Contextual Understanding of Microservice Architecture: Current and Future Directions

Tomas Cerny
Computer Science
Baylor University
Waco, TX, USA
tomas_cerny@baylor.edu

Michael J. Donahoo
Computer Science
Baylor University
Waco, TX, USA
jeff_donahoo@baylor.edu

Michal Trnka
Computer Science, FEE
Czech Technical University
Prague, Czech Republic
trnkami1@fel.cvut.cz

ABSTRACT

Current industry trends in enterprise architectures indicate movement from Service-Oriented Architecture (SOA) to Microservices. By understanding the key differences between these two approaches and their features, we can design a more effective Microservice architecture by avoiding SOA pitfalls. To do this, we must know why this shift is happening and how key SOA functionality is addressed by key features of the Microservice-based system. Unfortunately, Microservices do not address all SOA shortcomings. In addition, Microservices introduce new challenges. This work provides a detailed analysis of the differences between these two architectures and their features. Next, we describe both research and industry perspectives on the strengths and weaknesses of both architectural directions. Finally, we perform a systematic mapping study related to Microservice research, identifying interest and challenges in multiple categories from a range of recent research.

Keywords

SOA; Microservices; Architectures; Self-contained Systems; Systematic mapping study; Survey

CCS Concepts

•Information systems → Web services; •Applied computing → Enterprise architectures; Service-oriented architectures; •Computer systems organization → *Distributed architectures;*

1. INTRODUCTION

Over the last decades, industry demands have pushed software design and architectures in various directions. The ever-growing complexity of enterprise applications, along with change and evolution management ushered in the rise of architectures such as Common Object Request Broker Architecture (CORBA), Java RMI, and Enterprise Service Bus. Service-Oriented Architecture (SOA), became the answer to multiple industrial demands for large enterprises, replacing its predecessors; however, Microservice Architecture (µService) appears poised to replace SOA as the dominant industry architecture.

Both SOA and µServices suggest decomposition of systems into services available over a network and integratable across heterogeneous platforms. In both approaches, services cooperate to provide functionality for the overall system and thus share the same goal; however, the path to achieving the goal is different. SOA focuses on design of system decomposition into simple services, emphasizing service integration with smart routing mechanisms for the entire company's IT. The smart routing mechanism provides a global governance or so-called centralized management and is capable of enforcing business processes on top of services, message processing, and service monitoring / control. SOA services are uncoupled, reacting without knowing the event trigger. A new service can be easily integrated by reacting to such event. For instance, one service can write an invoice and another can initiate delivery. A new logging system can just listen to events, without impacting other services.

µServices, on the contrary, suggest decomposition preferring smart services while considering simple routing mechanisms [12], without the global governance notable in SOA. This naturally leads to higher service autonomy and decoupling, since services do not need to agree on contracts on the global level. However, services become responsible for business processes management as well as for interaction with other services.

There are, however, other perspectives to consider. SOA's difficulty comes with the complex stack of web service protocols necessary for transactions, security, etc., spanning through all the interoperable services. Moreover, since SOA enables building business processes on top of the services on the integration level, it brings flexibility to change the processes, but at the same time binds all services to a single general context. As a consequence, service contracts expressing the service operation expose its data types leading into dependencies regarding deployments [115], in an extreme case leading into one large monolith deploy.

In µServices, it is possible to involve light and heterogeneous protocols for service interaction. Each service only maintains its context and its own perspective over particular data, possibly leading into duplicities across services. If deployment dependencies exist among services, they are on a much lower scale since no general context and no centralized governance exists. The primary goal of µService is to enable independent service deployments and evolution.

The above features lead to multiple consequences. For instance, it is fairly easy to selectively deploy overloaded µSer-

vice in order to scale it; however, it is not easy in SOA [116]. The SOA integration mechanism and centralized governance predetermine a bottleneck when the system needs to scale up. When a scaling issue arises in some SOA feature, it is hard to determine where the bottleneck is and whether it is the service itself, the integration, or in a shared database. Self-contained µServices are more efficient when it comes to elasticity, scalability, automated, and continuous deploy with fast demand response. The above characteristics make µServices more cloud-friendly [62].

The price of such flexibility is that µServices must restate and redefine data definitions or even business rules across services, introducing replications in databases, lacking a centralized view on the overall system processing, rules, constraints, etc.

Industry seems to be in the shift towards µServices, leaving SOA behind. However, µServices are not a superset of SOA and many of its challenges do not exist in SOA. Various interpretations of these architectures [37] put part of the community on the side considering µServices to be a subset of SOA, although many others [93] see them as distinct architectures.

In this paper, we aim to describe the differences between SOA and µServices so that reader gets a clear picture what to expect from one or the other. We also point out strengths and weaknesses of each approach. Moreover, we each approach to disambiguate terminology and give the reader a solid understanding of benefits of the particular approach. Since µServices seem to be the future direction for the industry, we emphasize our focus on challenges this architecture has to face. Section 2 provides the background. The next two sections introduce SOA and µServices in details. Section 5 provides an example that enlightens the differences. The architecture comparison is the subject of Section 6. Open research challenges in service integration are discusses in Section 7. Section 8 provides a mapping study on µServices introducing challenges addressed in existing work from over 100 papers. The last section concludes the paper.

2. BACKGROUND

SOA and µServices are two major architectures that are being used for decomposing systems into services. The question is how to coordinate services to achieve particular use cases. In general, there are two well-accepted approaches: centrally orchestrated and independent or distributed. Centrally orchestrated approaches are the common pattern for SOA, and the decentralized is the dominant pattern for µServices. In this section, we provide definitions of terms that help us to compare and contrast these two architectures.

Service is a reusable software functionality usable by various clients for different purposes enforcing control rules. It implements a particular element of the domain, defines its interface, and can be used independently over the network. While involving well-known interfaces and communication protocols, it brings platform-independence. In SOA, services are registered in a directory or registry to easily locate them. In order to reduce coupling, services are composed to produce an outcome.

The interaction patterns for centralization and decentralization are called *orchestration* and *choreography*, respectively. These indicate how services collaborate, how the sequence of activities look like, or how the business process is built. Service orchestration expects a centralized business process, coordinating activities over different services and combining the outcomes. Fig. 1 depicts a service orchestration.

Choreography does not have a centralized element for service composition. Service choreography describes message exchange and rules of interactions as well as agreements among interacting services. The control logic is not in a single location. Fig. 2. depicts service choreography.

When involving orchestration through a mediation layer, we often introduce a *canonical data model*. In such a mode, various system parties agree or standardize their data models on the business objects they exchange. However, often [115] the entire system ends up with having just one kind of business object. For instance, there is a single Person, Order, Entry, Invoice, etc., with matching attributes and associations, since everyone agrees on them. It is easy to introduce such a model with orchestration; however, later changes to the model are very difficult since all parties have to agree on them and the individual system has limits on evolution.

An alternative approach arising from Domain-Driven development [108] is called *Bounded context*. Here each service aims to operate with particular business objects in a specific context, and thus it may make sense for some service to consider certain attributes, and ignore others. For instance, we may consider a User's address to process shipments and ignore others; however, for price calculations, it is sufficient to only have User customer-rank. Thus a large model is divided into small contexts, allowing the modeling of business objects differently based on particular needs. Not all services have the same needs and thus should have the independence to design their needs.

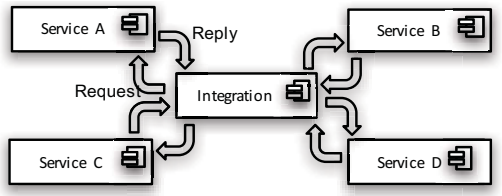


Figure 1: Service orchestration.

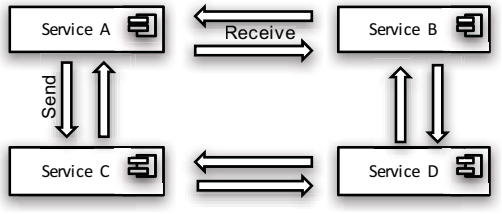


Figure 2: Service choreography.

3. SERVICE-ORIENTED ARCHITECTURE

The main reason for a software architect to use SOA or µServices is to modularize a system into services. SOA, however, requires significant upfront commitments, since the entire company IT must distribute into separate services. It is fairly easy to introduce a new SOA service; one can take a legacy application and define a new network accessible interface for it. A more advanced design divides an application onto multiple services, opening for broader service reuse and service orchestration. The challenging task when introducing SOA is the setup of centralized governance, the component responsible integration of services and their communication. Usually, the solution for integration is Enterprise Service Bus (ESB) that forms the backbone for SOA. As mentioned earlier, it enables orchestration; moreover, services can interact through messages or events where the trigger is unknown while multiple services react on the particular event. In addition, business processes may be defined at the integration level, allowing flexible reconfiguration. However, this solution inclines toward the introduction of a canonical data model. The key point here is that the integration platform is smart, but also complex. SOA is sometimes referred as "simple services and smart pipes" [62] for the this reason. On top of the system is usually a separate component of the user interface, such as a portal or a dedicated web system. Fig. 3a shows a sample SOA deployment with two systems, A and B, exposing services for the integration platform above them. The user interfaces part then communicates with the services through the integration component.

