Integrating Portable Micro-CHP into a Smart Grid

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*—We present an architecture to integrate a portable micro combined heat-and-power (pmCHP) unit into a smart energy grid. The pmCHP is a gateway technology to bridge conventional vehicles battery electric vehicles, increasing range and comfort. Furthermore, pmCHP are to be used in the house within a connected smart energy grid. A software system is required to drive the pmCHP operation within building and vehicle. The System needs to be highly adaptable to accommodate the high amount of changes a novel device will undergo as it is introduced into a real world scenario. To find the best architecture, we design three different architectures using different architectural styles and evaluate them based on five categories of software quality. 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; layered architecture; CEP; scenario-based evaluation

I. Introduction

This article is based on our previous paper [1] and expands upon it. A section showing the exact requirements for the development of the architectures was added as well as further information about the developed architectures giving more insight into the designs. Also, the evaluation was expanded, adding information about the application of the scenarios to the architectures, documenting the changes.

The current energy distribution system of Europe is in change, former tree-like distribution networks are replaced with small autonomous microgrids, which imitate a peer-to-peer network [2]. With ever growing demands for electrical energy old, tree-like structures do not scale well enough to further justify their construction. Peak loads demand over-building for capacities that are only used during a very small amount of time. With a peer-to-peer network, distributed energy generation can be introduced to the grid much easier, which allows every participant of the grid to generate and consume energy opening a new market for small scale energy trading. The change to distributed generation also alleviates the issue of overbuilding to compensate peak loads and allows more efficient expansion of the electrical grid to cover more extensive loads. This however requires a high amount of coordination between the different participants of the grids, resulting in ubiquitous automation of the grid.

In the same vein the automotive industry is currently changing. To combat global warming, cars have to meet strict emission goals to be allowed to operate in some countries. With current cars being optimized to the boundaries of engineering, alternative sources of propulsion are under review, resulting in an emergence of electrically driven cars, called battery-electric vehicles (BEV). However, the development of affordable BEVs with comparable range to conventional combustion-driven vehicles is slow, current BEVs often reach only half the maximum range of conventional cars. Additionally, BEVs only reach their maximum range without passenger climatization, since the energy needed for air conditioning and heating is drawn from the same battery as propulsion. Using a BEV therefore requires careful attention to the available driving range, since recharging station are comparatively scarce and a recharge often takes a lot of time. The fear of being stranded somewhere along the way when using a BEV is a major impediment to customer acceptance.

To alleviate issues in BEV and to bridge the gap between BEV and ICE-cars, the *University of Applied Sciences and Arts Hannover* is currently developing the portable micro-combined-heat-and-power unit (pmCHP). The pmCHP is a small scale version of conventional CHP-units, weighing less than 40 kilograms, and can be carried by a single person. Utilizing the co-generation of heat and electricity to achieve a very high efficiency, the pmCHP is designed to be used in a BEV or to be connected to a smart energy grid in a house. In the BEV the pmCHP generates heat (which can be converted to coldness) and electrical power, increasing the range of the car directly and allowing the conditioning of the passenger compartment [3]. It is comparable to a conventional range-extender, a small combustion engine, which generates electrical power to increase the range of a BEV, but is more efficient and provides conditioning. When the pmCHP is not needed in the BEV, it can easily be unplugged and carried to another point of use, for example in a house. In the house it supports the local heating and energy production, covering peak loads and recharging the local storages. This however requires an integration of the local smart energy grid to identify moments of peak load or other scenarios, in which to use the pmCHP efficiently. For example, the generation of electricity has to be coordinated with other, less flexible producers, like solar cells or small scale wind energy.

In this article we will present a smart grid integration of the pmCHP, starting with defining the requirements of the software, selecting the best architecture and showing the interoperability with the established smart grid standards. To start off, Section II will give an overview of the relatively small amount of related works. Requirements will be explained and listed in Section III and Section IV, before resulting architectures are presented in Section V. After the presentations of the architectures, we will show our process of architecture

comparison in Section VI, which is based on a scenario driven approach. The article ends with an evaluation in Section VII and our conclusion in Section VIII.

