Experience Report: First Steps towards a Microservice Architecture for Virtual Power Plants in the Energy Sector

Manuel Wickert
Fraunhofer IEE, Kassel, Germany
http://www.iee.fraunhofer.de
manuel.wickert@iee.fraunhofer.de

Sven Liebehentze
Fraunhofer IEE, Kassel, Germany
http://www.iee.fraunhofer.de
sven.liebehentze@iee.fraunhofer.de

Albert Zündorf
University of Kassel, Germany
https://seblog.cs.uni-kassel.de/
zuendorf@uni-kassel.de

Abstract

Virtual Power Plants provide energy sector stakeholders a useful abstraction for distributed energy resources by aggregating them. Software systems enabling this are critical infrastructure and must handle a fast-growing number of distributed energy resources. Modern architecture such as Microservice architecture can therefore be a good choice for dealing with such scalable systems where changing market and regulation requirements are part of every day business. In this report, we outline first experiences gained during the change from the existing Virtual Power Plant software monolith to Microservice architecture.

2012 ACM Subject Classification Software and its engineering → Software design engineering; Hardware → Smart grid

Keywords and phrases Microservices, VPP, Virtual Power Plants, Domain Driven Design

Digital Object Identifier 10.4230/OASIcs.Microservices.2017-2019.2

1 Introduction

In today's transition to green energy, the primary concept of a modern power plant is beginning to become that of a Virtual Power Plant (VPP), consisting of a huge number of small distributed energy resources (DERs) [14]. Usually these DERs are wind farms, photovoltaic parks, biogas plants, energy storages or flexible loads. VPPs have become the modern kind of a large power plant, replacing large conventional power plants over time. The coordination of a VPP is carried out by a software system, often referred to as an energy management system (EMS). Such systems are connected to the distributed energy resources to build the abstraction over a portfolio of DERs and provide monitoring and control capabilities.

During several research projects, Fraunhofer IEE has developed and evaluated such a software system, the VPP software solution IEE.vpp, which is in operation at some utilities and trading companies in the energy sector. The software has with time become a monolith with hundreds of KLOC (kilo lines of code). It consists of a canonical and generic data model [8] which has strengths and weaknesses. In situations where the data model did not fit into the corresponding business logic, the development time for the components increased significantly. In the energy domain, where the regulatory framework changes very often and new business models have to be implemented quickly, this will become a bigger issue over

Joint Post-proceedings of the First and Second International Conference on Microservices (Microservices 2017/2019).
Editors: Luís Cruz-Filipe, Saverio Giallorenzo, Fabrizio Montesi, Marco Peressotti, Florian Rademacher, and Sabine Sachweh; Article No. 2; pp. 2:1–2:10
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

time. Furthermore, it is hard for domain experts with algorithmic skills to contribute to the VPP solution because their main programming languages are Python and Matlab, but the VPP solution is written in Java. Therefore, we decided to migrate our macro architecture to a Microservice approach [5, 12], based on a Domain-Driven Design [4].

With the migration of our software systems we evaluate two different aspects. On the one hand, we outline how the modern architecture style of Microservices can be applied to the domain of VPPs. While there is currently some work on evaluating Microservices architectures in other areas of the smart grid such as metering [11] or IoT (Internet of Things) for smart buildings [2], most publications [16, 9, 10] for VPPs do not consider Microservices. Thus we present a first step for the application of Microservices as an architectural style for VPPs that should be transferable to other VPP software systems. On the other hand, using the example of a VPP, we show a novel migration approach to a different technology (e.g. programming language). For the reasons stated above, we planned to implement some Microservices in Python instead of using only Java.

This report presents the first steps of our migration to a Microservice architecture. The migration was done step-by-step during an agile development process to enable us to provide our customers with regular updates. The first step was a rough analysis of our domain to identify the main business functions to be supported by our VPP software system. Section 2 gives an overview of our Analysis. With this information, we are able to identify different bounded contexts for our software system in Section 3. For one of these bounded contexts, the migration to our first Microservice is described step by step in Section 4. In Section 5 we present the results of our first migration steps and discuss the advantages and disadvantages of our approach. Section 6 contains our conclusions, points out next steps as well as our most remarkable achievements.