The main advantages of SOA manifest when enough services are available. The business processes implement service orchestration with control over the company processes. It becomes easy to compose services, introduce alternative ways to deal with processes, and build new functionality. The services can be even open to third-parties. However, the layout is usually that various system parts (such as A and B in Fig. 3a) are maintained by different teams. Separate teams usually exist for the integration component or user interface. When changing a particular service that introduces an interface modification, the update most likely promotes to the integration level, as well as to the user interface demanding redeploy of multiple components. In such a manner, SOA deployment happens as a monolith involving multiple services; one defected service may prevent the entire application deployment with a complex rollback.

The situation becomes worse since processes span across services dedicated to different teams, exacerbating communication overhead and requiring coherence in the development and deploy. On top of this, companies tend to have a centralized administration unit managing all changes in the SOA service infrastructure, which leads to conservatism and limitation on individual service evolution.

One of the main issues in SOA is system versioning since we do not know the service users. There are even cases when a company maintains over 20 different versions of the same service with a slightly modified interface to accept different data [23]. This significantly impacts the operations involvement demanding monitoring and maintenance.

According to Red Hat [21], SOA community considers the transition to µServices because the common SOA practice ties services to complex protocols stacks, such as SOAP, a protocol for web service communication, and WSDL, to describe a service [54]. While this is not an SOA requirement, in practical usage it degrades to solely SOAP and web services.

To summarize, SOA makes it easy to change business processes, although, changing a service requires deploy of the component providing the service. This may cascade to the whole SOA monolith, not to mention the need to reflect the changes in the user interface. The integration platform is usually very complex when it comes to the first deployment, and since it is the centralizing particle, it can become the system bottleneck that has to deal with communication overhead or distributed transactions. The integration unit is usually an ESB, that serves the purpose of integration, orchestration, routing, event processing, correlation, and business activity monitoring. From the communication perspective, SOA is about orchestrating large services.

4. MICROSERVICE ARCHITECTURE

µServices base on three Unix ideas [115]:

- A program should fulfill only one task, and do it well.
- Programs should be able to work together.
- Programs should use a universal interface.

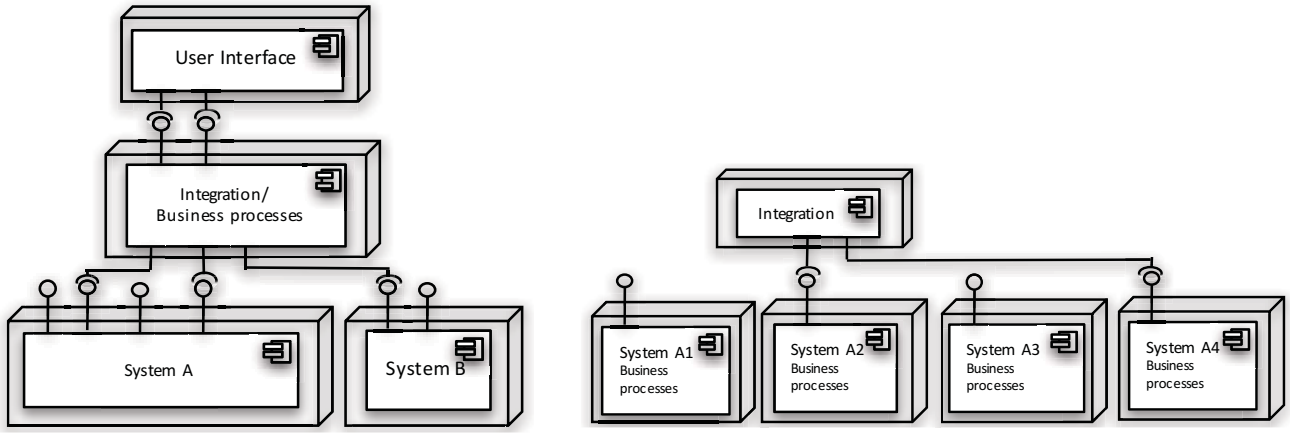
These ideas lead to a reusable component design, supporting modularization. The major point is that services are brought to production independently of each other, which is one of the main differences with most SOA solutions. It does not only impact deployment but also evolution and modification efforts. µService followers often cite Conway's law [29], stating that "Organizations, which design systems are constrained to produce designs which are copies of the communication structures of these organizations."

µServices emphasize lightweight virtual machines. They are implemented as containers (e.g., a Docker) or individual processes. This unbinds dependency on a certain technology, enabling usage of a service-specific infrastructure. µServices usually do share the same database schema as it would predetermine a bottleneck as well as coupling. Each µService is in charge of its own data model, which possibly leads to replication. In the Background section, we mentioned Bounded context, which is the direction for µServices. Later in this section, we elaborate more details on Bounded context.

Unlike SOA, µServices do not have integration component responsible for service orchestration and prefer choreography. Business processes are embedded in services, and there is no logic in the integration. Thus µServices themselves are responsible for interaction with others. This gives limited flexibility to design or adjust business processes over the company IT's. It is a payoff for µService independent service management and deploys. Of course, one can still utilize orchestration[1]; however, this is not a typical approach.

It can be noted that µServices emphasize isolation in a way that a particular process and user interaction operate in the scope of a particular service. A service is usually managed by

[1] https://github.com/Netflix/conductor



(a) Sample SOA deployment. **(b) Sample µService deployment**

Figure 3: Sample SOA and µService architecture deployment of Systems A&B.

a single team; however, changes to µServices require user interface propagation. For this reason, both the user interface and µService should be under control of a single team. This gives the team flexibility to manage changes while avoiding bureaucratic negotiation on interface changes. Having µServices independent regarding development and deployment bringing individual scalability and continuous delivery. This provides resilience to failure [12] since a request may be balanced among several service instances.

In [11], authors discuss a *DevOps* practice, which goes hand-in-hand with µServices. DevOps is a set of practices that aim to decrease the time between changing the system and deploying the change to production while maintaining software quality. A technique that enables these goals is a DevOps practice [13]. One prevalent DevOps practice is continuous delivery, enabling on-demand automated deployments supporting system elasticity to request load. Similarly, continuous monitoring provides early feedback to detect operational anomalies.

The difference from SOA can be seen in Fig. 3b that shows System A mentioned in Fig. 3a. Each µService is an individually-deployable unit maintained by a separate (or the same) team. Note in the figure that no complex integration technology exists over the enterprise; the integration part can be the user interface part or services can interact directly.

Compare to SOA, the user interfaces part may be integrated into the µService, which avoids communication overhead. The communication among µServices does not require REST or messaging, and the user interface integration may talk to other services and involve data replication instead; however both REST and messaging are commonly used.

µService should be comprehensible by individual developers and not developed by multiple teams [62]. At the same time, it cannot become a nanoservice since it would demand high network communication, which is expensive compared to local communication. Transactions spanning multiple µServices become complex. For this reason, the design should target transaction spanning a single µService or involve messaging queue. However, recently a no-ACID transaction type has been proposed for this context, known as compensation transactions[2]. Similarly, regarding data, a µService should be enough large to ensure data consistency.

For µServices, it is a critical decision when it comes to fragmenting the data model; we mentioned this when introducing the Bounded context. A single service cannot capture the whole context, but there must have a certain boundary. There are strategies [62] to determine how two systems interact, determining the boundary.

1. The *shared kernel* strategy suggests that each domain model shares common elements, but in the specialization areas, they differ.
2. *Customer/supplier* suggests that the subsystem provides a domain model that is determined by the caller.
3. In *conformist*, the caller uses the same model as provided by the subsystem and reuses its knowledge.
4. The *anti-corruption layer* provides a translation mechanism to keep two systems decoupled. This is often used for legacy systems.
5. *Separate ways* suggests no integration among systems.
6. *Open host service* expects a system to provide special services for everybody's use to simplify integration
7. *Published language* has unchangeable linguistic elements (contracts, events, etc.) visible to the outside with a meaning in multiple subdomains.

From the data model perspective, the above strategies (4-6) provide a lot of independence, while (1-3,7) tie the domain models together. From the communication efforts perspective among teams, (5) requires least efforts followed by (3), (4), (6), (7), (2) and (1) with most efforts.

4.1 Self-Contained Systems

The last architectural variation we mention is a *Self-Contained System* (SCS) [2]. In this approach, a particular system breaks into multiple SCS components that consist of two parts, a user interface and separately-deployable µServices sharing the same code-base. Various SCSs communicate asynchronously if necessary. Each functional is ideally under

[2] http://jbossts.blogspot.cz/2016/10/achieving-consistency-in-microservices.html

a particular SCS component. To draw a comparison, an e-commerce shop system would contain 100 µServices or would only consist of 5-25 SCSs. SCS suggests that a µService has around 100 lines of code. For instance, in practice, in Java EE the project setup would have a multi-module maven project sharing code between the UI and µServices where each part is separately deployable WAR file. SCS can be seen as a specialization of µServices. The approach is promoted by various authors [115].

