taking measures for immediate power loss or battery drainage of the platform itself (if powered by battery) and the attached sensors. In this case, the configuration of the sensor platform and its related sensors is saved in an internal file. This procedure permits the sensor platform to load the last configuration when the energy supply is restored after a loss of energy.

Data Management collects the data from the connected sensors. Beyond this basic functionalitiy, the component is responsible for optionally preprocessing the data before forwarding it to the cloud or middleware infrastructure. Thus, it is closely related to the specific IoT application. According to this, the algorithms used can vary extremely. Therefore, the platform gives the opportunity to integrate existing frameworks or self-developed code, depending on the system requirements [4]. The flexibility makes the platform suitable even for fog computing applications [5].

Data Persistence is secured in an internal database of the sensor platform, in order to avoid loss of data from the measurements of the sensors. Each instantiated sensor uses an internal sample collector, which is a thread that continuously monitors the availability of the sensors' information. In case a network connection is not available, it internally stores the collected sensor data in a database. As soon as the network connection is restored, the data is forwarded to the cloud service or middleware.

3 Conclusions

The presented Sensor Platform eases the deployment of large-scale networks of heterogeneous sensors by providing data collection hubs that allow for easily collecting, processing and forwarding sensor data. It builds the bridge to cloud computing and middleware solutions by acting as proxy and optionally as fog node. In addition, the platform provides a high level of flexibility in terms of the configuration of attached sensors as well as its data processing capabilities.
The sensor platform was successfully deployed in two different application domains, namely food tracking and ambient assisted living. For the first one, the food's environmental conditions within a package were monitored for dynamic estimation of their individual expiry date and the detection of damages resulting from the packaging process. In the second application scenario wearables were interconnected in order to measure ECG signals and activity of the persons. For both applications, Bluetooth Low Energy was used as wireless technology.

Acknowledgements
Work presented in the paper has been supported by EU Horizon 2020 grants No. 691829 "EXCELL—Actions for Excellence in Smart Cyber- Physical Systems Applications Through Exploitation of Big Data in the Context of Production Control and Logistics"

References

[1] S. Lucero, "IoT platforms : enabling the Internet of Things," no. March, 2016.
[2] M. TOLANI, S. Sharma, R. Singh, K. Shubham, and R. Kumar, "Two-Layer Optimized Railway Monitoring System using Wi-Fi and ZigBee Interfaced Wireless Sensor Network," *IEEE Sens. J.*, vol. 17, no. 7, pp. 1–1, 2017.
[3] J. Hughes, J. Yan, and S. Kenichi, "Development of wireless sensor network using bluetooth low energy (BLE) for construction noise monitoring," *Int. J. Smart Sens. Intell. Syst.*, vol. 8, no. 2, pp. 1379–1405, 2015.
[4] J. Á. Carvajal Soto, M. Jentsch, D. Preuveneers, and E. Ilie-Zudor., "CEML: Mixing and moving complex event processing and machine learning to the edge of the network for IoT applications," *IoT 2016*, pp. 103–110, 2016.
[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," *Proc. first Ed. MCC Work. Mob. cloud Comput.*, pp. 13–16, 2012.

Towards DBCloudBench
A Scenario-Based Database Benchmarking Framework

Andreas Bader and Oliver Kopp

University of Stuttgart, IPVS, Germany

Abstract. Many benchmarks can be used for measuring performance of different types of databases. To automate the process of benchmarking databases, this paper outlines DBCloudBench. It can be used to automatically setup a scenario and perform a benchmark run using a standards-based approach. The databases and benchmarks are stored in "cloud services archives" allowing them to be reused and combined as necessary. Each benchmark is accompanied with an adapter for runing the benchmark on certain database systems while using DBCloudBench.

1 Overview

Choosing a database in a project may be done in different ways such as comparing performance [3], comparing features [1], based on existing knowledge, or by architectural decisions [17]. A performance comparison of different databases is a crucial step when choosing a database. Performance can be compared with different metrics, e. g., completed transaction per time unit, or latency of transactions. Since most databases are grouped in overlapping groups (e. g., time series databases, NoSQL databases, or relational databases), many benchmarks are made for one or more groups of databases or for a specific purpose (e. g., storing time series data). This means that a benchmark can only be executed if the database supports all features that it requires. Beside their targeted database groups, benchmarks are usually different in the creation of their queries (e. g., synthetic creation) and in the measured metrics. As a result, there exist multiple benchmarks for the same group of databases (e. g., MySQL can be benchmarked with TPC-H [15] and YCSB [3]).