2 Analyzing the Domain

From the perspective of Domain-Driven Design, VPPs typically use the two different domains of energy trading and grid operation in the energy sector. In [3, 14], for these domains, two different types of virtual power plants have been introduced - the technical virtual power plant (TVPP) and the commercial virtual power plant (CVPP). Although some software systems try to support use cases in both domains, most VPPs focus on just one. For the European market this will be due to the fact that the liberalization of the energy markets in Europe stipulates that TVPPs and CVPPs have to be operated by different companies and therefore have to support different business models. The VPP IEE.vpp software solution is a CVPP energy management system. Thus, the next sections focus on domain energy trading.

To understand the energy trading domain, we start with an example. The trading company “Green Trader” is highly skilled in renewable energy trading, for DERs dealing in windfarms, photovoltaics, biogas plants, batteries etc. For this reason it buys energy from different power plant owners, paying a price fixed by contract for the power fed into the grid. The fixed price is typically negotiated for one year. The “Green Trader” company bundles all bought energy in one portfolio and trades it on the EPEX SPOT energy market. Because the energy has to be traded before delivery, wind and photovoltaic production is forecasted and the current generation is monitored. Differences between traded and delivered energy is subject to a penalty payment for the trader. For optimizing the DER behaviour, according to market prices and to avoid penalty payments due to forecast errors, the “Green Trader” company is able to control the production and consumption (e.g. for batteries) of the DERs. To summarize, the “Green Trader” company needs to be able to monitor and control all DERs and optimize their schedules according to market prices. Therefore the company uses an energy management system for VPPs, such as the IEE.vpp system.

This example highlights two important requirements, “Monitoring and Control” and “Optimization” for an energy management system of a VPP from a trading perspective. However energy trading is much more complex and requires a number of different software systems. Therefore the energy management system should also support the provision of the collected information concerning the DERs, which is used by third party systems such as market clients, process and data management tools, etc. This information provision may be also needed for communication with other companies such as forecast providers. The provision of information may be a use case for many other software systems as well. However, since EMS hold critical business information concerning the DERs, information provision to another system is a crucial functionality. Therefore, we highlight this requirement as well.

The portfolio of energy traders changes over time, in Germany every year. Thus the integration of new energy units in a VPP and the termination of existing connections is a permanent topic. In addition to base data management, the integration of a DER consists primarily of the establishment of a communication link. The configuration of different communication protocols and manufacturer‘ specific data models is also one of the primary features of an EMS. Over the previous few years in many meetings with about 30 percent of all German energy traders, we became aware that the easy integration of power plants is one of the most desired requirements when building a VPP.

The analysis of the typical use cases for energy trading led to the following four main business functions that need to be supported by our EMS:

- Monitoring and Control
- Flexibility Optimization
- Information Provision
- Administration of DER

3 Identifying Bounded Contexts

Based on the described use case analysis and the subsequent designated business functions, we were able to define the domain precisely and derive the bounded contexts [12, 5, 15]. To outline the dependencies between these, we then built a context map [4]. Please note, that the business function “Information Provision” was skipped during this analysis because it can be regarded as a more abstract business function. The main task of this business function is the provision of information to third party systems. In order to define the associated domain model, the external interface specification must be known. These reasons led to three bounded contexts and a context map. In the context map we outline how we can integrate concrete bounded contexts for information provision.

The first bounded context “SCADA” (Supervisory Control And Data Acquisition) includes the supervision of control and data aquisition in the VPP software system, see Figure 1. A DER is able to read different sensor values. These might be measurements such as the current active power production, different temperatures or wind velocities. On the other hand a DER might be able to process control commands, for example to reduce the current power production. In order to exchange information and control commands between the VPP and the control system of a DER, a certain protocol is used as a rule, for example OPC-UA or IEC 60870-5-104. A so-called mapping scheme defines the translation between specialized protocol items or addresses and the VPP internal representation.