5. ENLIGHTENING EXAMPLE

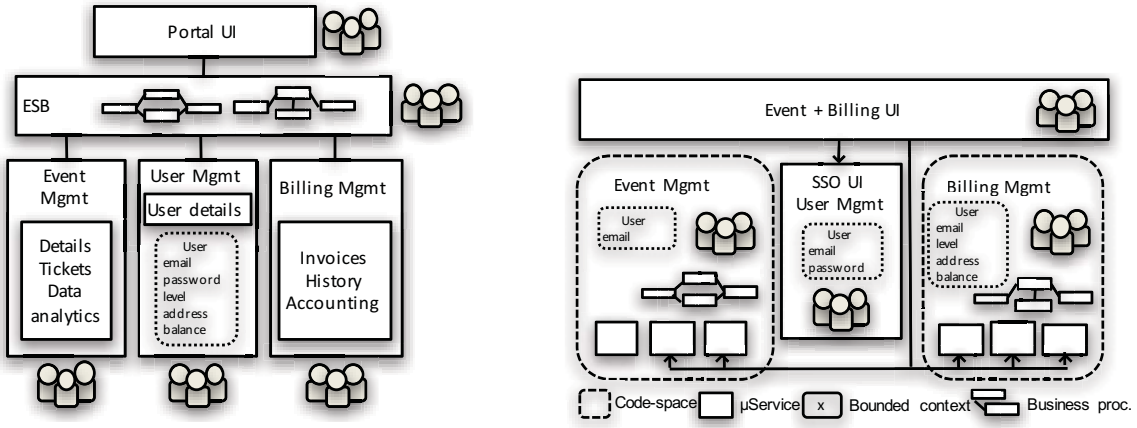
To clearly understand the differences, we examine an example application designed in SOA, µServices, and SCS. The aim is to emphasize the characteristics of each particular approach. The example system is an event ticketing system assisting users with ticket selection and purchase. For the purpose of demonstration, it consists of various components. We highlight three: Event Management (EMgm), User Mgmt (UMgm) and Billing Mgmt (BMgm).

In SOA design, we consider the canonical data model, which is enforced by contract agreement on the integration level implemented by ESB, which handles business processes and orchestrates management components. The User Interface (UI) part is a portal also handling user authentication across the system. Fig. 4a shows possible design decomposition in SOA. Each system is a separate application, exposing its own web services. While such applications are independent, the contract agreement and centralization pushes towards deployments as a monolith. Service orchestration goes through the ESB, and the portal sits on top of the ESB. Each management component has a separate code space; however, all component agree on data model in order to design processes in the ESB. We highlight *user* data model with multiple attributes managed by UMgm. A change to particular management component must be well discussed and promoted to the ESB by maintaining teams; the change most likely promotes to the portal as well impacting another teams. However, the dependencies may impact services in different components, e.g. user data model changes may impact other management components and their internals.

The µServices approach no longer considers the centralized integration via ESB; instead, it delegates business processes to services. Services may share the same code-space, avoiding repetition denoted by the *dashed box*, while still allowing extraction of various deployables as separate µServices. The original UMgm component partially dissolves into the Single Sign-On module (SSO) for authentication, which is implemented by the portal in SOA. Moreover, notice the bounded contexts of *user* data model in each code space and the SSO. Fig. 4b shows the example system. Note the different teams responsible particular components. Moreover, note the location of business processes definition and shared code space. The UI is maintained by a separate team responsible for the service integration. Moreover, a service from EMgm could call a service from BMgm. Changes impacting a particular µServices can be deployed individually, and the data model extension can be service specific. However, these may impact the UI and another team, which is an argument for SCS that we consider next as variant of µServices.

The SCS design is a specialization of µServices, where the same team maintaining a particular service is now responsible for the UI. However, it is not a separate UI application above the services, but a separately deployable application with direct code access. This has multiple advantages. The team is familiar with the knowledge, it fully controls the changes and change-propagation, and there is less overhead since no web services need to be used for given UI scope. However, the team deals with the whole development stack, including UI, middleware, and database development. SCSs should ideally not communicate with each other, while this is fine for µServices. Moreover, SCSs should favor integration at the UI layer. Fig. 5 shows the transition from µServices into SCS. It highlights the integration in the Billing UI involving the SSO and selected EMgm µServices. The distinct UIs for Billing and Events must correlate with the look-and-feel in order to confirm unity of a single system. When a small change is needed in one SCS, the UI and a particular µServices are redeployed, independent of other µServices in or out of the code base or other SCSs.

For a deeper understanding, consider a possible component interaction in a use case when a user purchases selected tick-



(a) A SOA system with highlighted user data format **(b) A µServices system, showing user data format divided**

Figure 4: An example system in various architectures highlighting user data.

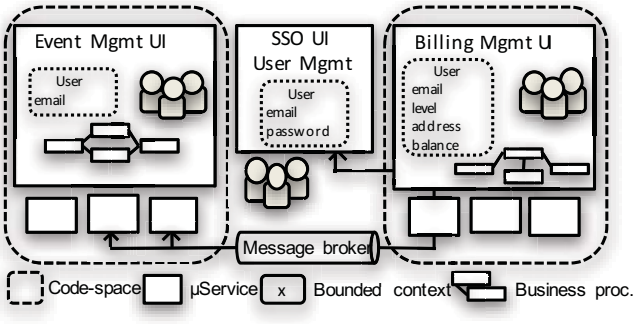


Figure 5: An SCS system, dividing user data format

ets. We consider that the user finds tickets through the UI, and next aims to process the purchase of his/her selection. First, consider SOA processing in our example in Fig. 4a. It would look similar to this:

1. The user sends a request via the portal to buy tickets
2. It propagates to ESB that triggers the business process orchestrated by the ESB
3. EMgm checks whether the tickets are still available, and it locks the selected tickets for 30 minutes
4. BMgm makes sure that user paid previous orders
5. UMgm loads user address and level for possible discount
6. BMgm resolves the discounted price and processes payment
7. EMgm removes tickets locks and makes them not available
8. BMgm issues an invoice for user address
9. UMgm recalculates user-level based on recent purchase

We can see that ESB has a rather complex responsibility to orchestrate a lot of services. However, it is easy to reroute the process to different order or consider alternatives on top of existing services.

Next, we consider the interaction in µServices to highlight the differences. The aim is to give the responsibility of a particular process handling to a given µService - in the example we call it payService under the BMgm. If possible we try to avoid distributed transactions; however services may act in a choreography. When considering Fig. 4b, the interaction may look like:

1. User selects tickets using the UI involving EMgm µServices and starts the purchase posting the selection the payService
2. The payService triggers the business process
3. It contacts EMgm µService to make sure the tickets are still available, locking them for 30 minutes in EMgm
4. The payService loads current user from bounded context and checks pre-existing payment dues
5. It uses user-level to determine possible discount
6. Next, payService processes the purchase payment
7. payService contacts EMgm µService to remove the ticket lock, updating ticket availability
8. It issues an invoice for an address from bounded context
9. Finally, the payService updates user-level in bounded ctx.

From above we see a dependency of the payService to the locking/unlocking EMgm µService. When considering SCS in Fig. 5 the system does not dependencies and the interaction is decoupled, e.g. through a message broker:

1. User selects tickets in EMgm UI and starts the purchase by locking the tickets for 30 minutes through EMgm UI; next, it routes and posts the user selection to the BMgm UI
2. The request triggers the business process in the BMgm UI
3. BMgm UI checks user's pre-existing due payments
4. To determine possible discount BMgm UI loads user-level from the bounded context
5. Next, BMgm UI processes the purchase payment
6. To remove the lock and set availability on selected tickets in EMgm, the UI BMgm contacts its µService, which emits an event to a message broker, to which an EMgm µService reacts - updating ticket availability and locks
7. BMgm UI issues invoice for an address from bounded ctx.
8. Finally, it updates user-level in bounded context

We can see that the interaction is delegated to a particular SCS, while it emits events to get outside of the SCS. Changes to the process involve SCS modification and new deployment, however, there is higher autonomy in performing such a change, possibly involving a single team.

In a second use case we consider data analytics. It aims to send an advertisement email to past customers offering a discount to events matching user history and details. We start with SOA and next consider µService approach:

1. EMgm is the initiator of the action over ESB
2. It polls an aggregate user list with details from UMgm
3. In batch for each user, it issues a business process via ESB
4. It fetches the last user event from BMgm
5. For no history it skips the user; for existing history, it determines the event category in the EMgm
6. In EMgm it finds a matching event by category, the earliest date nearby user address
7. BMgm determines the price basing on user-level
8. EMgm sends the advertisement content via user email

In µServices, a particular service performs the process:

1. EMgm µService is the initiator
2. It uses BMgm to fetch user filtered list with all details from bounded context and the last attended event
3. It initiates a batch business process for each user
4. It determines the event category, based on users last event
5. Next, it finds the first matching event by category, earliest date, and location near to user address
6. It determines the ticket price using BMgm µService, considering the particular user.
7. Finally, it sends the advertisement through EMgm µService

Similarly, even in this case, we see SOA's flexibility to reorder or extend the process using existing services on the ESB level. In SCS we would need to involve decoupling; however, it may introduce a redundancy in user history in both BMgm and EMgm to simplify the processing, which is, unfortunately, a common approach for preserving autonomy.

