A Microservice-Based Reference Architecture for Digital Platforms in the Proteomics Domain

Marwin Shraideh[1](✉), Patroklos Samaras[2], Maximilian Schreieck[1], and Helmut Krcmar[1]

[1] Technical University of Munich, Boltzmannstr. 3, 85748 Garching bei München, Germany
{marwin.shraideh,maximilian.schreieck,helmut.krcmar}@tum.de
[2] Technical University of Munich, Emil-Erlenmeyer-Forum 5, 85354 Freising, Germany
patroklos.samaras@tum.de

Abstract. Proteomics holds huge innovations for healthcare such as personalized medicine to tremendously increase people's health. Due to its rapid growth, its multidimensional data sets and the related need for the latest technologies and huge computing capacities, a diverse and scattered tool and repository landscape evolved in an uncontrolled manner. Therefore, tool usage is complicated, time consuming and almost impossible without expert IT skills. To create the conditions for new innovations in the proteomics domain and making first steps towards personalized medicine, digital platforms are needed. However, designing such a system is complex and was not yet supported by information systems. Consequently, we will design and implement a sustainable microservice-based reference architecture for digital platforms in the proteomics domain based on the example of ProteomicsDB that focuses on maintainability, extendibility, and reusability by following the design science requirements. With our reference architecture, we extend evidence-based design knowledge in real-world information systems contributing to information systems research and provide proteomics researchers, but also practitioners a foundation for establishing new business models and enhancing existing or developing new services or platforms.

Keywords: Reference architecture · Proteomics · Digital platform · Microservices · Design science · Information systems engineering

1 Introduction

Proteomics has become increasingly important in the context of personalized medicine and healthcare [1–4]. "Proteomics refers to the study of proteomes, but is also used to describe the techniques used to determine the entire set of proteins of an organism or system, such as protein purification and mass spectrometry." [5]. Since "[p]roteins are the major effectors of cell functions" [6] in the human body, they reflect "the actual workforce of the cell and are thus highly suited for studying the mechanism of disease development and progression" [6]. Consequently, it is increasingly utilized in clinical practice to conduct proteomics analysis on individual patients to identify complex diseases such as cancer [7] more effectively, efficiently, and reliably, but also to offer a more effective and gentle personalized treatment by informing drug selection and dosage [1, 6, 8]. This is also known as personalized medicine. Personalized medicine customizes disease treatment and post-treatment based on individual differences of patients [9]. The goal of personalized medicine is to ultimately prevent diseases instead of having to react upon its occurrence resulting in an overall improvement of people's health [4]. For personalized medicine, biomarkers are key as they "are defined as objectively measured indicators of biological processes or response to a therapeutic intervention." [10]. They inform about disease risk and status as they reflect specific characteristics of a disease and allow searching for similar characteristics in a patient's sample to identify diseases easily and for appropriate drugs to treat the disease. Consequently, proteomics has the potential to bring new innovations to healthcare and medicine [4]. For example, it contributed to important insights to understand the COVID-19 virus and revealed potential therapy targets [11]. However, analyzing complex and multi-dimensional proteomcis data requires interdisciplinary expert knowledge and a specialized set of analytics tools that can be automized to a limited extent only.

For this reason and due to the proteomics rapid evolution in the past years, the landscape of proteomics data repositories and tools has grown in an uncontrolled manner making data integration and tool usage complicated, time consuming and almost impossible without expert IT skills. Furthermore, there is no guidance for developing new tools and repositories and no standards for easier tool and repository integration. Most tools and repositories are only created to serve the purpose of publishing results from proteomics research and thus rarely offer seamless reusability by other labs and institutions.

To create the conditions for new innovations in the proteomics domain and making first steps towards personalized medicine, information systems (IS), in particular digital platforms, are a potential solution to integrate different proteomics repositories and tools and to cope with the many related challenges. Here, IS research can help with its knowledge in handling big data and information system and service system design [12–17]. However, designing such a system is complex and was not yet supported by information systems and service (systems) engineering in the proteomics domain. Existing solutions rarely provide a deep insight into the architecture of these systems and tools.

