On Microservices in Smart Grid Capable pmCHP

Richard Pump　　Arne Koschel　　Volker Ahlers
Department of Computer Science
University of Applied Sciences and Arts
Hannover, Germany
Email: {richard.pump, arne.koschel, volker.ahlers}@hs-hannover.de

***Abstract*—Portable-micro-Combined-Heat-and-Power-units are a gateway technology bridging conventional vehicles and Battery Electric Vehicles (BEV). Being a new technology, new software has to be created that can be easily adapted to changing requirements. We propose and evaluate three different architectures based on three architectural paradigms. Using a scenario-based evaluation, we conclude that a Service-Oriented Architecture (SOA) using microservices provides a higher quality solution than a layered or Event-Driven Complex-Event-Processing (ED-CEP) approach. Future work will include implementation and simulation-driven evaluation.**

***Keywords*–Smart Grid; pmCHP; microservices; service-orientation.**

I. INTRODUCTION

The energy grid of the future requires extremely interconnected devices to regulate the amount of energy produced precisely to the needed amounts. Former tree-like distribution networks are replaced with small autonomous microgrids which imitate a peer-to-peer network [1]. To coordinate a higher amount of generators, each device has to be intelligent.

Not only the energy grid is changing, also the automotive industry is in turmoil. More and more countries work on reducing the carbon footprint and greenhouse gases to fight climate change. Driven by legislation and public conscience the amount of BEVs is rising. However, low range and comfort (in comparison to conventional vehicles) keep consumers away.

Combining stationary small scale generation and mobile usage, the *University of Applied Sciences and Arts Hannover* is working on pmCHP. The pmCHP generates heat and electricity at much higher efficiency than comparable conventional devices. A novel feature of the pmCHP is the dual usage in buildings as well as BEV. In buildings, the pmCHP is attached to a smart grid and helps to cover peak loads, in a BEV the pmCHP enhances the passengers comfort and extends the range of the vehicle [2].

This contribution evaluates different software architectures for the control of the pmCHP, since the use in a constantly evolving Smart Grid requires a well-crafted adaptive Architecture. We will present three designs using different architectural paradigms and evaluate them using a scenario-based procedure. Section II will present related work, showing there is not much research concentrating on the architecture of pmCHP-Software. Section III presents the three developed architectures: SOA, ED-CEP, and layered. In Section IV and V, we evaluate the architectures using likely scenarios that the software will encounter over its lifespan. The last section concludes this work and gives some outlook to future work.

II. RELATED WORK

Regarding software-architectures for the smart grid, mostly interactions are standardized. For example, the Standards 61968/61970, designed by the International Electrotechnical Commission (IEC) describe a global domain model of the smart grid with pre-defined interfaces and messages. The Standards however do not describe a pre-defined internal software architecture.

In [3], Reinprecht et. al. describe the IEC Common Information Model (CIM) architecture, which is a layered architecture that ensures Standard compliant implementation over the different levels of the architecture. The authors describe multiple SOA-based designs which were created for the Smart Grid Interoperability test. A comparison or evaluation of the architectures is not mentioned.

Appelrath et. al. [4] show a reference architecture for smart grid software. It describes general interfaces for abstract devices, a real device might be composed of multiple abstract ones. However, neither a concrete implementation nor an evaluation of alternatives is presented.

An architecture to operate a pmCHP testbed is presented in [5]. There is no connection to the smart grid, although microservices are used to provide high architectural flexibility.

To compare different architectural designs, Kazman et. al. [6] present a scenario-driven comparison method that provides the general process used in this work.

The most important quality-aspects of smart grid software are proposed by the NIST in [7]. Since the Smart Grid is critical infrastructure one of the most desired qualities is the availability of the devices. These qualities are considered when comparing the different designs in section IV.

All considered there is no concrete work on how to integrate a pmCHP into a smart grid, let alone an evaluation of suitable software architectures for this purpose, known to the authors of this paper.

III. ARCHITECTURES

To compare different architectures, a rough sketch of the desired components is needed, ensuring functional and conceptual similarity. The goal of the software is to drive the pmCHP according to different energy requests with regards to the operational strategy. Energy requests can be accepted from an external source, like the smart grid or originate from an internal source. The requests arriving at the software have to be incorporated into the current operational plan, which defines the pmCHPs operation.