When a third-party service appears providing the distance from user location to the event venue, ESB has a wide pallet of communication adapters to contact it, while most likely introducing a new service as a facade. In µServices with protocol independence, we may contact the service directly, but only if our language provides an implementation of the protocol, which may lead again to the introduction of a new µService. It seems the best solution is to combine µServices with an integration framework providing a collection of adapters.

Table 1: Comparing µServices and SOA

Concern	µServices	SOA
Deploy	Individual service deploy	Monolithic deploy, all at once
Teams	µServices managed by individual teams	Services, integration and user interface managed by individual teams
User interface	Part of µService	Portal for all the services
Architecture scope	One project	The whole company/enterprise
Flexibility	Fast independent service deploy	Business process adjustments on top of services
Integration mechanism	Simple and primitive integration	Smart and complex integration mechanism
Integration technology	Heterogeneous if any	Homogeneous/Single vendor
Cloud-native	Yes	No
Management/governance	Distributed	Centralized
Data storage	Per Unit	Shared
Scalability	Horizontally better scalable. Elastic	Limited compared to µServices. Bottleneck in the integration unit or a message parsing overhead. Limited elasticity.
Unit	Autonomous, un-coupled, own container, independently scalable	Shared Database, units linked to serve business processes. Loosely coupled.
Mainstream communication	Choreography[1]	Orchestration
Fit	Medium-sized infrastructure	Large infrastructure
Service size	Fine-grained, small	Fine or coarse-grained
Versioning	Should be part of architecture, more open to changes	Maintaining multiple same services of different version
Administration level	Anarchy	Centralized
Business rules location	Particular service	Integration component

6. COMPARISON

Both µServices and SOA divide systems into services, but in different ways. SOA can still be seen as a monolith from the deployment perspective [115], while µServices lead towards independent deploys. Industry relates µServices to container technologies simplifying automated deployment [62]. Containers can be given the credit for building such self-contained µService deployment units. Moreover, µService go towards design autonomy with plenty of teams resulting in heterogeneity of components, which some may criticize.

SOA has a wide enterprise scope, while the intention of µServices is to do "one thing well" [79]. SOA gains it flexibility from centralized management while µServices inclines for distribution. µServices fit well to the context of cloud computing. Researchers refer to µServices as cloud-native, while SOA is rarely referenced that way in the literature [62]. The key features here are the individual and automated service deployment. In general, service-based approaches are vital for cloud-native approaches [37].

Considering industry demands and recent research directions [62], µServices seems to be the future direction. However, there are counterexamples. In [76] author argues that SOA and µServices are not the same as nothing in µServices build on SOA and multiple pieces are missing for µServices. [93] points out the fundamental concepts:

> µServices architecture is a share-as-little-as-possible architecture pattern that places a heavy emphasis on the concept of a bounded context, whereas SOA is a share-as-much-as-possible architecture pattern that places heavy emphasis on abstraction and business functionality reuse.

SOA is a better fit to a large, complex, enterprise-wide infrastructure than µServices [93]. SOA suits well to the situation with many shared components across the enterprise. µServices do not usually have messaging middleware and fit better to smaller, well-partitioned web based applications. Moreover, SOA better enables services and service consumers to evolve separately, while still maintaining a contract. µServices fail to support contract decoupling, which is a primary capability of SOA. Finally, SOA is still better when it comes to integrating heterogeneous systems and services. µServices reduce the choices for service integration.

Table 1 provides a summary comparison between SOA and µServices, which we described in this paper.

7. RESEARCH CHALLENGES IN SERVICE INTEGRATION

Both architectures come with drawbacks and features that are complex or cause difficulties. These are challenges to address in research. Clearly, SOA has the issue with monolithic deploy, centralization, and bound data model leading into canonical data model or complex protocol stack. On the other hand, it is flexible with business process changes, giving a centralized view on system processes. µServices bring higher autonomy to services reducing data structure dependencies, relocating business processes to particular services. This together with the connection of virtual boxes brings the benefits of individual service deploy enabling elastic service scalability.

Multiple issues can be found in service composition involving both SOA and µServices. In [67], authors consider such issues. They note cross-cutting concerns that repeat across services, such as exceptions, transactions, security, and ser-

vice-level agreements. One of the main issues they point out is knowledge reuse. We may want to reuse a component, a data transformation rule, a process fragment, or template. To some extent, SCS accomplishes this in a limited scope. One possible reuse approach is to describe the certain rules in machine readable or queryable format, e.g., it is easy to make a query to the database to find expected data constraints. The most common approach is to manually evaluate the knowledge and copy/paste it to a secondary location. However, this only exacerbates the complexity of evolution management, since once the knowledge changes, it has multiple locations to maintain. Next, there exist recommender systems [67] to facilitate the composition process. For instance, they perform on-the-fly similarity search over a knowledge base of reusable patterns.

Paths addressing the above issues in service integration use automation. A synthesis [67] reuses knowledge of integrated services. For instance, symbolic model checking may generate an executable business process for the integrating component [88]. Artificial intelligence can be applied involving semantic service with machine-readable descriptions of service properties and capabilities with reasoning mechanisms to select and aggregate services [92]. Naturally, the problem with this approach is the extensive development effort to provide and maintain the semantic information in correlation to the system internal knowledge. Finally, involving Model-Driven Development approach on service design allows transformation of knowledge across various services. However, because of the high-level of abstraction used in the approach demanding complex generalization and design of transformation rules, the approach is rarely used. Developers prefer to focus on specific problem description rather than its abstraction.

Specifically for µServices, the issue can raise from the service-specific data model or business processes hidden from others, which facilitates the service autonomy. On the other hand, this leads to replication of knowledge among services with service integration. Moreover, µServices sacrifice the centralized view on business processes, or generally the knowledge, that is now distributed and hidden across services. For example, consider two communicating Services A and B. These services exchange information while both maintain information in distinct data formats, conforming business rules or processes, or even security restrictions. Once the Service A changes its internal knowledge, it must still correlate to Service B. While the correlation applies only to some extent, it could cause system failure if not properly maintained. Let's consider that Service B could base its reasoning on a Service A internal knowledge in computation. This could leverage the maintenance efforts in service composition. A great example where this happens often is the user interface.

When low-level data format changes, the user interface forms, tables, and reports must reflect such change. However, this is very difficult to preserve [22] since components are maintained by distinct teams and limited type safety exist in user interfaces. In [22] the author suggests utilizing meta-programming and aspect-oriented programming to let the source service stream meta-information to the integration component. For example, a web browser JavaScript library weaves the together provided information and templates at runtime to assemble the user interface fragments reflecting the actual information from the service. Such approach minimizes maintenance efforts on the user interface part since any change in the service is immediately reflected in the user interface fragment. The approach complies well with [24] contemporary client-based frameworks such as Angular2 or React. Moreover, having the meta-information exposed allows rendering the user interface fragments on various platforms using native components, while all adapting to the changes in services or even context.

While [22] provides an approach for minimizing change impacts of structural data manipulation across integrating services, there is still a dependency on business rules and security. In [20], the authors suggest that services capture their constraints, business rules, and security untangled from the service source code, e.g., in domain specific language description or involving annotation descriptors. This not only improves readability of the above but also allows their automated inspection and transformation into a machine-readable format that can be shared across services. [20] shows a system with its internal constraints and business rules captured in Java EE and Drools being automatically inspected and transformed into a JavaScript description applied across user interface enforcing the constraints and rules already at the client-side.

There are multiple open challenges left to address, beyond the scope of this paper. We highlight the main areas:

1. How should be changes and evolution communicated across different teams working in distinct services?
2. How to detect/test incompatibilities in service integration?
3. How to determine the scope/boundary of a particular µService?
4. How to effectively mitigate failures in service integration?
5. Distributed transactions across services, e.g. compensating transactions, and no-ACID transactions.
6. Cross-cutting issues with code replication across services.
7. Security across services must correlate when one service allows half of the process the second can deny it.
8. How to migrate existing systems onto µServices?
9. How to deal with replicated info. in service specific databases?

8. MICROSERVICE MAPPING STUDY

This chapter discusses research trends and analyzes research challenges addressed by existing related work. The analysis is broad and considers one hundred papers that deal with µServices. Our strategy to harvest evidence considering the approach of a mapping study [1]. Through multiple research indexing sites and portals, we download an evidence to analyze and classify.

8.1 Harvesting the Evidence

To receive systematic evidence in the area of µServices, we involve the main indexing sites and portals (indexers) including IEEE Xplore, ACM Digital Library (DL), SpringerLink and ScienceDirect. These indexers provide a mechanism to search on existing conference and journal papers published in the past years using full text terms, utilizing logical com-

Table 2: Resources and filtering using indexers

Indexer	Abstract filter & Downloaded	First mention	Full text filter
IEEE Xplore	55	2014	53
ACM DL	29	2015	22
SpringerLink	24	2015	13
ScienceDirect	11	2015	9
Other sources	4	-	3
Sum	124	-	100

bination of these terms. While the syntax and capabilities differ, all provide the ability to search within the title, abstract, keywords and some even through full text.