Consequently, we will design and implement a reference architecture for digital platforms based on the example of ProteomicsDB. ProteomicsDB is a big data, multi-omics and multi-organism resource for life science research initially developed for hosting the first mass spectrometry-based draft of the human proteome [18, 19]. It is built upon the in-memory database platform of SAP HANA and publicly accessible. The many users of ProteomicsDB include people from both the academic and industrial setup. ProteomicsDB tries to evolve to a digital platform to solve aforementioned challenges and to provide a "one-stop-shop" for solutions in daily issues in life science research. Therefore, ProteomicsDB already offers first online analytics services for experiments as well as search and visualization functionalities to enable the online exploration, analysis, and comparison of (imported) protein expression datasets; for example, to find appropriate cell lines or drugs for an experiment or for hypothesis generation.

However, incomplete documentation as well as a missing guidance on how to create new features makes development of new features or endpoints in ProteomicsDB time consuming and arbitrary. This resulted in a complex landscape of island solutions that is hard to update and adapt to new challenges and utilizes existing features only to a limited extent hindering rapid integration of new tools and repositories.

With our reference architecture, we want to address these challenges and pitfalls by providing a clear structure and guidance that allows tools and repositories in the proteomics domain to stay maintainable, extendable, reusable and managable in the long run. It also allows a fast and unified development approach for new tools, repositories, and proteomics resources, such as ProteomicsDB, to evolve themselves to a digital platform that can integrate most important tools and repositories of the proteomics domain to provide a maximum of utility for proteomics research and enabling and accelerating new innovations in the proteomics domain.

We structured our paper as follows: First, we provide a brief introduction to microservice architectures and reference architectures in Sect. 2, then describe our research approach in Sect. 3 and present our results in Sect. 4. Finally, we conclude this paper by discussing our results and describing the limitations of our research.

2 Theoretical Background

2.1 Microservice Architectures

According to [20], microservices is an architectural style that aims at developing applications based on small services where each service has a single purpose. Microservices are well suited for big software systems [21]. It avoids software systems from becoming rigid, hard to maintain and extendable, and uneconomical due to growing functionalities and complexity [22]. Here, it pays special attention to interchangeability of single software components, interoperability of microservices and avoiding dependencies to other components [20, 23]. Due to its simple extendibility and concept of immutability (replacing components instead of adjusting them), it simplifies development while accepting an increasing complexity of the overall architecture which can be handled by additional tools and automatization [24]. Therefore, it is used by big tech companies such as Netflix, Google and other software platforms [25].

2.2 Reference Architectures

According to [26], a reference architecture is an abstract architecture, that simplifies the development of systems, solutions, and applications by providing knowledge and defining the frame for development. It is characterized by its (re)usable object or content that is used for constructing a concrete architecture for a system to develop. A reference architecture contains a technical focus but combines it with related domain knowledge. With its expression and content, it builds a common framework allowing to have detailed discussions of all stakeholders that are involved in development [26]. Especially in highly dynamic environments with specific requirements to integration in open systems, reference architectures support in coping with increasing complexity and scale [27]. They increase quality by using best practices and lower development cost and time due to reusability of modules of the reference architectures [28].

3 Research Approach

For designing our reference architecture, we follow the 8-step approach of [29] which is based on the design science requirements [30].

1. Defining the purpose and goal of the reference architecture
2. Literature Review
3. Situation and requirements from industry
4. Standardizing terms and creating a domain description
5. Extracting generic and optional requirements
6. Transferring identified requirements to n:1 functional and logical modules
7. Creating reference processes
8. Feedback cycles and implementation of reference architecture