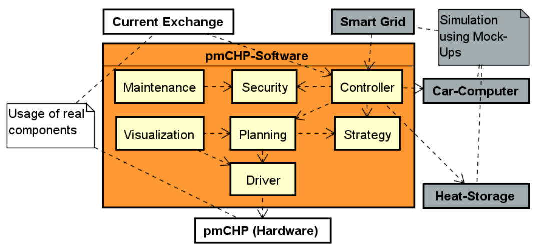

Figure 1. Rough sketch of the pmCHP-Software

Figure 1 shows the principal components of the pmCHP software.

The CONTROLLER coordinates the production of the pm-CHP with its environment, be it the Smart Grid or the computer of the BEV. It receives energy requests, validates their security, and checks if they can be fulfilled. Valid requests are handed the PLANNING for further processing.

The component STRATEGY provides the framework for the day-planning, as it decides which operational strategy is used. The pmCHP can be used in three different modes; electricity-driven, heat-driven, and combined heat- and electricity-driven. The electrically-driven mode controls the production of the pmCHP depending on the needed electricity, heat is seen as byproduct and will not be produced if no electricity is needed. In heat-driven mode, the pmCHP uses the heating requirements as the control value, the combined operation mode just produces depending on whichever energy is needed at the time. STRATEGY has to decide which of the three modes is the most sensible, depending on the operating environment of the pmCHP.

The component PLANNING is responsible for planning the day-to-day-operation of the pmCHP, according to the operational strategy. It uses data about previous operation and forecasts to create an operational plan which is used by the DRIVER to control the pmCHP. The Component DRIVER transforms the operational plan into control-commands, monitors the pmCHP state, and provides this information to other parts of the software.

The VISUALIZATION component presents the current state and planned operation to the user of the device. Also errors and warnings can be shown to the user, so corrective action can be taken if needed.

To facilitate remote access to the pmCHP in case the software needs to be updated or other remote action needs to be taken, the component MAINTENANCE exists. It allows remote software updates, access to log files and current operational status of the pmCHP as well as remote control in case the grid operating company needs to control the pmCHP manually.

Allowing remote access to the pmCHP Software without any security measures would be grossly negligent, therefore a component SECURITY needs to take care of authentication and authorization of all incoming requests.

Other Components like SMART GRID, CURRENT EXCHANGE, etc. are beyond the scope of the software and represent neighboring systems to interact with.

Based on the previously shown rough design, three different architectural styles were used to create three architectures: SOA, ED-CEP and layered.

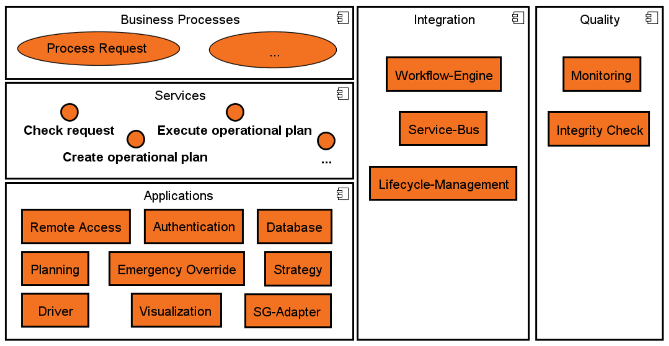

Figure 2. Overview of the SOA. (Not all services and processes are being shown to simplify presentation.)

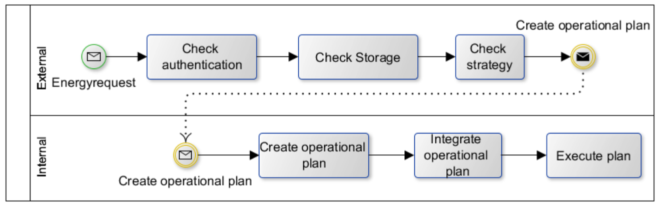

Figure 3. Business process: ’Process energy-request’.

A. Service-oriented Architecture

Service-oriented architectures use loose coupling and high cohesion in all parts of the software to achieve high flexibility under changing requirements [8]. They can be used in combination with microservices to allow for rapid redevelopment of all parts of the software.