The next identified bounded context is the so-called “Aggregation”, see Figure 2. It includes aggregation and disaggregation data management strategies regarding the respective energy market commitment of the DERs. The Aggregation handles the management of portfolios, which are part of the VPP. Each portfolio is split according to various criteria, such as plant type or marketing strategies.

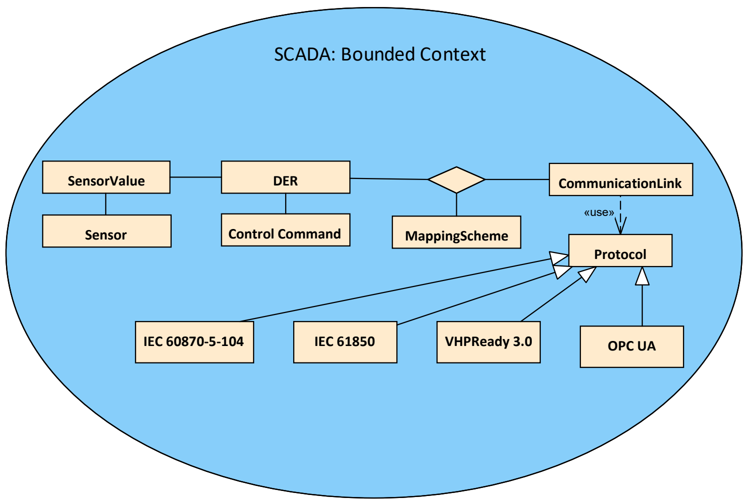

Figure 1 Bounded Context SCADA.

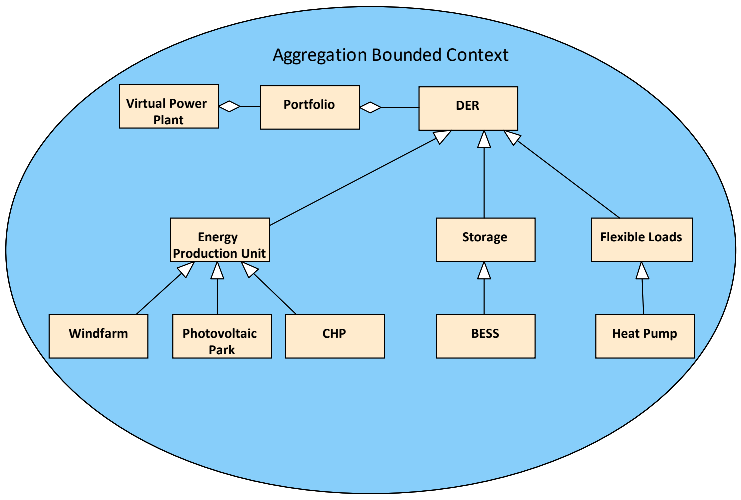

Figure 2 Bounded Context Aggregation.

The last identified bounded context is needed to model the unit commitment of the DERs. It is called “Optimization”, see Figure 3. A portfolio may be placed on one or more markets. Corresponding electricity price forecasts support the optimal electrical schedule calculation for the related plants. Figure 3 shows part of the domain model for the combined heat and power (CHP) optimization. For the best operation of a CHP, the energy production costs are mainly determined by the fuel costs, i.e. gas prices. Furthermore, the efficiency curve is taken into account in calculating how much electrical and thermal energy is produced depending on the amount of gas. Heat storage is another important influencing factor because it cannot be overfilled. The heat sink usually models the tight restrictions for the thermal energy consumption.

With the three defined bounded contexts we are able to realize the business functions from Section 2, except for the “Information Provision” as already noted. For the “Monitoring and Control” business function, the “SCADA” and “Aggregation” contexts are necessary. This is because the “SCADA” context contains the base model for control and monitoring. The