Our search query is rather primitive and we search for occurrences of "Microservice", "Micro-service", "Microservices" or "Micro-services". The composed query looks as follows:

```
("Microservice" OR "Micro-service"
          OR "Microservices" OR "Micro-services")
```

However, not all papers found by the search discuss µServices. Some papers define the same term for a different matter, or just mention µServices as an existing approach in related work, etc. For this reason, we filter papers outside of the target scope, papers with no specific output or presenting an opinion.

Table 2 shows in the first column the total number of papers we downloaded from a specific indexer. However, the total number of examined papers was even higher. For instance, a search with IEEE Xplore resulted in 218 papers to consider, which we further filtered to 55 papers. In the next, stage we filtered papers based on the full-text analysis. In the next stage, we eliminated papers focused on case studies, experience, and opinion papers as they do not fit to the scope of research challenges and directions. From the reverse perspective, we consider related work and references from selected papers to extend the considered pool of papers. Besides the four, above-mentioned indexers, we obtained work from other sources. The final numbers of papers we work with are represented by the last column of Table 2. The table also shows the year of first mention in the particular indexer.

8.2 Similar Studies

There are similar works addressing mapping study of SOA [62] and µServices [6] [82], providing a roadmap. [6] gives a reasonable format of categorization of research challenges, while considering credible sources. In this study, we consider similar categorization structure and thus build on [6]. We include the same 33 sources referenced from their work; however, our approach is different. First we consider 107 papers. Next, we extend the categorization set. Finally, in their work, they employ a process where they define keywords for which they search the full texts papers to draw categorization. In our case, we utilize keyword extraction since a search for a text occurrence does mean it fits to a keyword. To accomplish this, we utilize RAKE algorithm [95] that extracts reasonable keywords that we consequently categorize similarly to [6]. However, we manually verify and correct the categorization.

8.3 Research Challenge Categorization

We identify the following challenges and keywords from the existing evidence in these categories:

Communication/integration - considers interaction among particular services and their integration. Often it considers communication protocols, service types, and integrating mechanisms. The identifying keywords are: *API, API gateway, REST, sockets, TCP, reuse metrics, network communication, interaction flow, proxy, routing, router, enterprise service bus, orchestration, SOAP, JMS, Messaging*

Deployment operations - focuses on the deployment efforts, automation, resource utilization with respect to efficient scaling. Often the topics are related to isolated containers (such as Docker), and cloud-based infrastructure. This category is especially useful when it comes to system operations. The identifying keywords are: *sysops, devops, system operations, deployment executor, deployment cost, automated deployment, infrastructure management, infrastructure cost, orchestration deployment, configuration, rolling upgrades, images, container, deployment tool, docker*

Performance - is a common software quality attribute. In µServices, we are interested in throughput, latency, the timing for service execution, scaling, elasticity, synchronization and interaction overhead introduced by bounding context, etc. The following keywords identify performance challenges: *QoS, performance, SLA, horizontal scaling, total time, response time, processing delay, processing time, total latency, total traffic, traffic demands, load balancer, load balancing*

Case study - shares experience and case studies, usually to demonstrate the approach, and we identify such papers. The identifying keywords are: *case study, evaluation, simulation, production environment, proof of concept*

Fault tolerance - refers to the ability to prevent or recover from failure. This often relates to system migration, detection of failure and may cross-cut with another category related to monitoring mentioned next. The identifying keywords are: *fault, failure, recovery, health management, tolerance, healing, circuit breaker, CAP*[3]*, Consistency, availability, partition tolerance*

Testing/Quality Assurance - considers the testing specifics for µServices. These works consider quality attributes of the software or event testbeds. The identifying keywords are: *testing, testbed, test, quality attributes, quality assurance, software quality, verification*

Service discovery - is the process to determine and detect near services, register and maintain them over the time. It considers strategies on gateway configuration, reporting service availability, etc. The identifying keywords are: *discovery, registration, service registry*

Tracing/logging/monitoring - relates to the approaches used in µServices for tracing interaction, process logging which can span across services as well as to monitoring. This area often spans to fault tolerance, performance or even security. Approaches spanning from central to distributed logging exist. Often we need to debug the system to resolve errors or

[3]consistency (C), availability (A), and partition tolerance (P)

Table 3: Mapping of evidence to categories

Challenge	Keywords	Volume	References
Communication/integration	*API, API gateway, REST, sockets, TCP, reuse metrics, network communication, interaction flow, proxy, routing, router, enterprise service bus, orchestration, SOAP, JMS, Messaging*	40	[50][43][118][73][66][30][109] [5][74][27][17][44][106][3][36] [114][62][121][25][119][103][89] [80][86][87][113][32][99][120] [111][77][58][33][117][7][38] [42][65][46][110]
Deployment operations	*sysops, devops, system operations, deployment executor, deployment cost, automated deployment, infrastructure management, infrastructure cost, orchestration deployment, configuration, rolling upgrades, images, container, deployment tool, docker*	26	[55][61][66][30][112][106][3] [47][62][121][119][91][53][41] [113][28][58][117][102][7] [65][94][85][26][5][100]
Performance	*QoS, performance, SLA, horizontal scaling, total time, response time, processing delay, processing time,total latency, total traffic, traffic demands, load balancer, load balancing*	22	[15][60][14][80][87][61][66][113] [112][69][111][51][101][7][59] [114][62][31][19][46][85][48]
Case study	*case study, evaluation, simulation, production environment, proof of concept*	21	[82][7][59][77][16][58][112][62] [106][111][55][66][18][113][39] [104][84][56][52][8][81]
Fault tolerance	*fault, failure, recovery, health management, tolerance, healing, circuit breaker, CAP,Consistency, availability, partition tolerance*	21	[15][49][41][43][68][55][75][77] [40][58][33][10][107][114][97] [46][94][25][63][5][45]
Testing/Quality Assurance	*testing, testbed, test, quality attributes, quality assurance, software quality, verification*	16	[98][6][61][113][109][112][111] [77][51][44][3][59][9] [31][90]
Service discovery	*discovery,registration,service registry*	11	[71][64][4][96][40][36][102][19] [46][85][100]
Tracing/logging/monitoring	*runtime metrics, call graph, tracing, logging, debugging, profiling, monitoring, application monitoring, health monitoring*	10	[51][80][68][66][72][28][45] [17][34]
Modeling	*modeling, UML, chart, visualization, formal specification, schema*	9	[70][89][41][105][57][10][65] [17][75]
Security	*security, secure, authentication, authorization, OAuth, OAuth2, encryption, vulnerability, attack, access control, OpenID, privacy*	8	[35][44][98][4][114][83][90][103]
Mapping study/Survey	*survey, mapping study, literature, literature review, systematic mapping*	5	[82][6][16][62][45]
Context-awareness	*personalized, personalization, recognition, contextual, context aware, context-aware, neural network*	3	[78][80][10]

bottleneck and since µServices are distributed this is more challenging than in monoliths. The identifying keywords are: *runtime metrics, call graph, tracing, logging, debugging, profiling, monitoring, application monitoring, health monitoring*

Modeling - provides analytic methods, related to models, visualization and formal specification to build a better high-level understanding across the services. The identifying keywords are: *modeling, UML, chart,visualization, formal specification, schema*

Security - is a general topic and µServices have their specifics, various work consider usage of security protocols with respect to service interactions and trust, authentication or global identity management. The identifying keywords are: *security,secure, authentication, authorization, OAuth, OAuth2, encryption, vulnerability, attack, access control, OpenID, privacy*

Mapping study/Survey - is related to literature review. The identifying keywords are: *survey,mapping study, literature, literature review, systematic mapping*

Context-awareness - is the context-awareness provided by service. These works include adaptability or personalization. Some existing work also considers the usage of neural networks, which we include in this category. The identifying keywords are: *personalized, personalization, recognition, contextual, contextaware, context-aware, neural network*

8.4 Mapping to Categorization

Next, we apply the categorization to the aggregated evidence. Since the mapping is better to read in a structured format, we show the mapping in Table 3. The table shows the particular category, the volume of work and particular papers we considered. Multiple papers, however, fit into multiple categories, and we do not enforce exclusive designation. Fig. 6 then shows the relative volume of work per category. The top keywords when compared to [6] are much lower. The top keyword is API gateway with 9 occurrences, testing with 7, security with 5, and service discovery with 5. Other keyword occurrences showed 4 or fewer hits using the RAKE algorithm.

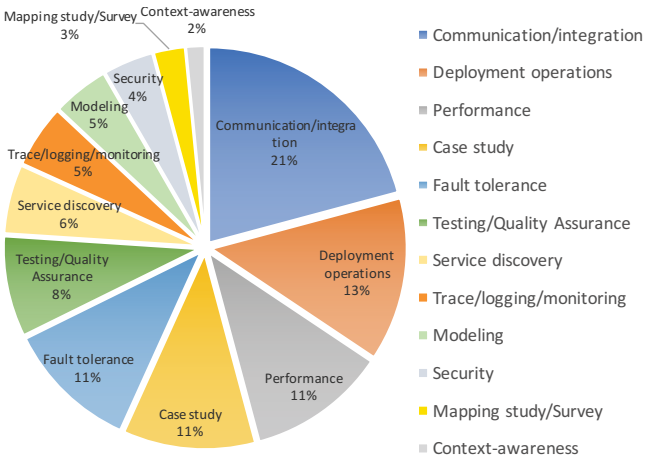


Figure 6: Relative interest volume in per category

This mapping shows an up-to-date research road map for µServices. Next, it depicts existing interest in particular categories. When considering the major benefits of µServices, such as alternative to SOA, independent development and deploy, automation for deployment, good scalability, we may notice that these reflect and correlate with the main research interest. However, important topics including monitoring, tracing, security, or service discovery do required research interests. Modeling and context-awareness usually apply to mature approaches. Significant attention is payed to case studies, but their actual merit on production-level approach is questionable.