In a SOA, the high level functions of the software are mapped to business processes which compose the services into useful processes. The services are responsible for small parts of functionality which can be reused in different contexts. Therefore, to create a service-oriented architecture for the pmCHP the usage scenarios need to be translated into business processes, which in turn need to be decomposed into small services, fit for a microservice approach.

Converting the rough sketch of Figure 1 into a SOA can result in the design shown in Figure 2. There are 13 business processes using over 20 different services, using the different Applications.

The most important business process is the processing of energy requests, arriving from an external source or created by the software itself, as shown in Figure 3. If a request arrives at the software at first the authenticity of the requests needs to be checked. If the request is valid, storage and strategy have to be checked; maybe the request can be fulfilled just by using the attached storage or cannot be fulfilled because of the currently selected operational strategy.

Assuming that a request is correct, another process is started, which creates an operational plan to fulfill the request, integrates that plan into the currently executed plan, and then executes the result using the driver. Requests not always originate from an external source, sometimes the software itself creates energy requests to achieve own interests. Internal requests do not need validation or crosschecking with storage

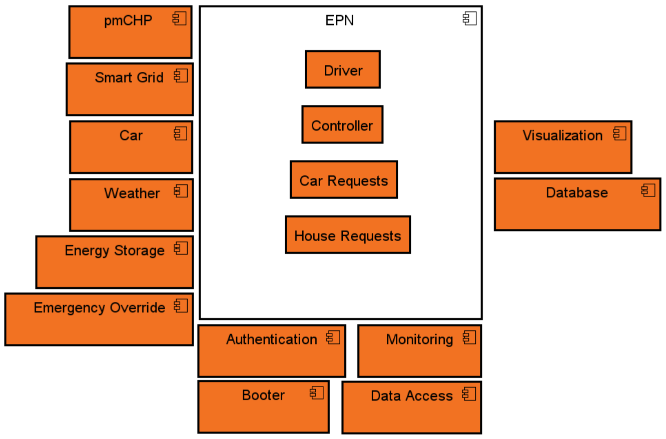

Figure 4. Components of the event-driven architecture.

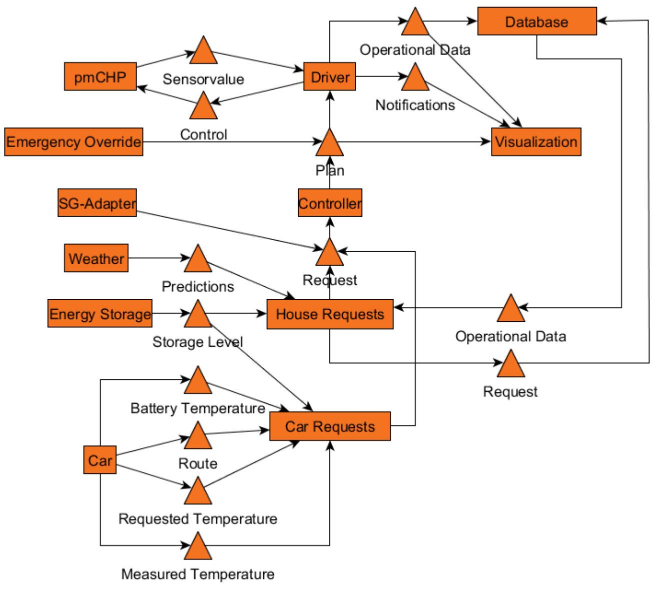

Figure 5. The Event-Processing-Network of the ED-CEP, showing Events and their flow from sources to sinks.

and strategy.

The Services used in the process are rather simple and only take care of a single functionality. For example, the service 'Check request' simply checks the requests validity; i.e. through using a signature. The services are implemented in so called applications to keep similar functionalities under the same name; an application can contain one or multiple microservices. In general, there is no direct interaction between the services, for orchestration a workflow-engine is used, which translates the business processes into service-calls.

B. Event-driven complex-event-processing

An ED-CEP-architecture divides the software into two major parts. Main controlling logic and business processes central to the software are mapped to event-processor-chains in the Event-Processing-Network (EPN), while the surrounding applications provide services for tasks to be executed [9].

Figures 4 and 5 show the components of the architecture and the EPN, respectively. There is a notably similarity between the applications of the ED-CEP and the applications in the SOA, since the components fulfill similar roles in both architectural styles.