When benchmarking performance, there are two possible approaches: 1) Attest and achieve a maximum performance by fine tuning the database and (optionally) the underlying system, 2) Getting insights on performance for choosing a database by benchmarking a set of databases with the same benchmark.

If using the second approach for choosing a database, one or more benchmarks are used to measure one or more metrics of several databases setup in one or more scenarios. That means that a user performing the benchmarking has to know how to setup multiple databases, multiple benchmarks, their metrics, and their required parameters. To chose a database, multiple scenarios may be required to be measured, e. g., different cluster sizes or different replication factors. Additionally, a user has to perform each benchmark run manually, which means that each setup has to done manually for each database, scenario, and benchmark. To ensure good results, the conditions for each run must be comparable [11].

The usage of cloud techniques (e. g., using an Elastic Infrastructure, EI for short) makes it possible to automate the process of performing a benchmark run, which includes setup, benchmarking, retrieving results, and cleanup. Therefore, this work focuses on the second approach, as fine tuning of a system cannot be done in an automated way yet, which makes the first approach impossible to automate. The proposed solution is a benchmark framework for databases being independent of databases and benchmarks that can be used to automatically perform benchmark runs that are setup according to a scenario definition on an EI: DBCloudBench.

A requirement for the benchmarking framework is to support as many EIs as possible and to be as independent from EI-specific code as possible. Additionally, it should be easy to use and to extent for a user. Therefore, a external solution is required that interacts with the EI and keeps DBCloudBench free from EI-specific code. These requirements were derived during the creation of a platform for a new marked role, called Decentralized Market Agent (DMA) [13], which was done in the context of the NEMAR project [9].

By sharing artifacts on how to setup a scenario using repositories, a user requires less knowledge to setup a benchmark run and execute it. A user can pick the required artifacts from the repositories, choose the components he requires for his scenario, and execute it automatically in an EI. This results in less required domain-knowledge, as a user must only know which components he needs, without deeper knowledge in the setup of the chosen database and benchmark. In other words, the aim is that a user that wants to chose a database only specifies the database, benchmark, and scenario he wants to use and the framework executes it automatically. One main part of the solution is to use the “cloud services archive” packaging format of TOSCA and the OpenTOSCA ecosystem [2] for the installation of different benchmarks and databases.

2 Related Work

There exist several benchmarks for measuring cloud performance, the most prominent are CloudBench [12] and CloudCMP [10]. Silva et al. [12] provide an overview over the rest of these types of benchmarks. All of these benchmarks have in common that they try to provide an answer to the question which EIs to use. DBCloudBench, however, has the focus to provide a general framework to meassure database performance.

HammerDB is a tool for automated benchmarking of databases, with a focus on relational databases [4]. As it does not support the automated deployment of databases and its focus is on relational databases, it cannot be used for our approach.

The Transaction Processing Performance Council (TPC) benchmarks consist of several benchmarks for measuring the performance of different business scenarios, e. g., TPC-C [14] that uses an Online Transaction Processing (OLTP) workload that emulates a wholesale company or TPC-H [15] that emulates a decision support scenario, which uses complex queries that are designed to have a long execution time. Most comparable to our approach is TPC-VMS [16], which benchmarks the performance of a database whilst running on Virtual Machines (VMs). To do this, it takes one of the other TPC benchmarks and runs it three times parallel on three identical VMs on an EI. TPC-VMS does not cover the automated deployment of the VMs used in the benchmark, as they are already setup before the benchmark starts.

Kolb and Wirtz [8] identified a similar scenario for users of Platform as a Service (PaaS). These cloud systems often come with heterogeneous environments and interfaces to work with. To assist a user in the decision for the right cloud platform and automate the deployment of user applications, they propose a unified feature description and interface for cloud platforms [7].

The BPEL/BPMN Engine Test System (betsy), a benchmark framework for testing the compliance of open-source BPEL/BPMN engines to the corresponding standards [5], was extended to use virtualization techniques (vbetsy) [6], which results in the use of VMs and their snapshot functionalities to perform benchmark runs. The idea is similar to our approach, but it lacks the support of creating a cluster of VMs depending on a scenario chosen by a user on an EI, which makes it not usable for our approach, even though it uses multiple benchmarks and test candidates (engines) for its tests.