Step one defines what aspects the reference architecture focuses on and how it is visualized and presented to be understandable for future users of the reference architecture. To do so, we used the classification schema of [29]. This phase is equivalent to the framing of research activities and determining the focus of the planned research. The focused aspects were selected in a way that they also accomplish the goals the reference architecture is built for. In step two, the current body of knowledge in research and available technologies must be searched to build a foundation for designing the reference architecture [29]. Since we focus on microservices, we conducted a literature review according to [31] and [32] to find the status quo of requirements towards designing, implementing, and maintaining microservices. The goal is to support the develop and build phase before starting with the first design science cycle by drawing from existing knowledge from previous information systems research about microservices and thus assuring a rigor design. In total, we found 52 unique requirements from 31 relevant articles out of 1439 after we coded the extracted requirements according to [33]. Goal of step three is to gather requirements from the proteomics domain and related industry [29]. Therefore, we conducted a qualitative study based on seven semi-structured expert interviews with 5 researchers, 3 developers and the professor of the Chair for Proteomics and Bioanalytics of the Technical University of Munich according to [34] to ensure relevance of our reference architecture for practice and the proteomics domain. Here, we focused on technical requirements rather than user requirements. Step four aims at standardizing descriptions of the domain and used terms to have a common understanding of sections and terms used in the reference architecture for all involved stakeholders [29]. Since we use terms common to microservices and stick to the terms and wording used in the interviews, step four does not need special attention. In step five, the results of step two and three must be aggregated to a list of unique requirements and separated in generic and optional requirements. Generic requirements are universally applicable and should be generally fulfilled. Optional requirements should be considered only for specific use cases [29]. We only aggregated our identified requirements to a list of unique requirements and did not differentiate between generic and optional requirements since we could not clearly separate them as we already focused on a narrowed set of requirements. Therefore, we considered all our requirements as generic requirements. Based on step five, every identified requirement must be transferred into one logical module in step six [29]. However, one module can fulfill multiple requirements, where every module must be a closed unit that is responsible for a specific purpose or functionality in the system according to the separations of concerns of [35]. If one requirement cannot be fulfilled by one module, it must be split up into multiple smaller requirements. These modules must be categorized as generic or optional based on how many generic or optional requirements they include [29]. As we decided to consider all requirements as generic in the previous step, we could only identify optional modules after implementing the prototype. All modules that were not implemented were categorized as optional. In total, we identified 20 modules. The derived modules and the identified requirements from the previous steps represent the foundation for designing the first version of our reference architecture in step seven. Lastly, in step eight, multiple feedback rounds must be conducted with developers and stakeholders in the context of requirements elicitation to improve the requirements and the reference architecture iteratively [29]. Since evaluation should be conducted as often as possible according to design science requirements [36], we integrated evaluation cycles where possible.

Once we designed the reference architecture and included requirements from literature and researchers, we finished the first design science iteration by conducting the phase "develop and evaluate". We then presented our first version of our reference architecture to developers of ProteomicsDB and considered their feedback in our second version. Afterwards, we implemented the second version of our reference architecture at the example of a prototype in ProteomicsDB. Here, the goal was to evaluate whether the designed reference architecture is applicable and usable. Furthermore, we wanted to get valuable insights from the implementation itself as well as feedback from experts for a second time to improve the overall design.

4 Results

For step one, we classified our reference architecture according to [29]. Our final reference architecture will have a mixed abstraction level, will be technology neutral, industry-specific and have a cross-product focus as well as a cross-company focus. Furthermore, we agreed upon that the reference character of our architecture will focus on researchers and developers in the proteomics domain as a reference point. There will also be variation points and the technical focus considers information from the proteomics domain. Therefore, our reference architecture is driven by research and practice. We also do not claim to design a holistic and complete reference architecture. The goal is to not only standardize the architecture of tools and repositories in the proteomics domain, but also to make it easier to use, extend, maintain, and integrate tools and repositories. For creating the reference architecture, we use a combination of an inductive and deductive approach [37]. With our reference architecture, we provide knowledge about the architecture and guidelines.

From the requirements we identified in step two and three, we only considered those from literature and the interview partners that were relevant for informing the reference architecture design. The only requirement of users was the possibility to use the same tool, feature or data set for multiple different purposes and fields for an interdisciplinary approach. Consequently, most important requirements came from the developers.