To ease comparison to the SOA, we will trace the events responsible to process an energy request from the smart grid in the EPN. With Figure 5 in mind, any smart grid requests enter the EPN through the SG-ADAPTER, which creates a *request*-event. *Request*-events in general are processed by the CONTROLLER, which after some processing creates an operational plan and wraps it in a *plan*-event. Using information from the *plan*-event the DRIVER creates *control*-events which are processed by the PMCHP to control the physical device. Further *notification*- and *operational data*-events are created, which are used to by different sinks.

C. Layered approach

The layered approach follows the simple principle, that components of a higher layer might use components of lower layers but not vice versa [10]. Generally, components will be divided into five layers, presentation/interface, requirements, business logic, technical logic, and data/hardware. Figure 6 gives an overview about the components of our layered architecture.

The *presentation/interface*-layer allows access to the functionalities of the software for external actors like other software and the user. Of note are the components EMERGENCY and MAINTENANCE, allowing other systems direct access to functions of the software.

Below the *Access*-layer are the three core logic layers of the software, decreasing in abstractness the further 'down' the architecture we traverse.

First is the *Request*-layer. Three different components make up the *Request*-layer, SMART GRID-REQUESTS, REQUESTCREATION CAR, and REQUESTCREATION BUILDING. While the first handles requests from the smart grid, the latter two create energy requests according to the operational strategy.

The second core layer is called the *Business Logic*, containing the components PLANNING and STRATEGY responsible for transforming the different requests into a single operational plan which can be executed by the lower layers.

Beneath, the third logic layer contains components which provide *technical* functionalities not dependent on external systems while being responsible for essential operations in the system. For example, the component BOOTER starts the whole system, initializing the other components.

The lowest layers *Data* and *Adapter* connect the software to other devices and systems, allowing operational data to be stored and queried, information about surrounding systems to be collected, and interaction with the pmCHP.

Figure 7 shows how the different components interact when a energy request arrives at the system. After being passed through to the AUTHENTICATION, the request is handled by the component SMART-GRID-REQUESTS where it is converted into an operational plan that can be executed by the DRIVER.

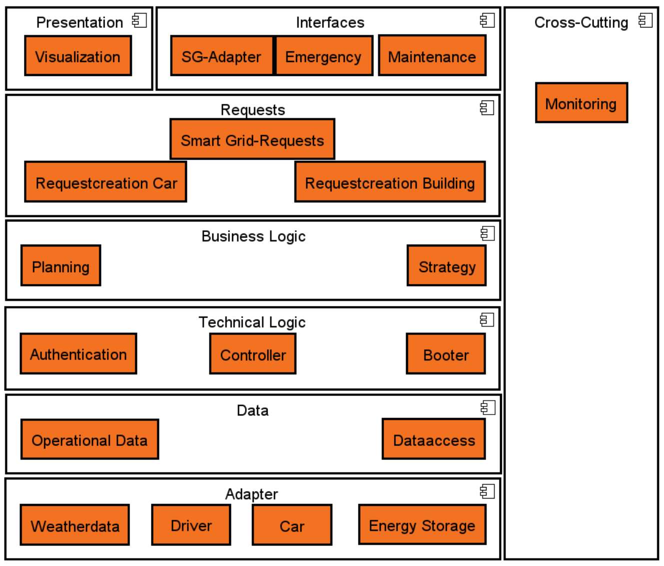

Figure 6. Decomposition of the components into layers following a canonical layered approach.

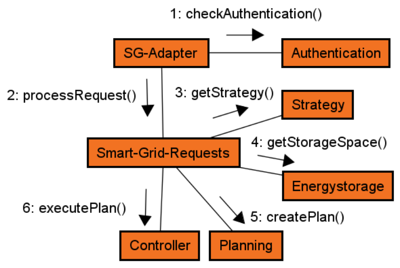

Figure 7. Interactions of the components required to process an energy request.

IV. Comparison of Architectures

Comparisons of architectures are a useful tool to increase software quality at an early stage of the development cycle [11]. Choosing the wrong architecture can decrease the maximum achievable quality by forcing bad design. For example, forcing an EPN into a micro-controller controlling a toaster will have less performance and increased development cost over a monolithic software. (Assuming the software is tasked with just turning the toaster off as soon as a signal is received. EPNs most likely handle complex scenarios better than monolithic approaches.)