Acknowledgments This work was created in the context of the BMWi-funded project NEMAR (03ET4018).

References

1. Bader, A., Kopp, O., Falkenthal, M.: Survey and Comparison of Open Source Time Series Databases. In: Datenbanksysteme für Business, Technologie und Web (BTW2017) – Workshopband. LNI, Gesellschaft für Informatik e.V. (GI) (2017)
2. Binz, T., Breitenbücher, U., Kopp, O., Leymann, F.: TOSCA: Portable Automated Deployment and Management of Cloud Applications, pp. 527–549. Advanced Web Services, Springer (Jan 2014)
3. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking Cloud Serving Systems with YCSB. In: Proceedings of the 1st ACM Symposium on Cloud Computing. ACM (2010)
4. HammerDB project team: HammerDB (2017), http://www.hammerdb.com
5. Harrer, S., Lenhard, J.: Betsy – A BPEL Engine Test System. Tech. Rep. 90, Otto-Friedrich Universität Bamberg (Jul 2012)
6. Harrer, S., Röck, C., Wirtz, G.: Automated and Isolated Tests for Complex Middleware Products: The Case of BPEL Engines. In: 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops. IEEE (mar 2014)
7. Kolb, S., Röck, C.: Unified Cloud Application Management. In: Proc. World Congress on Services. IEEE (2016)
8. Kolb, S., Wirtz, G.: Towards Application Portability in Platform as a Service. In: Proc. 8th Symp. Service-Oriented System Engineering. IEEE (2014)

9. Kopp, O., Falkenthal, M., Hartmann, N., Leymann, F., Schwarz, H., Thomsen, J.: Towards a Cloud-based Platform Architecture for a Decentralized Market Agent. In: Informatik 2015. LNI, Gesellschaft für Informatik e.V. (GI) (2015)
10. Li, A., Yang, X., Kandula, S., Zhang, M.: CloudCmp: Comparing Public Cloud Providers. In: Proceedings of the 10^{th} annual conference on Internet measurement. ACM (2010)
11. Lilja, D.J.: Measuring Computer Performance: A Practitioner's Guide. Cambridge University Press (2005)
12. Silva, M., Hines, M.R., Gallo, D., Liu, Q., Ryu, K.D., da Silva, D.: CloudBench: Experiment Automation for Cloud Environments. In: 2013 IEEE International Conference on Cloud Engineering (IC2E). IEEE (2013)
13. Thomsen, J., Hartmann, N., Klumpp, F., Erge, T., Falkenthal, M., Kopp, O., Leymann, F., Stando, S., Turek, N., Schlenzig, C., Schwarz, H.: Darstellung des Konzeptes – DMA Decentralised Market Agent – zur Bewältigung zukünftiger Herausforderungen in Verteilnetzen. In: INFORMATIK 2015. LNI, vol. P-246. Gesellschaft für Informatik e.V. (GI) (2015)
14. Transaction Processing Performance Council (TPC): TPC-C Benchmark (2017), http://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-C_V5-11.pdf
15. Transaction Processing Performance Council (TPC): TPC-H Benchmark (2017), http://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpch2.17.1.pdf
16. Transaction Processing Performance Council (TPC): TPC-VMS Benchmark (2017), http://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-VMS-V1.2.0.pdf
17. Zimmermann, O., Wegmann, L., Koziolek, H., Goldschmidt, T.: Architectural Decision Guidance Across Projects – Problem Space Modeling, Decision Backlog Management and Cloud Computing Knowledge. In: 12^{th} Working IEEE/IFIP Conference on Software Architecture. IEEE (2015)

Fine-grained User Support in Data Analytics Processes

Manuel Fritz and Holger Schwarz

University of Stuttgart,
Institute of Parallel and Distributed Systems,
Universitätsstraße 38, 70569 Stuttgart, Germany
`manuel.fritz@ipvs.uni-stuttgart.de`

1 Motivation

In recent years, data analytics gained relevance in many fields such as economy, science or public administration. Applying data analytics on a potentially huge data set aims to find new knowledge, which subsequently can lead to an advance in a specific area, e.g. economic benefits. Process models such as CRISP-DM [2] or KDD [3] guide the analyst through the analysis by defining a sequence of atomic steps. These process models typically range from (1) definition of the analytic goals over (2) selection of the data sets, (3) pre-processing of the data, (4) selection and application of an analytic algorithm to (5) evaluation of the results. They further demand an iterative approach enabling the user to improve incrementally the results of each step and thus improve ultimately the final result of the process. Hence, the analyst requires a strong technical knowledge in each step of the process as well as in the domain in order to gain solid results [1].