8.5 Threats to Validity

A primary threat to the validity of survey studies is inadequate coverage. This is addressed by the breadth of our coverage and the design of our research parameters. Similar to the guidelines for conducting systematic mapping studies [1], we performed the evidence elimination. While, 100 percent coverage of related papers cannot be guaranteed, we believe we have selected all relevant papers, within the scope of this study. We addressed this threat by selecting and examining several search strings that fit our paper control set. Next potential threat is data extraction bias based on the human factor. We addressed this threat by using a RAKE keyword extraction algorithm [95] that we matched with manually-extracted ones.

9. CONCLUSION

This paper attempts to demystify ambiguous usage of term and definitions of SOA and µServices. It specifies characteristics and differences of both architectures, pointing out their strengths, weaknesses, and differences. While both architectures address system integration, the industry seems to move toward µServices, leaving SOA as legacy. The main credit for this tendency can be given to the ability of independent service deploy and elastic scalability.

For instance, in the market of integrated systems, Gartner[4] predicts a boom where hyperconverged systems reach 24% of the overall market by 2019. Moreover, Gartner states that in this segment recently started new era bringing the continuous application and µServices delivery.

While µServices seems the be the winner of the two, there are still multiple challenges that come with the architecture that leaves developers with restatements and complex processing. These are addressed in future research; we show multiple of these issues are actively addressed by researchers. In this paper, we introduced a mapping study from around 100 papers related to µServices.

Readers may also wonder about the symbiosis with Internet of Things, Cloud computing, Platform as a Service (PaaS), and systems for handling Big Data. All these can be addressed using µService architecture; however these are beyond the scope of this paper and left for future work.

In the end, we raise a philosophical question. Since the evolution path went from heterogeneous system integration, through SOA into µServices, can we expect in near future another step back towards SOA?

[4] http://www.gartner.com/newsroom/id/3308017

Acknowledgments

This research was supported by the ACM-ICPC/Cisco Grant 0374340.

10. REFERENCES

[1] Guidelines for conducting systematic mapping studies in software engineering. *Inf. Softw. Technol.*, 64(C):1–18, Aug. 2015.

[2] Self-contained systems, 2017. http://scs-architecture.org.

[3] C. M. Aderaldo, N. C. Mendonça, C. Pahl, and P. Jamshidi. Benchmark requirements for microservices architecture research. In *Proceedings of the 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering*, ECASE '17, pages 8–13, Piscataway, NJ, USA, 2017. IEEE Press.

[4] M. Ahmadvand and A. Ibrahim. Requirements reconciliation for scalable and secure microservice (de)composition. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pages 68–73, Sept 2016.

[5] S. Alpers, C. Becker, A. Oberweis, and T. Schuster. Microservice based tool support for business process modelling. In *2015 IEEE 19th International Enterprise Distributed Object Computing Workshop*, pages 71–78, Sept 2015.

[6] N. Alshuqayran, N. Ali, and R. Evans. A systematic mapping study in microservice architecture. In *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 44–51, Nov 2016.

[7] M. Amaral, J. Polo, D. Carrera, I. Mohomed, M. Unuvar, and M. Steinder. Performance evaluation of microservices architectures using containers. In *2015 IEEE 14th International Symposium on Network Computing and Applications*, pages 27–34, Sept 2015.

[8] F. Andry, R. Ridolfo, and J. Huffman. Migrating healthcare applications to the cloud through containerization and service brokering. In *HEALTHINF*, 2015.

[9] T. Asik and Y. E. Selcuk. Policy enforcement upon software based on microservice architecture. In *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 283–287, June 2017.

[10] P. Bak, R. Melamed, D. Moshkovich, Y. Nardi, H. Ship, and A. Yaeli. Location and context-based microservices for mobile and internet of things workloads. In *2015 IEEE International Conference on Mobile Services*, pages 1–8, June 2015.

[11] A. Balalaie, A. Heydarnoori, and P. Jamshidi. Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software*, 33(3):42–52, May 2016.

[12] A. Balalaie, A. Heydarnoori, and P. Jamshidi. *Migrating to Cloud-Native Architectures Using Microservices: An Experience Report*, pages 201–215. Springer International Publishing, Cham, 2016.

[13] L. Bass, I. Weber, and L. Zhu. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 1st edition, 2015.

[14] F. Z. Benchara, M. Youssfi, O. Bouattane, and H. Ouajji. A new efficient distributed computing middleware based on cloud micro-services for hpc. In *2016 5th International Conference on Multimedia Computing and Systems (ICMCS)*, pages 354–359, Sept 2016.

[15] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan. Multi-objective scheduling of micro-services for optimal service function chains. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6, May 2017.

[16] J. Bogner, S. Wagner, and A. Zimmermann. Towards a practical maintainability quality model for service-and microservice-based systems. In *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*, ECSA '17, pages 195–198, New York, NY, USA, 2017. ACM.

[17] M. Brambilla, E. Umuhoza, and R. Acerbis. Model-driven development of user interfaces for iot systems via domain-specific components and patterns. *Journal of Internet Services and Applications*, 8(1):14, Sep 2017.

[18] F. Callegati, G. Delnevo, A. Melis, S. Mirri, M. Prandini, and P. Salomoni. I want to ride my bicycle: A microservice-based use case for a maas architecture. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 18–22, July 2017.

[19] A. Celesti, M. Villari, and A. Puliafito. An xri naming system for dynamic and federated clouds: a performance analysis. *Journal of Internet Services and Applications*, 2(3):191–205, Dec 2011.

[20] Cemus, Karel and Klimes, Filip and Kratochvil, Ondrej and Cerny, Tomas. Separation of concerns for distributed cross-platform context-aware user interfaces. *Cluster Computing*, pages 1–8, 2017.

[21] T. Cerny, M. J. Donahoo, and J. Pechanec. Disambiguation and comparison of soa, microservices and self-contained systems. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, RACS '17, pages 228–235, New York, NY, USA, 2017. ACM.

[22] Cerny, Tomas and Donahoo, Michael J. On Separation of Platform-independent Particles in User Interfaces. *Cluster Computing*, 18(3):1215–1228, sep 2015.

[23] Cerny, Tomas and Donahoo, Michael J. *Survey on Concern Separation in Service Integration*, pages 518–531. Springer Berlin Heidelberg, 2016.

[24] Cerny, Tomas and Donahoo, Michael Jeff. On Energy Impact of Web User Interface Approaches. *Cluster Computing*, 19(4):1853–1863, dec 2016.

[25] P. R. Chelliah. The hadoop ecosystem technologies and tools. Advances in Computers. Elsevier, 2017.

[26] G. Cherradi, A. E. Bouziri, A. Boulmakoul, and K. Zeitouni. Real-time hazmat environmental information system: A micro-service based architecture. *Procedia Computer Science*, 109(Supplement C):982–987, 2017. 8th International Conference on Ambient Systems, Networks and Technologies, ANT-2017 and the 7th International Conference on Sustainable Energy Information Technology, SEIT 2017.

[27] M. Ciavotta, M. Alge, S. Menato, D. Rovere, and P. Pedrazzoli. A microservice-based middleware for the digital factory. *Procedia Manufacturing*, 11(Supplement C):931–938, 2017. 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy.

[28] A. Ciuffoletti. Automated deployment of a microservice-based monitoring infrastructure. *Procedia Computer Science*, 68(Supplement C):163–172, 2015. 1st International Conference on Cloud Forward: From Distributed to Complete Computing.

[29] M. E. Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

[30] D. D'Agostino, E. Danovaro, A. Clematis, L. Roverelli, G. Zereik, and A. Galizia. From lesson learned to the refactoring of the drihm science gateway for hydro-meteorological research. *Journal of Grid Computing*, 14(4):575–588, Dec 2016.

[31] A. de Camargo, I. Salvadori, R. d. S. Mello, and F. Siqueira. An architecture to automate performance tests on microservices. In *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*, iiWAS '16, pages 422–429, New York, NY, USA, 2016. ACM.

[32] R. Debab and W.-K. Hidouci. Towards a more reliable and robust cloud meta-operating system based on heterogeneous kernels: A novel approach based on containers and microservices. In *Proceedings of the International Conference on Big Data and*

Advanced Wireless Technologies, BDAW '16, pages 7:1–7:13, New York, NY, USA, 2016. ACM.

[33] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina. *Microservices: Yesterday, Today, and Tomorrow*, pages 195–216. Springer International Publishing, Cham, 2017.

