What Microservices Can Learn From Enterprise Information Integration

Georg-Daniel Schwarz
Computer Science Department
Friedrich-Alexander University
Erlangen-Nürnberg, Germany
georg.schwarz@fau.de

Dirk Riehle
Computer Science Department
Friedrich-Alexander University
Erlangen-Nürnberg, Germany
dirk@riehle.org

Abstract

Microservices are an architectural style in which each service typically provides the complete stack of functions from a user or application programming interface through a domain model all the way to storage for that model. As a consequence, querying conjunct data from different microservices becomes a non-trivial engineering task. In this article, we review older and established general data integration theory in the enterprise context and then compare current microservice practice with enterprise information integration (EII) theory as an established approach to data integration. We find that microservices do not utilize all possible approaches for data integration that are common in enterprises. Specifically, microservices use middleware only partially and databases are not used at all to integrate data. Therefore, we further investigate whether, when, and how these two approaches can be used in a microservices context and present our findings. With our findings, we (i) clear the way for fellow researchers to investigate and improve unused integration strategies with microservices and (ii) raise the awareness of practitioners that some integration strategies may not work out of the box with microservices as they do in EII.

1. Introduction

The microservice-based architectural style has been around for a few years now and is still gaining popularity, because microservices promise among other benefits to provide scalability, resilience and a free choice of technology [1]. Microservices are services that have an independent life-cycle, especially regarding deployment. They communicate with each other via lightweight mechanisms over a network as a distributed system, and are typically built around business capabilities. Often, domain-driven design (DDD) is used for determining boundaries between microservices [2]. Microservices cooperate in order to perform work that a monolithic application would do all by itself.

The alignment of microservices with business capabilities leads to each microservice having the responsibility for its data. Each service can bring its own persistence mechanism that fits the job best, for example, a relational database. Direct access to data of other services is seen as an improper design practice. Doing so threatens the independence of microservices and should be avoided [3].

However, there are scenarios where depending on data of another microservice is not avoidable. This implies that integrating microservices is a necessity and a common issue practitioners experience with microservices. While current literature emphasizes some approaches dealing with this issue, it remains silent regarding older more established data integration techniques.

Those unutilized techniques do have their advantages over the recommended ones in some aspects. For example, using databases to join data could provide a significant performance improvement. This leads to the following research question:

> **RQ:** What can microservices learn from enterprise information integration?

Therefore, we conduct a gap analysis that compares established data integration approaches known from enterprise information integration (EII) to current approaches of microservices. The contributions of this paper are the following:

- The development of a model focusing on system architecture aspects of EII. This model is suitable for a gap analysis in order to compare EII to microservice data integration practices. (Section 4)
- The determination of a list of best-possible mechanisms for integrating microservices by a literature analysis. (Section 5)
- The analysis whether the developed EII model adequately captures the data integration approaches of microservices or has to be adjusted to the context of microservices. (Section 6)

URI: https://hdl.handle.net/10125/64420
978-0-9981331-3-3

With our findings we (i) clear the way for fellow researchers to investigate and improve unused integration strategies with microservices and (ii) raise the awareness of practitioners that some integration strategies may not work out of the box with microservices as they do in EII.

Section 2 gives an overview of the related work. Section 3 covers the overall research process of the article. We develop an analysis model for EII in Section 4, define a target for the model in Section 5 and finally apply the model on the microservices domain in Section 6. We discuss limitations of the results in Section 7 and give an outlook on future work in Section 8.

2. Related Work

2.1. Data Integration and EII

Lenzerini [4] defined data integration as the integration of data from multiple sources to generate a unified view. In the context of this paper, these data sources are controlled by different applications and the unified view is required to add some business capabilities to the application environment. In other words, we aim for a query that requires data that is under the control of different applications.

In the enterprise environment data integration without making use of centralized data integration systems like data warehouses is called enterprise information integration (EII) [5].

A larger field in data integration research is about semantic data integration and describes mappings from data of one source to data of another source [4]. Contextual knowledge is necessary to combine data from various sources according to those mappings. For example, we need to know that the attributes 'surname' and 'last_name' of customer data from different sources mean the same thing. This enables creating a query mechanism that includes data from both sources. This point of view regarding data integration is important and cannot be neglected, though we want to focus on an architectural point of view in this article.

2.2. Microservices

Microservices are an architectural style that emerged from industry. There is no standardized definition of this term, but there are common characteristics that microservice architectures share. Literature exists plenty regarding this topic, especially practitioner books due its industry origin. Parts of the literature are presented in the literature review in Section 5.