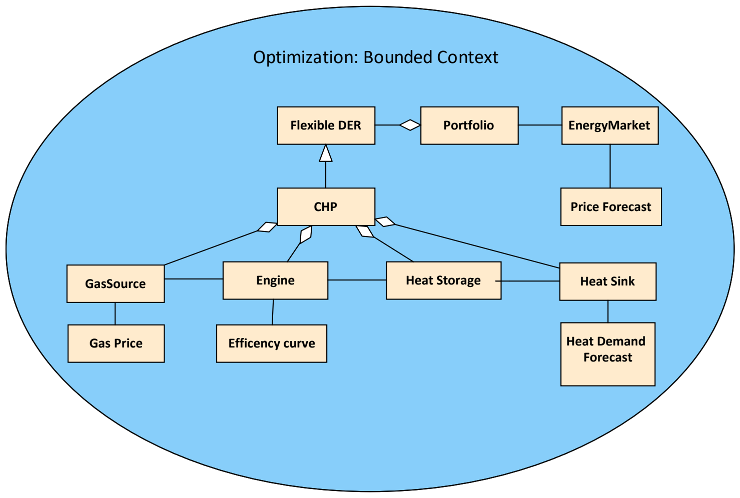

Figure 3 Bounded Context Optimization.

"Aggregation" context helps us to monitor different levels of aggregation and to disaggregate control signals. The "SCADA" context supports the "Administration of DER" business function as well. The domain model for the "Flexibility Optimization" is represented by the bounded context "Optimization" .

The three described bounded contexts and an abstract bounded context "Information Provision" are shown together in the context map in Figure 4. The Mappings between the bounded contexts are shown by the use of the patterns for strategic design, cf. [4]. We use the patterns Open Host Service (OHS) and Anticorruption Layer (ACL) for our purpose.

The "SCADA" context shares part of its model for providing sensor values and receiving control commands, the "Aggregation" context uses this OHS but translates it with the use of an ACL. The "Optimization" context uses processed information in a certain time resolution e.g. 15 minutes mean values. This information is provided by the bounded context "Aggregation" . The optimized schedules for the power plants may send directly to the "SCADA" context. However also schedules for the optimization have to be aggregated for the portfolios. In the current design, we plan to only have a link between "Aggregation" and "Optimization" . There is still the option to also establish a link between "Optimization" and "SCADA" . The context map also shows an abstract bounded context "Information Provision" . Typically this bounded context will receive information through a shared model from the "Aggregation" context and will use an ACL for the translation to the specific model.

4 Architectural Migration

We approached the migration case by case, starting with the bounded context Optimization. As already noted, in order to include the domain experts in the respective area of optimization, i.e. Mixed Integer Linear Programming (MILP) [13], we decided to implement the new Microservice in Python. Additionally, this enabled the use of popular MILP frameworks such as Pyomo [7, 6].

Before the migration, we identified parts of the old monolithic architecture, which are responsible for the Optimization (marked in yellow in Figure 5). Therefore a closer look at the monolithic architecture was needed. The software is implemented as a classic client server architecture, where the backend is built as a layered architecture (see 1. in Figure 5).