The existing process models for analytics are characterized by coarse-grained steps, e.g. step (4) has to be executed completely before step (5) can be initiated. This atomicity leads to a cognitive gap which hinders the discovery of knowledge. In times of big data, where terabytes of data are analyzed, this gap expands even further due to the long run time of established methods. To improve the ability to discover knowledge in a faster way, existing steps of processing models need further analysis. Research has to unveil how they can be drilled down effectively and how they can fade into each other. The result would be a fine-grained approach generating intermediate results, which enable the analyst to interact with the system more closely, thus enhancing the interaction significantly.

Concluding, the main issues of todays data analytics approaches are: (a) the rigidity of atomic steps in process models revealing the quality of the overall result eventually at the very last step, (b) the architecture to support a fine-grained analysis and (c) the necessity of domain knowledge in specific areas to navigate through an analytics process and to validate the quality of the result.

This research heavily contributes to the novel subject of "Human-in-the-Loop Data Analytics". More and more analysts with different technical and domain-specific knowledge will benefit from this research since it enables a broad range of complex analyses to be performed in a detailed manner with good performance. This directly leads to comprehensible, trustful and solid results.

2 Research Questions

The application of an analytic algorithm during an analytics process is the focus of this research. The desideratum addresses a method enabling the user to interact with a system in a fine-grained manner to obtain relevant findings from existing data sets. Applying this method should result in shorter iteration steps and therefore offer the user more options to interact with the system; suitable software solutions must support this (i.e. storage of data and execution of the detailed process). This research currently addresses four pivotal research questions:

RQ1: Which options in the analytic process need a refinement in terms of interaction? To answer this RQ, it needs to be reflected how data mining approaches and algorithms are executed and at which iteration the user should interact with the system. Furthermore, the way of interaction with a system, e.g. selection of data sets, algorithms and parameters, needs to be investigated.

RQ2: For each of the exposed options from RQ1, what kind of system support has to be considered? The possibilities range from a manual support, over a semi-automatic support to an automatic support through an analytics process. Research has to reveal which option at which step bears the most viable solution.

RQ3: What is the underlying architecture to support the user within a finegraind analytics approach? The research focuses on existing reference architectures from Data Warehousing, general data processing architectures and specific architectures of existing analytics tools. The result clarifies which concepts need to be further extended in order to include the analyst more tightly to a detailed analytics process with a higher possibility of interaction.

RQ4: How does an execution environment implement such an architecture from RQ3? Processing environments offer a set of features and characteristics to analyze data. It may be possible, that a specific feature can only be executed reasonably in e.g. Apache Spark. Further research has to reveal how these properties can be combined and extended to gain advantage of different processing engines.

The poster exhibits first insights on recommendations for data mining algorithms and their parameters (see RQ1). Partitioning methods, such as binary space partitioning, provide a good run time behavior and divide data sets into distinct areas. The properties of those areas, such as the density or the distance to other areas, provide valuable information for further research.

Acknowledgement. This research was partially funded by the Ministry of Science of Baden-Württemberg, Germany, for the Doctoral Program 'Services Computing'.

References

1. Abadi, D., Agrawal, R., Ailamaki, A., Balazinska, M., Bernstein, P.A., Carey, M.J., Chaudhuri, S., Dean, J., Doan, A., Franklin, M.J., Gehrke, J., Haas, L.M., Halevy, A.Y., Hellerstein, J.M., Ioannidis, Y.E., Jagadish, H.V., Kossmann, D., Madden, S., Mehrotra, S., Milo, T., Naughton, J.F., Ramakrishnan, R., Markl, V., Olston, C., Ooi, B.C., Re, C., Suciu, D., Stonebraker, M., Walter, T., Widom, J.: Beckman Report on Database. Communications of the ACM 59(2), 92–99 (2016)
2. Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., Wirth, R.: CRISP-DM 1.0. CRISP-DM Consortium p. 76 (2000)
3. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: The KDD process for extracting useful knowledge from volumes of data. Communications of the ACM 39(11), 27–34 (1996)