We understand microservices as a set of independently deployable services that interact in a lightweight way over the network [2]. This decomposition of applications into smaller ones that work together implicitly means that data integration is required more frequently. According to Lewis and Fowler [2], those services are organized around business capabilities. Cross-cutting functionalities that require data from multiple business capabilities raise the need for data integration. There is a natural need for thinking more about integration because "if you're building more services you end up with more integration problems" [6].

We identify two general types of interaction between microservices: Triggering distributed behavior and querying distributed data. The separation of both concepts is derived from distinguishing writing and reading data. As an example, a webshop system could consist of microservices for the management of customers, orders, and deliveries. Triggering distributed behavior requires the action of multiple services. For example, a correct order should always trigger a delivery eventually. Querying distributed data means to combine data from multiple services. For example, the sales department wants to send 10-percent-off coupons to all customers that bought a product in the last 3 months as an advertisement. Data from the customer service has to be combined with data from the order service.

Both interactions are commonly seen in microservice-based architectures. Communication mechanisms can support both. In this paper, we want to focus on the second scenario where data of multiple microservices has to be combined in order to implement some cross-cutting functionality.

2.3. Microservice Data Integration

Data integration has not been investigated much in microservices research. We only found a smal number of articles that address aspects of this field of research.

Villaça, Azevedo and Baião [7] investigated how microservices can be used to query across polyglot persistence implementations. This querying aspect matches our understanding of data integration. They identified five major strategies: Shared Database Infrastructures, CQRS, Event Data Pumps, Data Retrieval via Service Calls and Canonical Data Models. Parts of these resemble our findings, some differ. For example, we found that (data) integration on the user interface level is a valid approach but did not identify canonical data models as one. They list the pros and cons for each of the strategies together with example implementations. Additionally, each approach is evaluated with characteristics of ISO/IEC 25010. In our paper, we focus on the comparison to established data integration models which covers another aspect of this topic.

Diepenbrock et. al. [8] proposed an ontology-

based metamodel that describes domain models in microservice-based architectures. This enables the description of semantics for relationships between domain models of microservices. This topic is related to semantic data integration while we focus on the architectural aspects of data integration in microservices-based architectures.

2.4. Microservices in the Enterprise

Some first publications investigate how microservices fit into the enterprise ecosystem. Some of them touch the data integration topic but don't present a detailed analysis on which approaches of the established solutions present in the enterprise environment are or are not utilized in microservices as we do.

For example, Yu et. al. [9] introduce a reference architecture based on microservices "for implementing and managing enterprise microservices in the context of enterprise architecture". They describe building blocks that are necessary for such an undertaking, highlight architectural issues, and provide corresponding recommendations. The classical enterprise service bus (ESB) has to make way for "smart endpoints". A closer comparison of viable (data) integration approaches is not discussed; however, it is what we focus on.

Bogner and Zimmermann [10] extend original enterprise architecture reference models to allow for the integration of microservices. However, there is no discussion which specific ways of integrating microservices are viable. In contrast, we present these ways.

Indrasiri and Siriwardena [11] published a practitioner book on microservice in the enterprise and cover the approaches to sharing data between microservices as well. They focus on the microservice point of view on this topic, backed up with experience, while we perform a gap analysis that determines differences between microservice practices and established approaches of the enterprise ecosystem.

3. Research Process

This article compares established enterprise information integration (EII) theory with recommended practices of microservices regarding data integration. Figure 1 shows our overall research process. We first develop a comparison model that describes EII. Then we review microservice literature and analyze existing concepts in the sampled literature. Finally, we compare our findings with the comparison model from the first step. We present the results of the steps in the following way:

- Section 4 selects and vets a model of EII and develops a comparison model that is used for a gap analysis with the microservice domain later on.
- Section 5 reviews existing practitioner literature on data integration for microservices that has the highest impact. We discuss the concepts we find.
- Section 6 applies the EII model of Section 4 to the domain of microservices discussed in Section 5. The application of the model serves as a gap analysis that makes the differences between the approaches visible.

In the following subsections, we provide further details about the microservice literature sampling in Subsection 3.1 and about the concept analysis in Subsection 3.2.

3.1. Literature Sampling

We first performed a two-step literature review on microservice integration and then compared the results to established data integration theory. Since microservices emerged from practical experience, we included most cited grey literature (practitioner books, blog posts, magazine articles, expert interviews, etc.) as sources. Their claims about data integration approaches don't have to be backed up by science but by experience. We assume that most cited literature on this topic represents the popular opinions that we want to investigate in this paper.