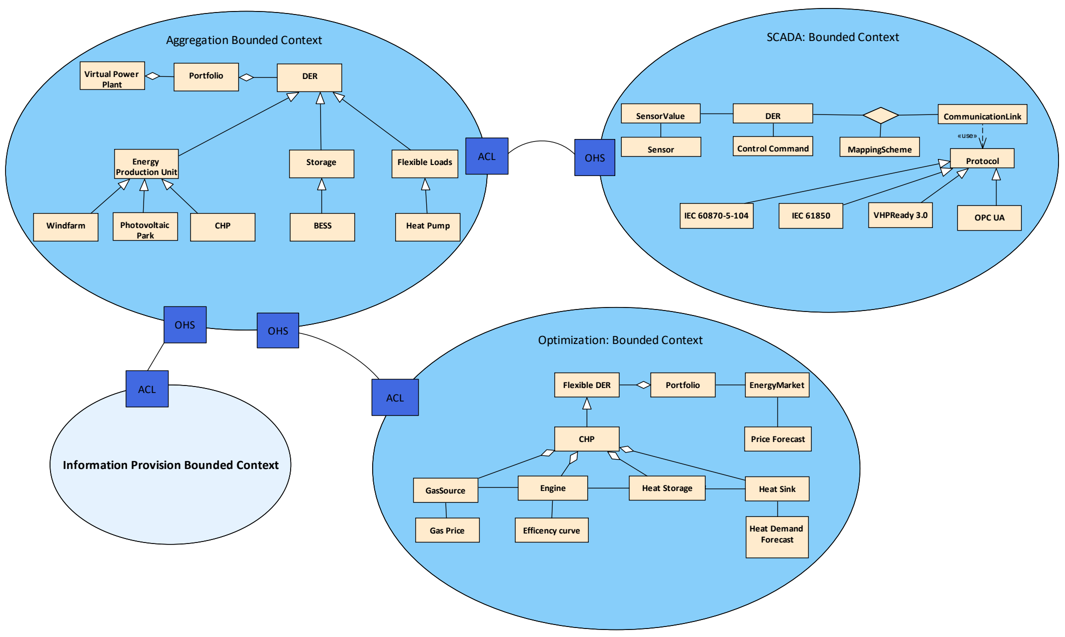

Figure 4 Context Map.

The user interface is written in JavaFX and implemented as a rich client. It connects via RESTful HTTP endpoints with the service layer of the backend. The service layer routes requests to the domain logic, e.g. the optimization kernel. The business logic uses raw data from DERs as well as aggregated and interpolated information. For this reason the business layer above an aggregation and disaggregation layer. The bottom layer consists of online data information retrieved directly from DERs as well as historical data retrieved from the database. In 1. of Figure 5 the parts that are responsible for the optimization are highlighted (marked in yellow). They are distributed over the different layers as well as in the UI and database.

In the next step (2. in Figure 5) the optimization kernel was implemented in Python. In contrast to a classical migration within the same programming language, a refactoring based migration, cf. [12], was not possible. To continue working within our agile development process and to keep the customer up to date, we decided not to extract all optimization related functionalities at this stage. Therefore we only extracted the core business logic into a SOA-like service and called these services directly from the layer where they were extracted from. In this step, we used the data model from the above defined bounded context and we used the OHS and ACL patterns, cf. [4], to transform the information between the monolith and the new service. The other layers, e.g. persistence and aggregation, as well as the UI, remained in the monolith. After this step, all existing integration and contract tests from the monolith could be used to verify that the VPP software worked as before.

In step three, we enhanced the SOA-Service to a Microservice and shifted the relevant parts of the aggregation and persistence layer from the monolith while adding a new database to the service as well (see 3. of Figure 5). Introducing a new user interface or a micro-frontend was skipped during this step. The main reason for this was the existing UI Technology. For JavaFX clients, there are no appropriate micro-frontend approaches which are independently deployable. Besides our Microservice migration we worked on the development of a web UI, therefore we shifted the UI separation to another step. To keep providing a common