Multiple ways exist to compare different architectures, but most commonly scenario-based methods are used. Scenario based methods use scenarios to estimate necessary changes to the architecture, which in turn can be used as an indicator for the quality of the architecture. The first step of a comparison is the definition of the architectures in some form, as already described in the previous section. In a second step, scenarios describing possible usages or changes of the architecture are defined, each providing a measurable way to describe quality. The scenarios are grouped after the five general aspects of software quality and use the rough system sketch as a common baseline for all architectures.

a) Availability: Availability scenarios describe situations in which the system has to take certain countermeasures to provide uninterrupted operation. Availability is the most important quality in an energy providing system [7].

Ava01 The Driver crashes due to an error, the system realizes the failure and immediately switches to a backup.

Ava02 The Driver receives a single incorrect measured value outside of the defined thresholds for this sensor. Instead of immediately shutting down the pmCHP the Driver averages values and prevents shutdown due to measurement errors.

Ava03 The connection between the Driver and the pmCHP is severed and cannot be reestablished. An error is presented to the user and the pmCHP switches to a safe operating mode instead of shutting down immediately.

Ava04 A usual high amount of energy requests is received from the smart grid. After a certain threshold is reached, the Controller rejects all further requests to provide protection against overload-attacks.

b) Security: Security-scenarios describe situations in which the software is possibly used in a way that it is not intended and unwanted. In an interconnected network with access to physical systems, security is one of the most important qualities the software has to achieve.

Sec01 A different system tries to access a pmCHP-software functionality, the authenticity of the accessing system is checked, before access is granted.

Sec02 When the pmCHP is activated, a minimal software checks the integrity of the pmCHP-software using a digital signature. If the signature is not correct, an error is presented to the user, and the software does not start.

Sec03 A manipulated component tries to access a function of the Driver which it normally would not access and is not authorized to do so. The component Security recognizes the unauthorized attempt, prevents it and produces an error message shown to the user.

c) Safety: In contrast to security, safety-scenarios are describing potentially dangerous situations in the normal operation of the pmCHP-software. Again safety is rather important in operating an energy generating device, as failures can harm humans and the operating environment.

Saf01 The Driver continuously monitors all of the pmCHPs sensors and detects dangerous operation. If a dangerous operation is recognized, the Driver transfers the pmCHP into a safe mode of operations, possibly even shutting it down.

Saf02 All control-signals are checked by the Driver, ignoring signals that might damage the pmCHP.

d) Maintainability: Since the smart grid is not completely clear at the moment, adaptability and maintainability is somewhat needed. The following scenarios include likely changes and developmental processes of the software's lifetime.

Mai01 After the end of the pmCHP-development a different developer is tasked to add smart market integration to the pmCHP-software. The smart market component needs to accept requests from the smart grid, overview their execution, and take care of the billing aspects according to the energy contract.

Mai02 The emergency shutdown shall be tested intensively; the required components can be interchanged with mock-ups without changing the DRIVER.

Mai03 The systems architecture is checked by a software architect. Similar problems are solved in similar ways using similar architectural or design patterns.

e) Performance: Performance is not overall important to the pmCHP-software, only a single scenario is presented.

Per01 A malfunction of the pmCHP requires an emergency shutdown, the shutdown happens fast enough to prevent damage.

f) Usability: Usability describes the grade at which the user's interaction is eased by good interface and software design. Since there is almost no interaction of the user with the pmCHP-software, usability is an afterthought.

Usa01 To start or stop the pmCHP, only a single button has to be pressed by the user.

Usa02 After being started the software presents the momentary state of the pmCHP and can display the current operational planning.

A. Evaluation

To evaluate the architectures, we trace the necessary changes to the different architectures when applying the scenarios. We differentiate between easy changes, which will take some hours and difficult changes which will probably take significantly longer to execute. This provides an estimate for the loose coupling and functional adequateness of the architecture; if a functionality change incorporates changing a lot of components, loose coupling might not be present or the architecture might not have been designed with the scenario in mind and does not cover the requirements adequately.

For example, the scenario [Mai01] requires a new business process and a new service in the SOA, which equates in three changes to the architecture. To facilitate billing, the process to *process requests* has to be extended with service calls to a new service BILLING. Also the service bus has to be modified to connect the service to the application containing the logic.