We can integrate data on every architectural level of a system [12]. We can further narrow this down to the architectural levels where communication over the network takes place in the microservice domain. Thus, we decided to search for strategies on how to integrate microservices in general, without constraining it to data integration. Taking this detour instead of focusing on data integration itself from the beginning turned out to be a necessity, because there hardly exists literature about data integration in the microservice domain like Section 2 shows.

For the first step, we used Google Scholar, ACM Digital Library, and IEEE Xplore. We utilized the software Publish or Perish (version 6.48.6402) to scan Google Scholar. As search term, 'microservices' in different writing styles in the title in combination with the terms 'data integration' was used. We followed the general recommendations of Garousi et. al. [13] on how to conduct a grey literature review explained by using theoretical saturation as stopping criteria. Grey literature is included in Google Scholar search and thus requires a quality assessment criteria. In order to find literature with the most impact on the general understanding of microservice-based architectures, we decided to use the number of citations as the main impact criterion and include only grey literature within the 20 most cited works

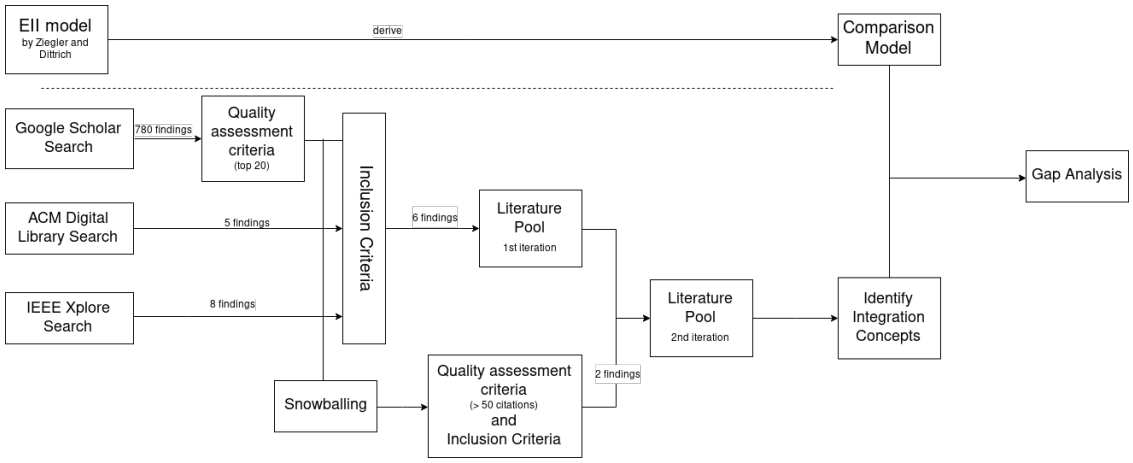

Figure 1. Research Process

of the search query result or if they are cited over fifty times regarding the second step of the literature review.

ACM Digital Library and IEEE Xplore only contain peer-reviewed literature and thus need no such quality assessment criteria.

We analyzed the findings and categorized them according to the following points and only included the ones into the literature pool that passed the inclusion criteria.

Inclusion criteria:

- Discussions about different approaches of integrating microservices and how they communicate

Exclusion criteria:

- Descriptions of microservice-based systems, but no detailed discussions about how they are integrated
- Literature that does not express own opinions and relies on citation instead of reasoning. We used those primarily for snowballing in the second step of the research process.

As the second step, snowballing and further exploratory search techniques were applied. For snowballing, we especially focused on literature that was described in literature reviews and mapping studies [14, 15, 7]. We applied the mentioned quality assessment criteria for grey literature (more than 50 citations) and inclusion criteria for all additional findings.

Section 5.1 presents the results of the literature sampling.

3.2. Literature Analysis

We analyzed the literature that we found during the sampling process for general integration strategies and grouped them into categories. Therefore, we transitioned from an author-centric to a concept-centric literature review by using a concept matrix as Webster and Watson [16] propose. We enhanced this approach by using abbreviations in the cells of the tables in order to provide more detailed information about our findings.

We focused on the architectural point of view and scanned the literature for conceptual approaches regarding data integration that has an impact on the architectural layer. The results are used for the gap analysis comparing with established EII.

Section 5.2 presents the results of this analysis.

4. EII Analysis Model

As the first step, we need to define what data integration means in the enterprise ecosystem. Therefore, we develop a model that will be applied to the microservice domain in Section 6. Subsection 4.1 presents the originally selected EII model and Subsection 4.2 introduces slight changes in order to develop a suiting model for the gap analysis.