Figure 1 shows our reference architecture that was created in step seven and considers all 20 modules identified in step six based on step five and the feedback of all described evaluation cycles of step eight.

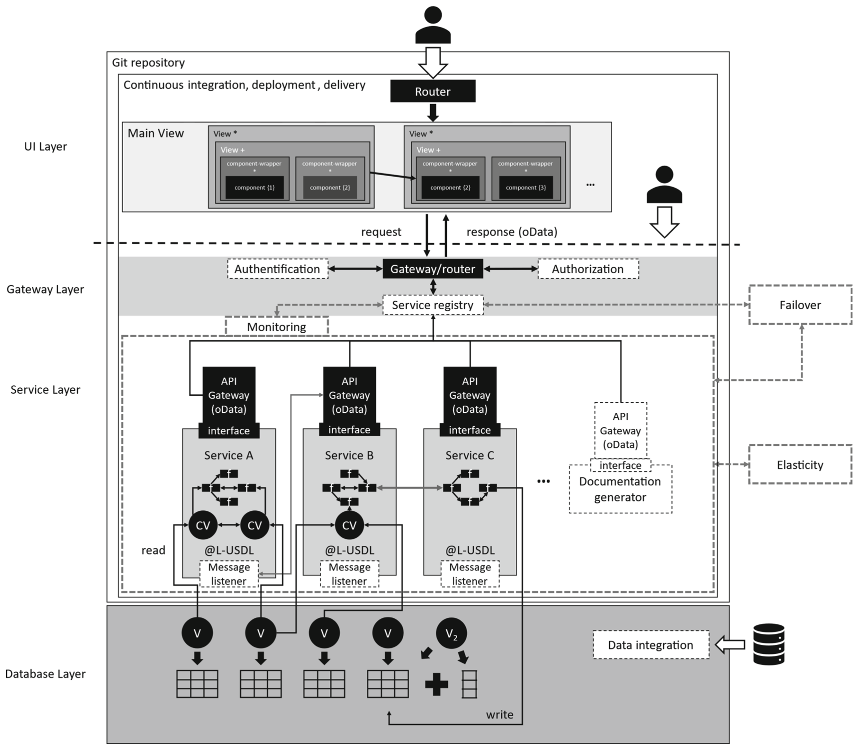

Fig. 1. Reference architecture

The reference architecture is divided in four layers: database layer, service layer, gateway layer, and UI layer. The database layer consists of the data schema and includes an abstraction layer which is demonstrated by the circles marked with a V, further referred to as view. These can be SQL views or other views that only allow read access to the tables so that database tables are not directly exposed to read operations. This layer also makes the database layer more flexible. Especially, when the data schema changes frequently as it is often the case in the proteomics domain. Since every view always provides the content of one table, it reflects the latest version of a table. Given the condition that columns in database tables and tables itself are not deleted and only new added and for every change in a table a new version of the view is created as depicted by V_2, it is possible to make changes in the database schema without affecting services that access older versions of the view. Therefore, extensibility and maintainability are increased since only new services must use the latest view version. Also, the effort for developers is decreased as they do not have to migrate all changes to the services that do not use this data (the additional column) if they would directly access the database table. This also allows a user-friendly versioning for the API since only major changes will affect older versions and therefore provide API users more time to migrate to the latest version. However, if an existing view needs the new data or must be updated due to important changes in the data schema, all services that access this view must be adjusted, too. As the possibility of integrating data is crucial in the proteomics domain, data integration must be possible. Ideally on the database layer directly due to performance reasons. Also, some databases come with preinstalled data integration tools that should be used because of its simplicity and fast implementation.