The ED-CEP-architecture however requires a lot of changes, changing the REQUEST-event to accommodate the information necessary for the billing requires changes in all REQUEST-event-processing components. Also new events and processors for the billing itself have to be created.

Including the smart market into the layered architecture requires changes to two existing components and a new component; changing the SG-ADAPTER and the SMART GRID REQUEST to pass the needed data and oversee the completion of an order, as well as a new BILLING-adapter connecting to the appropriate smart market.

For brevity, we skip the evaluation of each architecture using every mentioned scenario and directly present the results.

TABLE I. COUNT OF CHANGES TO COMPONENTS NECESSARY TO FULFILL ALL SCENARIOS.

SOA Component	Count	ED-CEP Component	Count	Layers Component	Count
Driver (difficult)	2	Driver	3	Driver (difficult)	3
ServiceBus	2	SG-Adapter	2	SG-Adapter	2
WfE	1	Controller	2	Controller	1
process request	1	Driver (difficult)	1	Maintenance	1
Billing*	1	Planning	1	Visualization	1
		Emergency override	1	Smart Grid-request	1
		Monitoring	1	Billing	1
		Booter	1	Booter	1
		Request	1		
		Billing-event	1		
		Billing	1		
Total:	7		16		13
of total					
- difficult:	2		1		3
- easy:	5		15		10

B. Evaluation results

Table I shows the amount of changes necessary to the architectures. Overall, the event-driven complex-event-processing-architecture needs the most number of changes (although most of them easy), while the layered-architecture needs the most difficult changes.

The high number of changes to the ED-CEP-architecture originate from the scenario to integrate the software into the smart market, since a long chain of interactions had to be changed to facilitate billing. Also to isolate the DRIVERs functions from the rest of the architecture, checking access at every possible place, a lot of changes had to be made. Of note however is the ease with which changes (e.g. detection of dangerous operational states) can be done to the DRIVER of the CEP architecture. The rule-based implementation of logic provides an easy, powerful way to define actions on detection of certain states. In contrast, applying the same change to the service-oriented or the layered architecture requires more work to incorporate complex condition-action-mechanisms.

There are a lot of difficult changes which need to be applied to the layered architecture, mostly concerning the DRIVER. These primarily originate in the scenarios [Sec03], [Saf01], and [Saf02] each requiring a lot of functionality to be added to the component, since no other component is be responsible for the needed functions.

The SOA however requires the least amount of changes. Adding a smart market integration for example requires the least amount of changes in the SOA, while more extensive changes have to be made to the CEP or the layered architecture. The fine grained nature of the underlying microservices also helps the maintainability and flexibility of the software.

Assuming difficult changes need a lot more time than easy ones and a high amount of time needed is a sign of a bad architecture the following ranking can be created:

1) SOA using microservices
2) ED-CEP
3) Layered architecture

Considering this, we can infer that the SOA provides the loosest coupling and the highest functional adequateness for our scenario.

C. Scenario interactions

After checking loose coupling and functional adequateness, we need to validate the cohesion of the components, i.e. the proper separation of functionalities into components. Since every scenario should concern a single functionality, we can check the proper separation by looking for components which interact with more than one scenario.

Table II shows the components which are affected by multiple scenarios.

TABLE II. COUNT OF SCENARIO INTERACTIONS WHEN APPLYING THE SCENARIOS TO THE COMPONENTS OF THE DIFFERENT ARCHITECTURES.

SOA		CEP		Layers	
Component	Count	Component	Count	Component	Count
Driver	2	Driver	4	Driver	3
ServiceBus	2	SG-Adapter	2	SG-Adapter	2
		Planning	2		
		Controller	2		

Under this metric, the CEP-architecture is the worst of the three. The division of a functionality over a lot of small components (i.e. events) proves to be a negative factor under the scenario interaction. This infers an unclear separation of functionality over the components.

Layers and the SOA seem to be somewhat equal concerning scenario interaction, especially the driver seems to be a component which needs further refinement.

Under the scenario interaction metric, no clear ranking can be established, the layered approach and the SOA seem to be equal, the CEP seems to be the worst.

V. Evaluation

Considering the previous sections, the SOA can be inferred to be the best choice for the integration of a pmCHP into the smart grid.