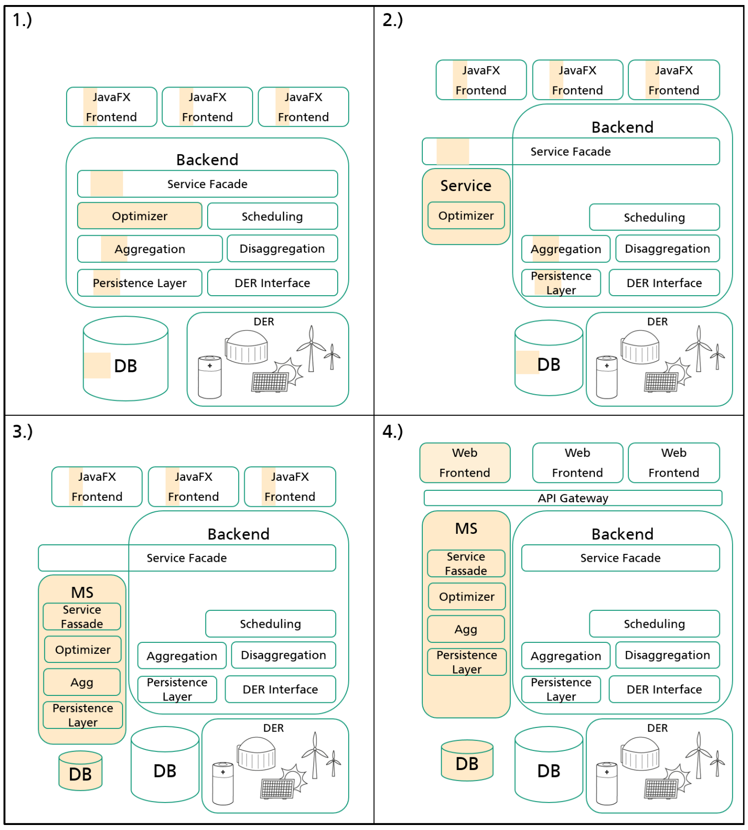

Figure 5 Four migration steps including the highlighted use of the bounded context Optimization.

interface to the UI and other applications that directly communicate with the REST service, we needed some kind of API Gateway. To minimise infrastructure changes for the first Microservice, we used the existing service facade as the API Gateway for the monolith and the Microservice as well.

The last step is to introduce a web UI and a separate API Gateway. However this step is only useful if the UI is completely based on a web UI. Currently the migration of the UI to Angular [1] is ongoing. After finishing this we will decide which existing API Gateway solution we are going to use. With the completion of this step, the full migration is finished and independent deployment will be possible.

The Architectural Migration was performed in four separate steps during our SCRUM-based development process with 2 teams (see Section 5). We had three-week iterations and each step took several iterations because the migration was done in parallel to the feature development. Overall it took about six months to deliver the first deployment to our customer. The reason for this being that we still needed some iterations in order to implement the existing functionality in Python. Also the integration of the domain experts into the new development Team took some time. For each of the migration steps we could perform existing contract tests, partial integration tests and deploy our software in production on schedule.

During our migration we identified two main challenges. The first was to bring the python modules into operation including logging and metrics, providing health status, evaluating web service frameworks and safety and security considerations, etc. The team's goal was to provide at least the same stable operation behavior as our Java Software already had. The second challenge was to implement component tests. Now, the mocking of the optimization changed from mocking Java Components to mocking a Microservice or an external Interface. Thus the tests scenarios became much more complex.

5 Results

As a result of the first migration we received a smaller monolith (about 270 KLOC) and a new Optimization Microservice (about 10 KLOC). Consequently we could significantly increase the number of developers for the product by the inclusion of our domain experts. We started with one development team for the monolith with 6 developers and have now two development teams with 5 (allmost all well experienced in software engineering) and 4 developers (1 software engineer, 3 economists/industrial engineers). However, the new development team has more skills in linear programming than in software engineering. Thus the implementation of the agile development process is not yet complete.

The deployment of the software has also changed. To frequently deliver updates we changed our production infrastructure to Docker together with the infrastructure team of our customers. This helped us to deploy Python software including all required modules in a stable environment. It also separated the infrastructure dependencies of the monolith and the new Microservices. A consequence of this is that the infrastructure is now more complex than before. To cope this, we are now working on the implementation of a distributed logging environment.

The migration itself gained from the fact that we implemented a Service oriented architecture (SOA) in the second step. This focused our implementation on the domain logic parts of the Microservice and without a database the infrastructure was less complex in the second step. In comparison with a direct Microservice implementation, our approach needs more development time because the integration consists of more steps. For working in agile development teams we still recommend our new approach for migrating to a different programming language, also for completely different domains.

6 Conclusion

In this experience report, we analyzed the domain of VPPs and pointed out relevant business functions for a CVPP. For each business function, we identified supporting models and built bounded contexts with a context map. With this domain-driven approach, we started the migration with the detachment of the bounded context Optimization. With this step a new Microservice evolved.

For the architectural migration, we examined a novel approach which separates the process into four steps. This approach is also transferable to other software systems that intend to use a new programming language within a new Microservice. Our most important achievement was the realization of the respective upcoming migration step while delivering new software features at regular intervals. The separation of the last step was important due to the existing UI technology. This may also be an issue for other existing systems.

Our greatest benefit was the first migration step, namely the inclusion of further expert knowledge in the development team by using Python as the programming language of the new Microservice. This also opened up the set of actively supporting development team members within our organization. Another benefit is the sharper domain within the new Microservice supporting a more rapid development cycle for optimization features.