As shown in Fig. 1, the service layer can contain multiple microservices where each service (Service A, Service B, Service C and further) has a specific purpose, functionalities, and features that it also offers to other services enabling reusability and capsulation. Consequently, each microservice has its own set of small services and components which is highlighted in Fig. 1 by the circles marked with CV and the rectangles marked with f, further called customized views (CV) and feature (f). Ideally, customized views are created already in the database layer. There is no limitation in the number of customized views and features. However, each customized view and feature is only allowed to be part of a microservice if it serves the overall purpose and functionality of this microservice. If not, a new microservice must be created. Consequently, the number of customized views and features are naturally limited to a certain number that varies greatly. Customized views get the needed data from the database views and modify, limit, or filter the data as needed by the features or to provide data to serve the purpose of the microservice. Therefore, customized views can access multiple views and can have multiple hierarchies of reuse of customized views inside a microservice, as visualized by the arrows between the customized views in service A. The features of a microservice never directly access views. They always must access customized views to work with required data. As customized views, features can also have multiple hierarchies of reuse of other features. To share functionality and data with other microservices or even external users, each service has its own interface and standardized API gateway that messages over a standardized protocol such as oData. Therefore, every microservice can control which customized views or features it offers to other microservices via its interface and which of these customized views or features it offers to external users or services via its API Gateway by registering it in the service registry. To also wait for specific messages from other microservices in the service layer, every microservice has a message listener. This message listener allows to react upon messages sent by other microservices to also enable advanced automatization and communication between microservices. To keep an overview of the growing amount of microservices, their purpose, offered customized views and features, and their dependencies to other microservices, a proper documentation is crucial to keep them maintainable. Consequently, every microservice must be documented on code-level using standardized documentation languages such as unified service definition language (USDL) or linked unified service definition language (L-USDL). This allows to use an automated documentation generator that reads specific flags in the code of the microservices to collect information about their functionalities, offered data entities and analyzes the relationships in between microservices but also used database views. The collected information is provided to either internal developers or external API users to a restricted extent also via a standard API gateway. Goal of this documentation generator is to reduce the necessary effort to create documentations for microservices and the API manually after the implementation. It also prevents incomplete documentations already on code-level because continuous integration and deployment checks automatically detect missing or empty flags and prohibit productive deployment or merging a branch on the git repository.

Since not all microservices work dependent of other microservices, outages might not be recognized immediately. Therefore, a permanent monitoring of every microservice, its customized views, and features is required for availability by a microservice in the service layer. In case of an outage, failover must deploy a clone of the affected microservice and inform the service registry to reroute to the newly deployed microservice. If a microservice needs more or less computing power due to increased or decreased use, an elasticity service must assign hardware capacities appropriately. The failover service and the elasticity service are both external services that can manage the hardware directly. As mentioned before, the service registry holds all microservices and the customized views and features they offer for public use by the frontend or by external users. The Gateway/router represents the central component that can be accessed externally and receives all user and frontend requests. As all microservices have their own API, requests can simply be routed to the appropriate destination through the service registry once authentication was successful and authorization granted. The requested information is routed back to the Gateway/router by the microservice and the response is sent to the user interface or the user. In case of critical outages, the service registry is informed about the status of the requested resource and gets additional information from the failover service to inform the user accordingly.

The UI layer is separated from the overall complexity of the structure underneath the gateway layer through the API as the single point of access. The user interface (UI) itself consists of one main view and multiple views, component-wrappers and components. Components represent the smallest part of the UI and reflect visualization components, such as a table, bar chart or other types of charts that should be used once or multiple times. Therefore, each component exists only once, can only accept data, and provides status data about itself. Consequently, each component must come with one component-wrapper, that takes care of requesting data from the API, adjusting it as needed and handing its data over to the component. Also, one component-wrapper can only feed one component. A component-wrapper and a component are an independent unit that can be used and reused by any view making UI development more efficient. Since views follow the same principle, UI components can be built and reused in different context without having to write new code accelerating UI development.

Finally, a central router is responsible for navigating site requests to the respective view.