4.1. Original EII Model

We reviewed multiple models for integrating applications and chose the one that seemed to fit best for our comparison. Some of them were not about architecture directly. For example, Schwinn and Schelp [17] proposed patterns classified by dimensions as redundancy and direct or indirect data access. Land and Crnkovic [18] distinguished integrating software systems between import and export facilities, enterprise application integration (EAI), integration on data level, and integration on a source code level. Still, the architectural perspective was not represented well.

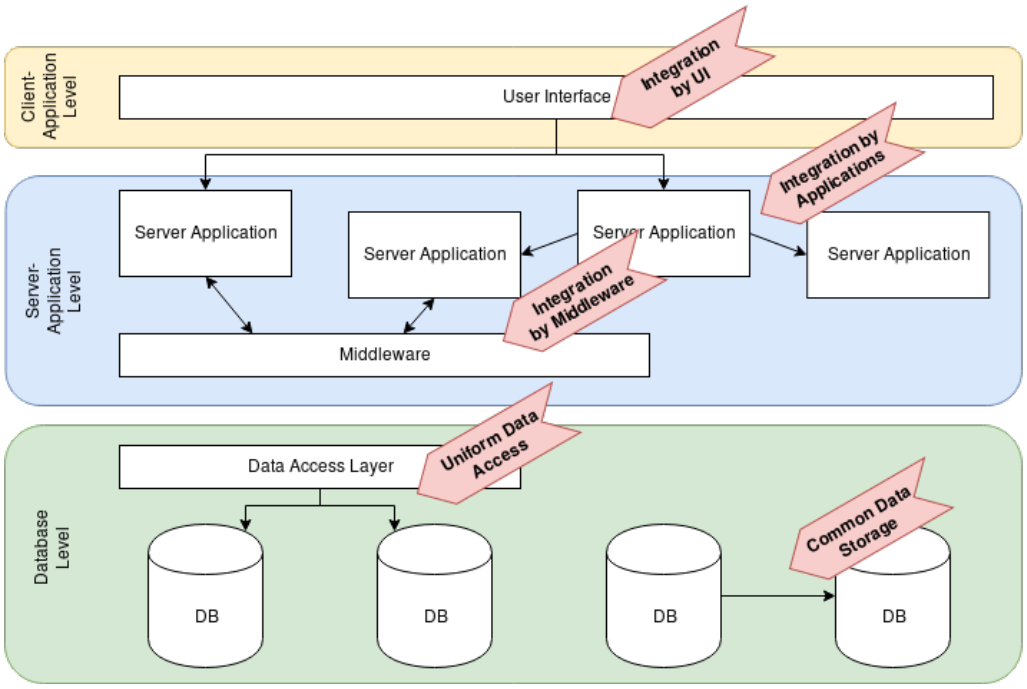

Figure 2. Derived Data Integration Model

A promising candidate was the layered model of He and Da Xu [19]. They found that research for integrating distributed enterprise systems can be divided into the communication layer integration, (semantic) data integration, business logic integration, presentation layer integration. They also mention more recent technologies like using an enterprise service bus (ESB). Nevertheless, these layers represent points that have to be thought of when integrating systems but not different levels in architecture where data integration can take place.

Ziegler and Dittrich [12] proposed a (for us) good starting point for the model required for the gap analysis. They claim that data integration in the enterprise ecosystem can take place on every architectural level of a system. The model describes how information systems with physically distributed data can integrate so that it seems to the outside world like there is only one information system.

There are six ways of integrating data derived from the architectural abstraction levels. The first two refer to manual data integration performed by the user, the other four describe a programmatic approach:

Manual Integration: Users directly work with the different interfaces of systems and integrate the data by hand. They need to know where the data is located, how it is represented logically, and how the semantics are defined.

Common User Interface: There is a common user interface, but the data presentations of different data sources are separated from each other. The user still has to homogenize and integrate data by hand, but does not have to use different user interfaces.

Integration by Applications: Integration applications access different data sources in order to return integrated results to the user or other clients.

Middleware Integration: The middleware undertakes the common integration functionality, but integration efforts are still required in the application.

Uniform Data Access: There is a global unified view of physically distributed data that is used to query the data. The integration is performed at runtime.

Common Data Storage: Data is physically transferred to a new data storage in order to integrate it.

4.2. Derived Comparison Model

We agree with the statement that data integration can potentially take place on every level of the architecture [12]. The proposed reference architecture with these architectural levels is suitable, but there are some open points that need clarification and enhancement.

Manual data integration is undesirable in our opinion. This aspect is found in the **Manual Integration** and in the **Common User Interface** approach. Because of that, we decided to ignore these two ways of integration in our discussions.