[34] T. F. Dullmann and A. van Hoorn. Model-driven generation of microservice architectures for benchmarking performance and resilience engineering approaches. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, ICPE '17 Companion, pages 171–172, New York, NY, USA, 2017. ACM.

[35] C. Esposito, A. Castiglione, C. A. Tudorica, and F. Pop. Security and privacy for cloud-based data management in the health network service chain: a microservice approach. *IEEE Communications Magazine*, 55(9):102–108, 2017.

[36] C. Y. Fan and S. P. Ma. Migrating monolithic mobile application to microservice architecture: An experiment report. In *2017 IEEE International Conference on AI Mobile Services (AIMS)*, pages 109–112, June 2017.

[37] M. Fowler. Microservices resource guide, 2016. http://martinfowler.com/microservices.

[38] C. Gadea, M. Trifan, D. Ionescu, and B. Ionescu. A reference architecture for real-time microservice api consumption. In *Proceedings of the 3rd Workshop on CrossCloud Infrastructures & Platforms*, CrossCloud '16, pages 2:1–2:6, New York, NY, USA, 2016. ACM.

[39] D. Golembeski, R. Forziati, B. George, and D. Sherman. Pipelinex: A feature animation pipeline on microservices. In *Proceedings of the ACM SIGGRAPH Digital Production Symposium*, DigiPro '17, pages 1:1–1:4, New York, NY, USA, 2017. ACM.

[40] G. Granchelli, M. Cardarelli, P. D. Francesco, I. Malavolta, L. Iovino, and A. D. Salle. Microart: A software architecture recovery tool for maintaining microservice-based systems. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 298–302, April 2017.

[41] G. Granchelli, M. Cardarelli, P. D. Francesco, I. Malavolta, L. Iovino, and A. D. Salle. Towards recovering the software architecture of microservice-based systems. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 46–53, April 2017.

[42] D. Guo, W. Wang, G. Zeng, and Z. Wei. Microservices architecture based cloudware deployment platform for service computing. In *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 358–363, March 2016.

[43] D. Guo, W. Wang, J. Zhang, G. Zeng, Q. Xiang, and Z. Wei. Towards cloudware paradigm for cloud computing. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pages 164–171, June 2016.

[44] H. Harms, C. Rogowski, and L. Lo Iacono. Guidelines for adopting frontend architectures and patterns in microservices-based systems. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 902–907, New York, NY, USA, 2017. ACM.

[45] S. Haselbock and R. Weinreich. Decision guidance models for microservice monitoring. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 54–61, April 2017.

[46] S. Haselböck, R. Weinreich, and G. Buchgeher. Decision guidance models for microservices: Service discovery and fault tolerance. In *Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems*, ECBS '17, pages 4:1–4:10, New York, NY, USA, 2017. ACM.

[47] S. Hassan, N. Ali, and R. Bahsoon. Microservice ambients: An architectural meta-modelling approach for microservice granularity. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 1–10, April 2017.

[48] S. Hassan and R. Bahsoon. Microservices and their design trade-offs: A self-adaptive roadmap. In *2016 IEEE International Conference on Services Computing (SCC)*, pages 813–818, June 2016.

[49] W. Hasselbring and G. Steinacker. Microservice architectures for scalability, agility and reliability in e-commerce. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 243–246, April 2017.

[50] F. Haupt, F. Leymann, and K. Vukojevic-Haupt. Api governance support through the structural analysis of rest apis. *Computer Science - Research and Development*, Sep 2017.

[51] R. Heinrich, A. van Hoorn, H. Knoche, F. Li, L. E. Lwakatare, C. Pahl, S. Schulte, and J. Wettinger. Performance engineering for microservices: Research challenges and directions. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, ICPE '17 Companion, pages 223–226, New York, NY, USA, 2017. ACM.

[52] J. Innerbichler, S. Gonul, V. Damjanovic-Behrendt, B. Mandler, and F. Strohmeier. Nimble collaborative platform: Microservice architectural approach to federated iot. In *2017 Global Internet of Things Summit (GIoTS)*, pages 1–6, June 2017.

[53] B. Jambunathan and Y. Kalpana. Multi cloud deployment with containers. *International Journal of. Engineering and Technology (IJET)*, 8(1):2319–8613, 2016.

[54] N. Josuttis. *Soa in Practice*. O'Reilly, 2007.

[55] H. Kang, M. Le, and S. Tao. Container and microservice driven design for cloud infrastructure devops. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 202–211, April 2016.

[56] K. Khanda, D. Salikhov, K. Gusmanov, M. Mazzara, and N. Mavridis. Microservice-based iot for smart buildings. In *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 302–308, March 2017.

[57] M. Khemaja. Domain driven design and provision of

micro-services to build emerging learning systems. In *Proceedings of the Fourth International Conference on Technological Ecosystems for Enhancing Multiculturality*, TEEM '16, pages 1035–1042, New York, NY, USA, 2016. ACM.

[58] T. Kiss, P. Kacsuk, J. Kovacs, B. Rakoczi, A. Hajnal, A. Farkas, G. Gesmier, and G. Terstyanszky. MicadoâĂŤmicroservice-based cloud application-level dynamic orchestrator. *Future Generation Computer Systems*, 2017.

[59] S. Klock, J. M. E. M. V. D. Werf, J. P. Guelen, and S. Jansen. Workload-based clustering of coherent feature sets in microservice architectures. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 11–20, April 2017.

[60] H. Knoche. Sustaining runtime performance while incrementally modernizing transactional monolithic software towards microservices. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, ICPE '16, pages 121–124, New York, NY, USA, 2016. ACM.

[61] Z. Kozhirbayev and R. O. Sinnott. A performance comparison of container-based technologies for the cloud. *Future Generation Computer Systems*, 68(Supplement C):175–182, 2017.

[62] N. Kratzke and P.-C. Quint. Understanding cloud-native applications after 10 years of cloud computing - a systematic mapping study. *Journal of Systems and Software*, 126:1–16, 2017.

[63] A. Krylovskiy, M. Jahn, and E. Patti. Designing a smart city internet of things platform with microservice architecture. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 25–30, Aug 2015.

[64] V. D. Le, M. M. Neff, R. V. Stewart, R. Kelley, E. Fritzinger, S. M. Dascalu, and F. C. Harris. Microservice-based architecture for the nrdc. In *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, pages 1659–1664, July 2015.

[65] A. F. Leite, V. Alves, G. N. Rodrigues, C. Tadonki, C. Eisenbeis, and A. C. M. A. d. Melo. Dohko: an autonomic system for provision, configuration, and management of inter-cloud environments based on a software product line engineering method. *Cluster Computing*, 20(3):1951–1976, Sep 2017.

[66] P. Leitner, J. Cito, and E. StÃűckli. Modelling and managing deployment costs of microservice-based cloud applications. In *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, pages 165–174, Dec 2016.

[67] A. L. Lemos, F. Daniel, and B. Benatallah. Web service composition: A survey of techniques and tools. *ACM Comput. Surv.*, 48(3):33:1–33:41, Dec. 2015.

[68] X. Li, K. Li, X. Pang, and Y. Wang. An orchestration based cloud auto-healing service framework. In *2017 IEEE International Conference on Edge Computing (EDGE)*, pages 190–193, June 2017.

[69] J. Lin, L. C. Lin, and S. Huang. Migrating web applications to clouds with microservice architectures. In *2016 International Conference on Applied System Innovation (ICASI)*, pages 1–4, May 2016.

[70] D. Liu, H. Zhu, C. Xu, I. Bayley, D. Lightfoot, M. Green, and P. Marshall. Cide: An integrated development environment for microservices. In *2016 IEEE International Conference on Services Computing (SCC)*, pages 808–812, June 2016.

[71] K. B. Long, H. Yang, and Y. Kim. Icn-based service discovery mechanism for microservice architecture. In *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 773–775, July 2017.

[72] A. Longo and M. Zappatore. A microservice-based mool in acoustics addressing the learning-at-scale scenario. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 391–400, July 2017.

[73] D. Lu, D. Huang, A. Walenstein, and D. Medhi. A secure microservice framework for iot. In *2017 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 9–18, April 2017.

[74] D. Malavalli and S. Sathappan. Scalable microservice based architecture for enabling dmtf profiles. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 428–432, Nov 2015.

[75] B. Mayer and R. Weinreich. A dashboard for microservice monitoring and management. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 66–69, April 2017.

[76] J. McKendrick. 3 situations where SOA may be preferable to microservices, 2017. http://www.zdnet.com/article/3-situations-where-soa-may-be-preferable-to-microservices.

[77] K. Meinke and P. Nycander. *Learning-Based Testing of Distributed Microservice Architectures: Correctness and Fault Injection*, pages 3–10. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[78] A. Melis, S. Mirri, C. Prandi, M. Prandini, P. Salomoni, and F. Callegati. Integrating personalized and accessible itineraries in maas ecosystems through microservices. *Mobile Networks and Applications*, Feb 2017.

[79] S. Newman. Building microservices : designing fine-grained systems, 2015.

[80] P. Nguyen and K. Nahrstedt. Monad: Self-adaptive micro-service infrastructure for heterogeneous scientific workflows. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 187–196, July 2017.