Fine-grained Elasticity Control Framework for the Cloud

Georgiou Zacharias

Department of Computer Science, University of Cyprus
zgeorg03@cs.ucy.ac.cy

Introduction. Cloud computing offers organizations the opportunity to improve the efficiency of their operations and reducing their costs. The possibility of acquiring resources in a dynamic way is an essential characteristic of the cloud, however deciding the optimal amount of resources, in time, to achieve and offer cloud elasticity is not a trivial task. To date, many algorithms have been proposed and several control services that employ elastic scaling exist. For example, AWS Auto-Scaling is one of the first scaling services offered to cloud developers and is still one of the most popular. AWS Auto-Scaling employs simple metric violation rules that do not require fine-grain cloud application topology modelling to regulate infrastructural resources and schedule scaling actions. In general, elasticity control is provided at the deployment phase and is based on horizontal scaling rules applied to low-level metric rule violations (e.g., if CPUusage $>$ 75% ADD VM).

Challenges. The above approach endeavours a number of risks due to: (i) the inability of novice users to determine optimal thresholds; (ii) "ping-pong" effects where sudden and short-lived (e.g., few seconds) spikes on highly sensitive metrics (e.g., CPU usage) resort to resources being provisioned and de-provisioned rapidly, but most importantly they are billed although real demand does not exist [1]; and (iii) fail to acknowledge "cold-start" effects where the time required for an instance to be bootstrapped is not limited to only VM boot-time reported by the cloud provider. Instead, it includes service bootstrapping and delays imposed due to data movement while "joining" a cluster of other instances [2]. To better understand, consider a scenario in which a cluster of VMs is running a database service(Cassandra, CouchDB). When there is an increase on the workload, a scaling action is triggered and a new VM joins the cluster. The cluster now needs to replicate and transfer the data among the instances, leading to a momentary increase on the average resource usage. This increase is wrongly interpreted by the elastic controller, as a need for adding a new VM in the cluster. When the data transfer is completed, the resource usage is decreased to a level that an action to remove the extra VM is triggered and rebalance starts again, leading the system to an endless cycle between under and over provisioned state. Currently, most of the cloud providers offer the user with a configurable parameter known as cool down period, where the system doesnt allow any other scaling actions to happen during that period. A choice of a long cooldown period, gives enough time to the system to prepare its resources and overcome this problem, however it leads to long over-provisioned periods, increasing the cost.

Proposed Solution. To accommodate these challenges resulting in sub-optimal elasticity control, we aim for the development of a framework capable of supporting fine-grained elasticity control over the cloud environment. We aspire to develop a framework, able to provide with a certain confidence, the novice user with an accurate tuning of such parameters that minimize the periods in which the system is in under or over provisioned state to eliminate these problems without the excessive usage of the resources. This can be achieved by monitoring the resource usage of each instance running the user's application and based on the statistical properties of those metrics, identify the periods of time that determine the state of the system, to make the right decisions for provisioning and de-provisioning. Specifically, we are planning to exploit the capabilities of time series analysis, phase detection methods used in ADMin [3] and supervised learning mechanisms to discover patterns of the application. Currently, we are conducting an experiment based on the scenario described above, in which we demonstrate the aforementioned issues. We identify that adding or removing a VM, can lead to a variable amount of time for the system to prepare its resources, due to the unpredictable time it takes for data movement to complete, making impossible the pre-configuration of cooldown periods. However, by observing the patterns and the statistical properties of the resource metrics, we advocate that the actual period of time that the system needs to prepare its resources, can be detected in an automatic way, resulting to a better and accurate elasticity control.

References

1. Georgiana Copil, Hong-Linh Truong, Daniel Moldovan, Schahram Dustdar, Demetris Trihinas, George Pallis, and Marios D Dikaiakos. Evaluating cloud service elasticity behavior. *International Journal of Cooperative Information Systems*, 24(03):1541002, 2015.
2. Alessio Gambi, Daniel Moldovan, Georgiana Copil, Hong-Linh Truong, and Schahram Dustdar. On estimating actuation delays in elastic computing systems. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2013 ICSE Workshop on*, pages 33–42. IEEE, 2013.
3. Demetris Trihinas, George Pallis, and Marios Dikaiakos. Adaptive monitoring dissemination for the internet of things. In *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications (INFOCOM 2017)*, IEEE INFOCOM 2017, Atlanta, USA, May 2017.