After implementing the reference architecture in ProteomicsDB, we resulted in the architecture shown in Fig. 1 without dotted elements. The most important requirements of the developers of ProteomicsDB were an overall design that heavily utilizes reuse of already implemented functionalities and features, sticking to one hardware stack and making maintainability and extendibility as easy and fast as possible with at least development effort possible. For this reason, we were not able to fulfill all microservice principles and adapted the architecture accordingly. For example, loosely coupled services with rather code duplications than dependencies between microservices for the sake of reusability. Also, failover and elasticity could not be realized due to the single hardware stack where one outage would affect all microservices. Consequently, monitoring is also not needed anymore since all microservice operations are tracked by the system. As ProteomicsDB is an open resource, we also left out authentication and authorization in the gateway layer. However, since authorization was already managed on the database level, all API requests are checked against the authorization of the database layer. Lastly, we also did not implement the message listener since it was not needed in the prototype. The documentation generator can be implemented any time since all microservices must be documented with L-USDL. According to the feedback from the developers, we met those requirements that are most important for fast development and easy maintainability with a small ruleset that does not limit a developer's coding flexibility.

5 Discussion and Limitations

The current reference architecture went through one complete iteration of a design science cycle with multiple evaluations with proteomics researchers and developers including interviews and feedback from prototyping. Therefore, it is a rigorous and relevant design artifact of design science [38]. However, we do not claim the reference architecture to be holistic and to consider all perspectives. We want to encourage to extend our reference architecture by using it as a foundation for further design science cycle iterations considering more perspectives from related stakeholders and information from the domain. For example, information from pharma companies or the already existing tool and repository landscape. Ultimately, creating a reference architecture for digital platforms in the proteomics domain that considers as much domain knowledge as possible to accelerate implementation of such platforms, overcome related challenges and pitfalls and thus enabling new innovations in the proteomics domain that improve the overall healthiness of people.

Proteomics requires huge computing capacities and therefore extraordinary expensive hardware which cannot be separated to virtualize the hardware and assign computing resources as needed. Consequently, a pure microservices architecture is not applicable in the proteomics domain due to the requirements of the domain, for practical applicability and economic reasons. Also, for the sake of maintainability, creating separated microservices that do not have the same code structure and are developed independently and individually, the problem of a landscape of island solutions that evolves uncontrolled cannot be solved. Especially in an environment where the development team changes every three to four years but the team-size remains constant over time, the effort of extending or maintaining existing services increases and becomes increasingly complex to a point where it cannot be handled anymore. Also because of the cross-expertise needed that cannot be built up under such circumstances. This led us to an interesting question for future research, whether following all microservice principles strictly is sustainable, manageable, and economic over the long run or only some of the most important principles are. Therefore, a more precise separation of strictly-to-follow and optional microservice requirements is necessary depending on the context they are implemented in.

With our research we contribute to both information systems research and proteomics research: We extended evidence-based design knowledge in real-world information systems by implementing and evaluating the designed microservice-based reference architecture in a cyclical design science approach for a digital platform in the proteomics domain [39, 40] and highlighting necessary adaptations. Our reference architecture guides proteomics and information systems researchers to systematically develop digital platforms in the proteomics domain in a unified way that also keeps its services maintainable, reusable, and adaptable in the long run. Proteomics or even bioinformatics and information systems researchers can use this reference architecture as a foundation for future research and a fast development blueprint. Pharma companies and startups can also benefit from these contributions by using the reference architecture for establishing new business models and enhancing existing or developing new services or platforms.

Acknowledgements. We thank the German Federal Ministry of Education and Research for funding this research as part of the project 031L0168 (DIAS). We explicitly thank Prof. Dr. Bernhard Küster, Prof. Dr. Mathias Wilhelm, Sascha Ladewig and all involved people for actively supporting our research.