The next step will be to also perform the migration of the user interface to a pure Web UI. With this technology it is possible to build so called Microfrontends and deploy them together with the Microservice. This also enables us to postpone UI framework or library decisions to development teams. Afterwards we will be able to start the decomposition of the remaining monolith to a SCADA and an administration service. A further decomposition may follow afterwards. One of the future challenges will be the measurement of the effect of the Microservice migration in terms of scaling behavior in operation, shorter development times, etc. This will give us indicators for further architectural decisions.

References

1 Angular Framework. URL: `https://angular.io/`.

2 Kaibin Bao, Ingo Mauser, Sebastian Kochanneck, Huiwen Xu, and Hartmut Schmeck. A Microservice Architecture for the Intranet of Things and Energy in Smart Buildings: Research Paper. In *Proceedings of the 1st International Workshop on Mashups of Things and APIs, MOTA@Middleware 2016, Trento, Italy, December 12-13, 2016*, pages 1–6, December 2016. `doi:10.1145/3007203.3007215`.

3 Martin Braun. Virtual power plants in real applications: Pilot demonstrations in Spain and England as part of the european project FENIX. *Fraunhofer IWES*, January 2009.

4 Evans. *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

5 Martin Fowler and James Lewis. Microservices, 2014. URL: `http://martinfowler.com/articles/microservices.html`.

6 William E. Hart, Carl D. Laird, Jean-Paul Watson, David L. Woodruff, Gabriel A. Hackebeil, Bethany L. Nicholson, and John D. Siirola. *Pyomo–optimization modeling in python*, volume 67. Springer Science & Business Media, second edition, 2017.

7 William E Hart, Jean-Paul Watson, and David L Woodruff. Pyomo: modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3(3):219–260, 2011.

8 G. Hohpe and B.A. WOOLF. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. The Addison-Wesley Signature Series. Prentice Hall, 2004. URL: `http://books.google.com.au/books?id=dH9zp14-1KYC`.

9 B. Jansen, C. Binding, O. Sundstrom, and D. Gantenbein. Architecture and Communication of an Electric Vehicle Virtual Power Plant. In *2010 First IEEE International Conference on Smart Grid Communications*, pages 149–154, October 2010. `doi:10.1109/SMARTGRID.2010.5622033`.

10 Ingo Mauser, Christian Hirsch, Sebastian Kochanneck, and Hartmut Schmeck. Organic Architecture for Energy Management and Smart Grids. In *2015 IEEE International Conference on Autonomic Computing, Grenoble, France, July 7-10, 2015*, pages 101–108, 2015. `doi:10.1109/ICAC.2015.10`.

11 Antonello Monti, editor. *Technologies and Methodologies in Modern Distribution Grid Automation*, volume 15. Elsevier, Amsterdam [u.a.], 2018. URL: `http://publications.rwth-aachen.de/record/745831`.

12 Sam Newman. *Building Microservices*. O'Reilly Media, Inc., 1st edition, 2015.

13 Seyyed Mostafa Nosratabadi, Rahmat-Allah Hooshmand, and Eskandar Gholipour. A comprehensive review on microgrid and virtual power plant concepts employed for distributed energy resources scheduling in power systems. *Renewable and Sustainable Energy Reviews*, 67(C):341–363, 2017. `doi:10.1016/j.rser.2016.09.02`.

14 H. Saboori, M. Mohammadi, and R. Taghe. Virtual Power Plant (VPP), Definition, Concept, Components and Types. In *Proceedings of the 2011 Asia-Pacific Power and Energy Engineering Conference*, APPEEC '11, pages 1–4, Washington, DC, USA, 2011. IEEE Computer Society. `doi:10.1109/APPEEC.2011.5749026`.

15 Vaughn Vernon. *Implementing Domain-Driven Design*. Addison-Wesley Professional, 1st edition, 2013.

16 Matej Zajc, Mitja Kolenc, and Nermin Suljanovic. *Virtual Power Plant Communication System Architecture*, chapter 11, pages 231–250. Elsevier, 2018. doi:10.1016/B978-0-12-812154-2.00011-0.