The original model covers integration on the user interface level in a manual way, but we are missing a pro-

grammatic approach for it. In the context of modern web applications, a client application is capable of interacting with multiple applications and combining the data. We assume that the authors placed this scenario in the Integration by Applications approach. For our gap analysis, we want to differentiate between integration happening on the client or on server side. Thus, we add the programmatic data integration on the user interface layer under the name **Integration by UI**. We locate it in the group of **Client-Application Level Integration**.

Integration by Applications is a way of integrating data on the server side like described in the last paragraph. Ziegler and Dittrich [12] mention that this approach is applicable for integrating a small number of applications. As integration applications grow in size, the more interfaces they have to integrate. In our opinion, there is a special case hidden in this approach where one of the source applications is the integrating one at the same time. This means that this application integrates data of a foreign application with its own data. In regards to communication mechanisms, we assume that different forms of remote procedure calls (for example, RPC, SOAP, or REST) are typical examples, because location information about the receiver is required.

In **Middleware Integration** the opposite is the case. Location information is typically not required by the sender. The middleware deals with the delivery to the right place. We see message-oriented middlewares as a good example, where applications can subscribe to topics or channels of messages they are interested in. Some implementations even offer to add message or protocol transformation features for better interoperability. The description of the original model introduces SQL-middleware as an example that routes the queries to all components that are involved to retrieve the expected result. The integration of the results of different components, however, is not done by the SQL-middleware but by the receiving application [12]. We find that middleware integration operates on a similar level of abstraction as integration by application but can contain some parts of the data integration logic. We summarize both of them under the group of **Server-Application Level Integration**.

Uniform Data Access takes the example of the SQL-middleware to the next level by moving the integration logic to the database layer. This requires a global virtual view over all data that is to be integrated in order to resolve queries and deliver partial queries to the right physical place. The location of physical data is not changed and remains distributed. This point changes when speaking about **Common Data Storage**. The data is physically transferred in order to integrate. We find that a suitable example is a DBMS that replicates data over several nodes in order to achieve spatial proximity for queries. An edge case of this approach is that applications share the same physical database and integrate over it. We summarize both approaches under the group of **Database Level Integration**.

The resulting model in Figure 2 resembles the typical structure of web-applications. There is an underlying database layer, a backend layer with all required server applications and a frontend layer with the user interfaces. When speaking about data integration in the backend layer, we make a differentiation between two basic communication mechanisms. One is to communicate via an API, therefore the location of the other participant in the interaction has to be known. The other is to make use of a middleware that distributes messages to the right place without having the sender to know the location of the other participant and performing add further logic on the messages. On the database layer, we distinguish between the physical and the virtual integration approach.

5. Microservice Data Integration

In the previous section, we developed a model for EII. In this section we examine how microservices integrate data. The results will be used as a target for the application of the EII model in Section 6.

Subsection 5.1 describes the sampled literature following the process of Section 3.1. The results of the concept analysis defined in Section 3.2 are presented in Subsection 5.2.

5.1. Literature Sampling Results

We applied the two-step literature sampling process described in Subsection 3.1. Table 1 shows the final list of literature we used for our further analysis.

We considered only Villaca et. al. [7] from searching ACM digital library and IEEE Xplore. We excluded the paper in the end because it evaluates different strategies regarding querying data from multiple microservices while our defined inclusion criteria focus on discussions and opinions about integration approaches. Nevertheless, we considered the concepts of their research when refining our integration concepts. We excluded Diepenbrock et. al. [8] that we mentioned in Section 2.3 because it does not discuss different integration approaches on an architectural level.

The first six entries represent the results of the first step of the literature review; they all were found through Google Scholar. The last two entries in the table were added as a result of snowballing. Note that the blog article [2] was considered, because it has an aggregate citation count of over 400.

Table 1. Literature Pool

Ref	Author	Title	Ranking / Citations
[1]	Newman	Building microservices: Designing fine-grained systems	#1 / 935 citations
[6]	Thönes	Microservices	#4 / 212 citations
[20]	Wolff	Microservices: Flexible software architecture	#9 / 80 citations
[21]	Richards	Microservices vs. service-oriented architecture	#11 / 78 citations
[22]	Butzin et. al.	Microservices approach for the internet of things	#16 / 55 citations
[23]	Sill	The design and architecture of microservices	#20 / 48 citations
[24]	Namiot et. al.	On micro-service architecture	- / 217 citations
[2]	Fowler and Lewis	Microservices a definition of this new architectural term	- / over 400 citations

5.2. Literature Analysis Results

We analyzed the selected literature pool following the process of Subsection 3.2. Therefore, we identified integration concepts that focus on the architectural point of view. Villaca et. al. [7] gave us a good starting point for identifying concepts regarding integration in microservice-based architectures. We adapted and enhanced the concepts with the findings of grey literature.