References

1. Li, N., Zhan, X.: Signaling pathway network alterations in human ovarian cancers identified with quantitative mitochondrial proteomics. EPMA J. **10**(2), 153–172 (2019)
2. Duarte, T.T., Spencer, C.T.: Personalized proteomics: The future of precision medicine. Proteomes **4**(4), 29 (2016)
3. Parker, C.E., Borchers, C.H.: The special issue: clinical proteomics for precision medicine. Prot. Clin. Appl. **12**(2), 1600144 (2018)
4. Buriani, A., Fortinguerra, S., Carrara, M.: Clinical perspectives in diagnostic-omics and personalized medicine approach to monitor effectiveness and toxicity of phytocomplexes. In: Pelkonen, O., Duez, P., Vuorela, P.M., Vuorela, H. (eds.) Toxicology of Herbal Products, pp. 385–476. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-43806-1_16
5. Nature: Proteomics - Latest research and news (2021). https://www.nature.com/subjects/proteomics. Accessed 15 Mar 2021
6. Giudice, G., Petsalaki, E.: Proteomics and phosphoproteomics in precision medicine: applications and challenges. Brief. Bioinform. **20**(3), 767–777 (2019)
7. Bozorgi, A., Sabouri, L.: Osteosarcoma, personalized medicine, and tissue engineering; an overview of overlapping fields of research. Cancer Treat. Res. Commun. **27**, 100324 (2021)
8. Drew, L.: Pharmacogenetics: the right drug for you. Nature **537**, S60–S62 (2016)
9. ESF Forward Look: Personalised Medicine for the European Citizen. Towards more precise Medicine for the Diagnosis, Treatment and Prevention of Disease (iPM) (2012). http://archives.esf.org/fileadmin/Public_documents/Publications/Personalised_Medicine.pdf. Accessed 04 Apr 2021
10. Firestein, G.S.: A biomarker by any other name.... Nat. Clin. Pract. Rheumatol. **2**(12), 635 (2006)
11. Bojkova, D., et al.: Proteomics of SARS-CoV-2-infected host cells reveals therapy targets. Nature **583**, 469–472 (2020)
12. Chekfoung, T., Sun, L., Kecheng, L.: Big data architecture for pervasive healthcare: a literature review. In: European Conference on Information Systems (ECIS) 2015 Proceedings, Münster, Germany (2015).

13. Chen, T., Lu, P., Lu, L.: Design of ASD subtyping approach based on multi-omics data to promote personalized healthcare. In: Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS), Maui, Hawaii (2020)
14. Simons, L.P.A.: Health 2050: Bioinformatics for rapid self-repair; a design analysis for future quantified self. In: BLED 2020 Proceedings, Bled, Slovenia (2020)
15. Jarvenpaa, S., Markus, M.L.: Genetic platforms and their commercialization: three tales of digital entrepreneurship. In: Proceedings of the 51st Hawaii International Conference on System Sciences (HICSS). Hilton Waikoloa Village, Hawaii (2018).
16. de Reuver, M., Lessard, L.: Describing health service platform architectures: a guiding framework. In: Americas Conference on Information Systems (AMCIS) 2019 Proceedings, Cancún, Mexico (2019)
17. Vassilakopoulou, P., et al.: Building national eHealth platforms: the challenge of inclusiveness. In: International Conference on Information Systems (ICIS) 2017 Proceedings, Seoul, South Korea (2017)
18. Samaras, P., et al.: ProteomicsDB: a multi-omics and multi-organism resource for life science research. Nucleic Acids Res. **48**(D1), D1153–D1163 (2019)
19. Wilhelm, M., et al.: Mass-spectrometry-based draft of the human proteome. Nature **509**, 582–587 (2014)
20. Di Francesco, P., Malavolta, I., Lago, P.: Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. In: 2017 IEEE International Conference on Software Architecture (ICSA), Gothenburg, Sweden, pp. 21–30 (2017)
21. Josélyne, M.I., Tuheirwe-Mukasa, D., Kanagwa, B., Balikuddembe, J.: Partitioning microservices: a domain engineering approach. In: Proceedings of the 2018 International Conference on Software Engineering in Africa, Association for Computing Machinery, Gothenburg, Sweden, pp. 43–49 (2018)
22. Schwartz, A.: Microservices. Informatik-Spektrum **40**(6), 590–594 (2017). https://doi.org/10.1007/s00287-017-1078-6
23. Garriga, M.: Towards a taxonomy of microservices architectures. In: Cerone, A., Roveri, M. (eds.) SEFM 2017. LNCS, vol. 10729, pp. 203–218. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74781-1_15
24. Fu, G., Sun, J., Zhao, J.: An optimized control access mechanism based on micro-service architecture. In: 2018 2nd IEEE Conference on Energy Internet and Energy System Integration (EI2), Beijing, China, pp. 1–5. IEEE (2018)
25. Alshuqayran, N., Ali, N., Evans, R.: A systematic mapping study in microservice architecture. In: 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), Macau, China, pp. 44–51. IEEE (2016).
26. Reidt, A., Pfaff, M., Krcmar, H.: Der Referenzarchitekturbegriff im Wandel der Zeit. HMD Praxis der Wirtschaftsinformatik **55**(5), 893–906 (2018). https://doi.org/10.1365/s40702-018-00448-8
27. Cloutier, R., Muller, G., Verma, D., Nilchiani, R., Hole, E., Bone, M.: The concept of reference architectures. Syst. Eng. **13**(1), 14–27 (2009)
28. Trefke, J.: Grundlagen der referenzarchitekturentwicklung. In: Appelrath, H.-J., Beenken, P., Bischofs, L., Uslar, M. (eds.) IT-Architekturentwicklung im Smart Grid, pp. 9–30. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29208-8_2
29. Reidt, A.: Referenzarchitektur eines integrierten Informationssystems zur Unterstützung der Instandhaltung. Universitätsbibliothek der TU München, München (2019)
30. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. **28**(1), 75–105 (2004)
31. Vom Brocke, J., Simons, A., Niehaves, B., Riemer, K., Plattfaut, R., Cleven, A.: Reconstructing the giant: on the importance of rigour in documenting the literature search process. In: European Conference on Information Systems (ECIS) 2009 Proceedings, Verona, Italy (2009)