Especially the loose coupling and ease of change of service orientation and microservices help to create an architecture with a high grade of flexibility. In scenarios where a lot of changes to the other two architectures were necessary, SOA needed only a small amount of little changes. Considering the unclear future of the pmCHP-software this high flexibility is a very desirable property.

The layered architecture however is not a good choice, since it requires a relatively high amount of difficult changes to adapt to the scenarios, meaning more work in the long term. Similarly, the ED-CEP-architecture seems a bad fit for the problem. This however might be explained by insufficient design or bad evaluation-metrics. Counting the number of changes to introduce a new process into an EPN has to result in a high amount of changes, since a lot of small components need to be added, resulting in a bad score. If we look at the EPN as a whole however, only a single component changes in a lot of scenarios, interaction however grows also. A good way to handle this might be to split the EPN in smaller function-specific EPNs.

VI. Conclusion and Future Work

We designed and evaluated three different designs to integrate pmCHP into a future smart grid. Using a scenario-based comparison we conclude that the usage of microservices can result in testable better architectures when a high degree of flexibility is needed. This is mostly due to the few interactions between Microservices in a SOA; a business process can just chain service-calls to achieve its desired result. To make changes to a certain functionality, services can be changed independently, new services can be introduced without changing others.

The results of this paper however only hold for the special case of integrating a pmCHP into a smart grid, other goals will likely result in different results.

Also, since all architectures have been developed by the same person over a short span of time, they likely influence each other. Especially the CEP and the SOA share some applications, which can also be explained by similar design philosophies. Further work combining the two might prove an even better solution for the smart grid integration of pmCHP.

In future steps, we will implement the SOA and evaluate the impact of pmCHPs in a smart grid.

Acknowledgment

This work was supported by the VolkswagenStiftung and the Ministry for Science and Culture of Lower Saxony (project funding number VWZN2891). We would like to thank all our colleagues from the research focus Scalability of mobile Micro-CHP units and the Institute for Engineering Design, Mechatronics and Electro Mobility (IKME) for their support and the productive cooperation.

References

[1] H. Farhangi, “The path of the smart grid,” Power and energy magazine, IEEE, vol. 8, no. 1, 2010, pp. 18–28.

[2] C. Schmicke, J. Minnrich, H. Rüscher, and L.-O. Gusig, “Development of range extenders to mobile micro combined heat and power units in vehicles and buildings; *Weiterentwicklung von Range Extendern zu mobilem mikro-Blockheizkraftwerken in Fahrzeugen und Gebäuden*,” Techniktagung Kraft-Wärme-Kopplungssysteme, April 2014.

[3] N. Reinprecht, J. Torres, and M. Maia, “IEC CIM architecture for Smart Grid to achieve interoperability,” International CIM Interop in March 2011, 2011.

[4] H.-J. Appelrath, L. Bischofs, P. Beenken, and M. Uslar, IT-architecture-development in the Smart Grid; *IT-Architekturentwicklung im Smart Grid*. Springer, 2012.

[5] C. Schmicke, J. Minnrich, H. Rüscher, and L.-O. Gusig, “Examination of mobile micro-chp on testbeds of the University of applied Sciences and Arts Hanover; *Untersuchung von mobilen mikro-BHKW an Prüfständen der Hochschule Hannover*,” Ingenieurspiegel, vol. 4, 2015, pp. 60–61.

[6] R. Kazman, G. Abowd, L. Bass, and P. Clements, “Scenario-based analysis of software architecture,” IEEE Software, vol. 13, no. 6, Nov 1996, pp. 47–55.

[7] A. Lee and T. Brewer, “Smart grid cyber security strategy and requirements,” Draft Interagency Report NISTIR, vol. 7628, 2009.

[8] A. Arsanjani, “Service-oriented modeling and architecture,” https://www.ibm.com/developerworks/library/ws-soa-design1/ 2017.11.01, November 2004.

[9] R. Bruns and J. Dunkel, Event-driven architectures: software architecture for event-driven business-processes; *Event-driven architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse*. Springer-Verlag, 2010.

[10] E. W. Dijkstra, “Structure of an extendable operating system,” http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD275.PDF 2017.11.01, November 1969, circulated privately.

[11] P. Clements, R. Kazman, and M. Klein, Evaluating software architectures. Addison-Wesley Professional, 2003.