Table 2 shows the identified concepts. Some literature made statements that apply to subcategories of a category and did not further distinguish between them. We illustrate this in Table 2 by joining corresponding cells. Especially the two books [1, 20] discuss almost the full bandwidth of solutions. The following summary to each integration concept is not meant to be thorough, detailed advantages and disadvantages can be read up in the particular literature.

C1 Integration by Database: Discussing integration on database level is not trivial. This approach not recommended in general because of the following disadvantages:

- Changes of the internal data model in one microservice can affect other microservices and thus either crash them or discourages change [1, 20, 22].
- One database technology is enforced across microservices, so developers cannot independently choose the best technology for their microservice [20].
- The database may represent a single point of failure. This is especially dangerous if the chosen database is not resilient [1].
- “The logic to perform the same sorts of manipulation to a customer may now be spread among multiple consumers” [1].

Changes to one microservice may affect those other microservices that directly access the tables of the first microservice. Wolff [20] claims that a replication-based approach to decouple microservices by involving a schema transformation is viable. Most often those transformations need custom implementations on other architectural levels, though. The approach still has the disadvantage of the restriction to one database technology for the microservices that are meant to be integrated. The other literature [1, 22] does not mention the possibility of data replication and generalize their statements without considering this approach.

In general, a well-defined interface should be used for integration [21]. Could a versioned view be such a well-defined interface, at least for reads? Indirect access to database content with some decoupling steps between seems not to be explored to its fullest yet from the point of view that the chosen literature gave.

C2 Integration by Service Calls: Calling other microservices by using their APIs and integrating the received data in a specific microservice is generally a viable choice. Richards [21] claims that ”[...] microservices architectures tend to rely on REST as their primary remote-access protocol [...]”. REST is recommended for use by Newman [1] and Wolff [20] explicitly, especially in connection with HATEOAS and its links between resources. Although REST has some drawbacks due to the use of HTTP as basis, “REST over HTTP is a sensible default choice for service-to-service interactions” [1].

The literature discusses remote procedure call (RPC) technologies for the integration of microservices as well, including SOAP. Those technologies are viable [1, 20], although they have some downsides [1]. There are very different RPC technologies, of which some are better suited, some worse. “The use of a separate interface definition can make it easier to generate client and server stubs for different technology stacks” [1]. Technologies that don’t restrict technology choice are favorable, e.g. like Apache Thrift [20] and Googles gRPC [23].

Generally, integration by service calls seems to be a supported choice according to examined literature. Sill [23] advances his opinion that there are “design trends

Table 2. Integration Concepts for Microservices

C1	Integration by Database
a	Direct Access
b	Indirect Access
C2	Integration by Service Calls
a	REST
b	RPC
C3	Integration by Middleware
a	ESB
b	Simple Messaging/Pub-Sub
C4	Integration by UI
a	Monolithic UI
b	Distributed UI

Article	Concepts							
	C1		C2		C3		C4	
	a	b	a	b	a	b	a	b
[1]	w	-	p	r	w	r	p	p
[6]	-	-	-	-	w	r	-	-
[20]	w	p	r	r	-	r	p	r
[21]	-	-	-	(r)	(w)	(r)	-	-
[22]	w	-	(r)		-	r	-	(r)
[23]	-	-	(r)		-	(r)	-	-
[24]	-	-	-	-	-	r	-	-
[2]	(w)	-	(r)		w	r	-	-

r = recommendation, w = warned / rejected,
p = partially viable, () = claimed without explanation,
- = not discussed

for cloud services that favor speed and responsiveness over human readability of the exchanged data and API calls". Various technological approaches will compete with established REST over HTTP in the future.

C3 Integration by Middleware: Middleware is a very broad category of ways to integrate applications. In microservice literature, the two mentioned ones are the enterprise service bus (ESB) from service-oriented architecture (SOA) and simple message brokers supporting publish-subscribe functionality. In the upcoming comparison, we assume that an ESB includes additional logic like message enhancements, message transformations or protocol transformations.

The asynchronous messaging-style of integration that enables choreography is frequently encouraged by literature. This comes with the fact that "event-driven applications [...] allow you to decouple compared to using point-to-point RPC the whole time" [6]. It also enables data replication via events from one microservice to another as an Event Data Pump [1]. Wolff [20] claims that good implementations of a publish-subscribe infrastructure can improve the resilience and scalability of a system. On the downside, these kinds of architectures have the potential to become more complex [1].