[81] P. Nicolaescu, G. Toubekis, and R. Klamma. *A Microservice Approach for Near Real-Time Collaborative 3D Objects Annotation on the Web*, pages 187–196. Springer International Publishing, Cham, 2015.

[82] C. Pahl and P. Jamshidi. Microservices: A systematic mapping study. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1 and 2*, CLOSER 2016,

pages 137–146, Portugal, 2016. SCITEPRESS - Science and Technology Publications, Lda.

[83] A. Panda, M. Sagiv, and S. Shenker. Verification in the age of microservices. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS '17, pages 30–36, New York, NY, USA, 2017. ACM.

[84] S. Patanjali, B. Truninger, P. Harsh, and T. M. Bohnert. Cyclops: A micro service based approach for dynamic rating, charging amp; billing for cloud. In *2015 13th International Conference on Telecommunications (ConTEL)*, pages 1–8, July 2015.

[85] R. Peinl, F. Holzschuher, and F. Pfitzer. Docker cluster management for the cloud - survey results and own solution. *Journal of Grid Computing*, 14(2):265–282, Jun 2016.

[86] R. Petrasch. Model-based engineering for microservice architectures using enterprise integration patterns for inter-service communication. In *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 1–4, July 2017.

[87] F. Piccialli, P. Benedusi, and F. Amato. S-intime: A social cloud analytical service oriented system. *Future Generation Computer Systems*, 2016.

[88] M. Pistore, P. Traverso, P. Bertoli, and A. Marconi. Automated synthesis of composite bpel4ws web services. In *IEEE International Conference on Web Services*, ICWS '05, pages 293–301, Washington, DC, USA, 2005. IEEE Computer Society.

[89] F. Rademacher, S. Sachweh, and A. ZÃijndorf. Differences between model-driven development of service-oriented and microservice architecture. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 38–45, April 2017.

[90] M. Rahman and J. Gao. A reusable automated acceptance testing architecture for microservices in behavior-driven development. In *2015 IEEE Symposium on Service-Oriented System Engineering*, pages 321–325, March 2015.

[91] A. Ranjbar, M. Komu, P. Salmela, and T. Aura. Synaptic: Secure and persistent connectivity for containers. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 262–267, May 2017.

[92] J. Rao, P. Küngas, and M. Matskin. Composition of semantic web services using linear logic theorem proving. *Inf. Syst.*, 31(4-5):340–360, June 2006.

[93] M. Richards. *Microservices Vs. Service-oriented Architecture*. O'Reilly, 2015.

[94] D. Richter, M. Konrad, K. Utecht, and A. Polze. Highly-available applications on unreliable infrastructure: Microservice architectures in practice. In *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 130–137, July 2017.

[95] S. Rose, D. Engel, N. Cramer, and W. Cowley. Automatic keyword extraction from individual documents. *Text Mining: Applications and Theory*, pages 1–20, 2010.

[96] C. Rotter, J. Illes, G. Nyiri, L. Farkas, G. Csatari, and G. Huszty. Telecom strategies for service discovery in microservice environments. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 214–218, March 2017.

[97] J. Rufino, M. Alam, J. Almeida, and J. Ferreira. Software defined p2p architecture for reliable vehicular communications. *Pervasive and Mobile Computing*, 2017.

[98] D. I. Savchenko, G. I. Radchenko, and O. Taipale. Microservices validation: Mjolnirr platform case study. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 235–240, May 2015.

[99] W. Scarborough, C. Arnold, and M. Dahan. Case study: Microservice evolution and software lifecycle of the xsede user portal api. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, XSEDE16, pages 47:1–47:5, New York, NY, USA, 2016. ACM.

[100] A. Sheoran, X. Bu, L. Cao, P. Sharma, and S. Fahmy. An empirical case for container-driven fine-grained vnf resource flexing. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 121–127, Nov 2016.

[101] A. Sheoran, P. Sharma, S. Fahmy, and V. Saxena. Contain-ed: An nfv micro-service system for containing e2e latency. In *Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems*, HotConNet '17, pages 12–17, New York, NY, USA, 2017. ACM.

[102] J. Stubbs, W. Moreira, and R. Dooley. Distributed systems of microservices using docker and serfnode. In *2015 7th International Workshop on Science Gateways*, pages 34–39, June 2015.

[103] Y. Sun, S. Nanda, and T. Jaeger. Security-as-a-service for microservices-based cloud applications. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 50–57, Nov 2015.

[104] D. Taibi, V. Lenarduzzi, C. Pahl, and A. Janes. Microservices in agile software development: A workshop-based study into issues, advantages, and disadvantages. In *Proceedings of the XP2017 Scientific Workshops*, XP '17, pages 23:1–23:5, New York, NY, USA, 2017. ACM.

[105] T. Thiele, T. Sommer, S. Stiehm, S. Jeschke, and A. Richert. Exploring research networks with data science: A data-driven microservice architecture for synergy detection. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pages 246–251, Aug 2016.

[106] L. P. Tizzei, M. Nery, V. C. V. B. Segura, and R. F. G. Cerqueira. Using microservices and software product line engineering to support reuse of evolving multi-tenant saas. In *Proceedings of the 21st International Systems and Software Product Line*

Conference - Volume A, SPLC '17, pages 205–214, New York, NY, USA, 2017. ACM.

[107] G. Toffetti, S. Brunner, M. Blöchlinger, F. Dudouet, and A. Edmonds. An architecture for self-managing microservices. In *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud*, AIMC '15, pages 19–24, New York, NY, USA, 2015. ACM.

[108] V. Vernon. *Implementing Domain-Driven Design*. Addison-Wesley Professional, 1st edition, 2013.

[109] M. Vianden, H. Lichter, and A. Steffens. Experience on a microservice-based reference architecture for measurement systems. In *2014 21st Asia-Pacific Software Engineering Conference*, volume 1, pages 183–190, Dec 2014.

[110] N. Viennot, M. Lécuyer, J. Bell, R. Geambasu, and J. Nieh. Synapse: A microservices architecture for heterogeneous-database web applications. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, pages 21:1–21:16, New York, NY, USA, 2015. ACM.

[111] M. Villamizar, O. Garces, H. Castro, M. Verano, L. Salamanca, R. Casallas, and S. Gil. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *2015 10th Computing Colombian Conference (10CCC)*, pages 583–590, Sept 2015.

[112] M. Villamizar, O. Garces, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang. Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 179–182, May 2016.

[113] M. Villamizar, O. Garces, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang. Cost comparison of running web applications in the cloud using monolithic, microservice, and aws lambda architectures. *Service Oriented Computing and Applications*, 11(2):233–247, Jun 2017.

[114] P. Wizenty, J. Sorgalla, F. Rademacher, and S. Sachweh. Magma: Build management-based generation of microservice infrastructures. In *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*, ECSA '17, pages 61–65, New York, NY, USA, 2017. ACM.

[115] E. Wolff. *Microservices: Flexible Software Architectures*. CreateSpace Independent Publishing Platform, 2016.

[116] Z. Xiao, I. Wijegunaratne, and X. Qiang. Reflections on soa and microservices. pages 60–67, 2016.

[117] Y. Yu, H. Silveira, and M. Sundaram. A microservice based reference architecture model in the context of enterprise architecture. In *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, pages 1856–1860, Oct 2016.

[118] H. Zeiner, M. Goller, V. J. Expósito Jiménez, F. Salmhofer, and W. Haas. Secos: Web of things platform based on a microservices architecture and support of time-awareness. *e & i Elektrotechnik und Informationstechnik*, 133(3):158–162, Jun 2016.

[119] T. Zheng, Y. Zhang, X. Zheng, M. Fu, and X. Liu. Bigvm: A multi-layer-microservice-based platform for deploying saas. In *2017 Fifth International Conference on Advanced Cloud and Big Data (CBD)*, pages 45–50, Aug 2017.

[120] X. Zhu, X. Gong, and D. H. K. Tsang. The optimal macro control strategies of service providers and micro service selection of users: quantification model based on synergetics. *Wireless Networks*, Jan 2017.

[121] O. Zimmermann. Microservices tenets. *Computer Science - Research and Development*, 32(3):301–310, Jul 2017.

ABOUT THE AUTHORS:

Tomas Cerny received his Bachelor's, Master's, and Ph.D. degrees from the Faculty of Electrical Engineering at the Czech Technical University in Prague, and M.S. degree from Baylor University. He is a Professor of Computer Science at Baylor University. His area of research is software engineering, security, aspect-oriented programming and design, user interface engineering, enterprise application design and networking.

Michael “Jeff” Donahoo received his B.S. and M.S. degrees from Baylor University, and Ph.D. in the Computer Science from Georgia Institute of Technology. Jeff is currently a Professor of Computer Science at Baylor University where he conducts research on networking, security, and enterprise application development.

Michal Trnka obtained master degree in computer science from Faculty of Electrical Engineering of Czech Technical University in Prague and he is currently Ph.D. student in Czech Technical University in Prague. He received Fulbright scholarship for academic year 2017/2018. His area of interests is software engineering and especially in context aware applications security. Lately he focused on using context information to enhance security rules as well as methods to obtain context from trusted sources, like Internet of Things devices.