32. Webster, J., Watson, R.T.: Analyzing the past to prepare for the future: writing a literature review. MIS Q. **26**(2), xiii–xxiii (2002)
33. Mayring, P., Fenzl, T.: Qualitative inhaltsanalyse. In: Baur, N., Blasius, J. (eds.) Handbuch Methoden der empirischen Sozialforschung, pp. 543–556. Springer, Wiesbaden (2014). https://doi.org/10.1007/978-3-531-18939-0_38
34. Gläser, J., Laudel, G.: Experteninterviews und qualitative Inhaltsanalyse als Instrumente rekonstruierender Untersuchungen, 4th edn. VS Verlag für Sozialwissenschaften, Wiesbaden (2010)
35. Laplante, P.A.: What Every Engineer Should Know about Software Engineering, 1st edn. Taylor and Francis Group, Boca Raton (2007)
36. Peffers, K., Rothenberger, M., Tuunanen, T., Vaezi, R.: Design science research evaluation. In: Peffers, K., Rothenberger, M., Kuechler, B. (eds.) Design Science Research in Information Systems. Advances in Theory and Practice. DESRIST 2012. LNCS, vol. 7286, pp. 398–410. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29863-9_29
37. Rehse, J.-R., Hake, P., Fettke, P., Loos, P.: Inductive Reference Model Development: Recent Results and Current Challenges. In: Mayr, H.C., Pinzger, M. (eds.) Informatik 2016, pp. 739–752. Gesellschaft für Informatik e.V., Bonn (2016)
38. Baskerville, R., Baiyere, A., Gergor, S., Hevner, A., Rossi, M.: Design science research contributions: finding a balance between artifact and theory. J. Assoc. Inf. Syst. **19**(5), 358–376 (2018)
39. Brax, S.A., Bask, A., Hsuan, J., Voss, C.: Service modularity and architecture – an overview and research agenda. Int. J. Oper. Prod. Manag. **37**(6), 686–702 (2017)
40. Böhmann, T., Leimeister, J.M., Möslein, K.: Service systems engineering. Wirtschaftsinformatik **56**(2), 83–90 (2014). https://doi.org/10.1007/s11576-014-0406-6