In comparison, the ESB known from SOA containing much more logic is rejected by most literature. One interviewee even claims he never saw any project succeed that incorporates a big central ESB [6]. The general mantra that most literature agrees to is "keep your middleware dumb, and keep the smarts in the endpoints" [1].

C4 Integration by UI: We found two general ways of integrating microservices on the UI level. One is a monolithic UI that makes API calls to backend services and integrates the data for presentation. Newman [1] calls this approach "API composition"; it enforces that API changes of microservices have to be combined with adjustments in the UI. The modularization of a UI decouples the components as far as possible. They could communicate with each other e.g. by utilizing events on this level [20].

The other general approach claims that "microservices should bring their own UI along with them. By having the UI included with the relevant microservice, changes to that microservice that affect the UI can be done in one place. It is then necessary to integrate the UIs of the microservices together to form the system as a whole" [20]. This can be achieved in multiple ways. Either linking HTML pages served by microservices, using skeletons that pull in other fragments via JavaScript for example or utilizing Edge Side Includes (ESI) or Server Side Includes (SSI) [20].

Some of the mentioned approaches provide closer integration, some come with more decoupling capabilities. It is a trade-off. Which one to choose depends on the use-case specific requirements. Regarding multiple UIs, there is the Backends for Frontends approach [1, 20]. Each UI has its own server-side aggregation endpoint or API gateway. It can lead to less chatty conversations and promises to fulfill the different requirements of UIs better [1].

6. Gap Analysis

This section uses the developed EII model of Section 4 and applies it to the microservice domain that Section 5 summarized. This last step is meant to make the differences between both approaches visible and perform a gap analysis.

Like already mentioned, we distinguish between triggering distributed behavior and querying data from microservices when speaking about integration. This implies that we have to focus on the case of distributed data aggregation from different microservices and ignore the distributed behavior case in order to make a comparison with data integration.

The structure of encountered integration concepts has similarities with approaches of established EII. We steer the discussion through every established EII approach and conclude whether the corresponding approach is recommended, rejected, or restrictedly viable in microservice-based architectures

Integration by UI turned out to be a **recommended** approach in microservice-based architectures under concept C4. There are more fine-grained technical solutions to this with each having its pros and cons. The method of implementing Backends for Frontends fits better into the next category, though.

Integration by Applications with different technologies is well represented and **recommended** in microservices. REST seems to be the most used technology, other RPC-like approaches are also viable. We found those under the concept C2 Integration by Service Calls.

Integration by Middleware is applicable with microservices within a certain degree as concept C3 shows. The recommendation is to keep the middleware free of domain logic. Publish-Subscribe messaging is frequently encouraged in literature for decoupling services. Other kinds of middleware are not described in investigated literature, like SQL middleware. We conclude this approach is present in microservice-based systems in a **restricted** way.

Unified Data Access and **Common Data Storage** are not distinguished in microservice literature but discussed under concept C1 Integration by Database. The general opinion is to avoid this kind of integration. Without spectating distributed behavior cases, the argument of spread manipulation logic across microservices can be neglected in this context. Moving write functionality to the database level like stored procedures could cope with this problem as well.

Two disadvantages of the database integration approach are connected to the technological choice of the database. We are certain that there are use-cases where restricting to one kind of database technology can be considered as good trade-off when gaining performance in return. Additionally, with the choice of a suitable database technology, we can invalidate the argument of lost resilience as a single point of failure.

The open point that remains is the danger to independent evolvability of microservice, especially the independent deployability. There are database techniques like schema evolution that could deal with this issue as well. We think as long as a view on the data exists that is maintained across versions of the microservice, as some kind of versioned interface like it is good practice in RESTful APIs as well, this approach has the potential to be a good alternative to the recommended approaches. Even though, literature **rejects** the approach to integrate on database level.

7. Limitations

In Section 4 we selected and vetted an EII model and developed it further in order to apply it for the comparison with approaches of microservices. However, in order to claim that the chosen model completely represents established data integration theory in the enterprise ecosystem, a comprehensive literature review is necessary that makes sure the developed model covers all the important aspects.

Our approach regarding data integration in microservice-based architectures is limited to a selective elicitation of discussions about the integration of microservices. In Section 5 we focus on grey literature by applying quality assurance criteria of citation count in order to identify literature that has the most impact. This assumption may lead to disregarding recent insights that have not yet achieved high citation counts.

8. Conclusion and Future Work

This article developed a model for EII focusing on system architecture in Section 4, reviewed microservice literature regarding integration in Section 5, and applied the developed model on those findings in Section 6.

With this approach, we showed that not the whole solution space that is open for data integration in the enterprise ecosystem is utilized by microservices. Microservices can utilize the UI for integrating data and presenting the result directly to the user. Another recommended approach is to use service calls in order to retrieve data and perform data integration in another microservice. Regarding data integration via middleware, only technologies that provide simple publish-subscribe functionality are recommended. Literature explicitly warns to

move integration logic like data or protocol transformations into a middleware. Integration on database level is rejected as well.

The recommendations and rejections we found in literature are backed up with experience and not by research. We think that there are scenarios where integrating microservices on the database level has advantages over the other approaches. The well-known fact, that data typically lives longer than applications is one of the indicators that this approach is worth investigating further. In our future work, we will propose patterns that deal with the negative implications of the integration approach on database level that literature explains and compare it to established approaches that microservices prefer.

Acknowledgements

We thank our colleagues, especially Ann Barcomb and Andreas Bauer, for proof reading and their comments that greatly improved the article, as well as six anonymous reviewers for sharing their insights for further improvements.

References

[1] S. Newman, *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.", 2015.

[2] M. Fowler and J. Lewis, "Microservices a definition of this new architectural term," *URL: http://martinfowler. com/articles/microservices. html*, 2014.

[3] D. Taibi and V. Lenarduzzi, "On the definition of microservice bad smells," *IEEE software*, vol. 35, no. 3, pp. 56–62, 2018.

[4] M. Lenzerini, "Data integration: A theoretical perspective," in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 233–246, ACM, 2002.

[5] A. Y. Halevy, N. Ashish, D. Bitton, M. Carey, D. Draper, J. Pollock, A. Rosenthal, and V. Sikka, "Enterprise information integration: successes, challenges and controversies," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pp. 778–787, ACM, 2005.

[6] J. Thönes, "Microservices," *IEEE software*, vol. 32, no. 1, pp. 116–116, 2015.

[7] L. H. Villaça, L. G. Azevedo, and F. Baião, "Query strategies on polyglot persistence in microservices," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pp. 1725–1732, ACM, 2018.

[8] A. Diepenbrock, F. Rademacher, and S. Sachweh, "An ontology-based approach for domain-driven design of microservice architectures," *INFORMATIK 2017*, 2017.

[9] 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)*, pp. 1856–1860, IEEE, 2016.

[10] J. Bogner and A. Zimmermann, "Towards integrating microservices with adaptable enterprise architecture," in *2016 IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW)*, pp. 1–6, IEEE, 2016.

[11] K. Indrasiri and P. Siriwardena, "Microservices for the enterprise," *Apress, Berkeley*, 2018.

[12] P. Ziegler and K. R. Dittrich, "Three decades of data integration — all problems solved?," in *Building the Information Society*, pp. 3–12, Springer, 2004.

[13] V. Garousi, M. Felderer, and M. V. Mäntylä, "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering," *Information and Software Technology*, vol. 106, pp. 101–121, 2019.

[14] 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)*, pp. 44–51, IEEE, 2016.

[15] O. Zimmermann, "Microservices tenets," *Computer Science-Research and Development*, vol. 32, no. 3-4, pp. 301–310, 2017.

[16] J. Webster and R. T. Watson, "Analyzing the past to prepare for the future: Writing a literature review," *MIS quarterly*, pp. xiii–xxiii, 2002.

[17] A. Schwinn and J. Schelp, "Design patterns for data integration," *Journal of Enterprise Information Management*, vol. 18, no. 4, pp. 471–482, 2005.

[18] R. Land and I. Crnkovic, "Software systems integration and architectural analysis-a case study," in *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, pp. 338–347, IEEE, 2003.

[19] W. He and L. Da Xu, "Integration of distributed enterprise applications: A survey," *IEEE Transactions on industrial informatics*, vol. 10, no. 1, pp. 35–42, 2012.

[20] E. Wolff, *Microservices: flexible software architecture*. Addison-Wesley Professional, 2016.

[21] M. Richards, *Microservices vs. service-oriented architecture*. O'Reilly Media, 2015.

[22] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the internet of things," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–6, IEEE, 2016.

[23] A. Sill, "The design and architecture of microservices," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 76–80, 2016.

[24] D. Namiot and M. Sneps-Sneppe, "On micro-services architecture," *International Journal of Open Information Technologies*, vol. 2, no. 9, pp. 24–27, 2014.