II. Related Works

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 predefined interfaces and messages. The standards however do not describe a predefined internal software architecture.

In [4], 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. [5] 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 [6]. 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. [7] 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 [8]. 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 VI.

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 article.

III. Requirements

Building an architecture requires a clear set of goals, often referred to as requirements. The requirements define a clear goal of what the software has to achieve to be successfully developed and provide a baseline, against which a software can be evaluated. For the pmCHP the requirements are split into three categories, based on the planned operations for the device. There are general requirements describing its use in all scenarios, requirements for the usage in the car and requirements for the usage in the building.

A. General Requirements

The general requirements describe the essential processes in operating the pmCHP, regardless of the location of usage. Overall, they contain safety relevant issues, user experience, and general operating procedures.

First, the pmCHP has to choose the operational strategy, which decides how the pmCHP is going to be driven. 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. Each of the modes has different advantages, depending on the situation the pmCHP is employed in. For example, operating the pmCHP in heat-driven mode in a house secures government grants on electricity produced through the usage of the power-heat-co-generation, due to its high efficiency. However, choosing the wrong operational strategy will result in a bad user experience, in the BEV running in heat-driven mode will not result in higher ranges, as long as the passengers do not require air conditioning.

After choosing the general strategy for the pmCHP operation, operational plans have to be made. Ideally, the pmCHP is operated for long uninterrupted periods of time, with the combustion engine running in its optimal operating point. This decreases wear on the moving parts and also guarantees high efficiency with low fuel consumption, allowing the pmCHP to operate at the minimum costs to the user. For the operational plans, external inputs have to be considered, depending on the location of usage.

With the pmCHP up and running within optimal parameters, the device has to be monitored. Engine temperature, rpm, fuel flow, air flow, cooling flow, etc. have to be watched carefully to diagnose errors early and allow safe shutdowns. To accommodate this, the software has to handle many different sensor inputs and correlate the sensors into an operational state.

The state monitoring is needed for two different purposes, first and most important is the emergency shutdown. If the state of the pmCHP is critical, for example if the engine is leaking fuel and has caught fire, the pmCHP has to be shutdown safely, disconnecting it from other devices and stopping its operation. The shutdown has to be fast enough to prevent further damage and has precedence over every other procedure in the software. Current operational parameters, like operational plan and strategy as well as sensor readings, have to be logged for examination.

Also, errors and problems have to be reported to connected devices and the user of the pmCHP. In the connected smart grid or the car, other electronic control units exist and need to know the state of the network, i.e., if devices like the pmCHP are performing as the control unit expects them to. The user needs the information about the device state to make decisions on maintenance.

Lastly the pmCHP-software should give the user an overview over the current operational state and planning to increase transparency of the device.

The general functional requirements can be summarized as follows:

GR01	Smart choice of the operational strategy.
GR02	Creation of operational planning depending on external inputs and operational constraints.
GR03	Monitoring of device state.
GR04	Emergency shutdown on critical or dangerous state.
GR05	Reporting of errors or malfunctions to other devices and the user.

GR06 Displaying the current operational state and planning to the user.

B. Usage in BEV

After considering general requirements, we will present the requirements for the usage of pmCHP in a BEV. The usage in the car is dominated by the use of the pmCHP as a range extender, the requirements revolve around functions to increase range and comfort. An important point for the usage in the BEV is the lack of other electrical generators in the car. The generation of electrical power hence takes precedence over efficient usage of the pmCHP resources.

To facilitate operation in the car, the pmCHP has to monitor different aspects of the cars state. Most importantly, the pmCHP has to monitor the charge of the internal battery and the expected range depending on current consumption, planned route and expected consumption. If electrical energy is likely needed to complete the planned route, the pmCHP has to act and charge the battery before it depletes completely. Ideally, the pmCHP starts operation at the beginning of the drive to charge the battery and condition the passengers, so it can shut down when thermal requirements are met.

The primary function of the pmCHP is the generation of electrical power, if needed to continue driving. A different requirement is the operation as emergency range extender, when the battery is about to be depleted. Thermal energies produced during emergency mode are however considered as waste, therefore the pmCHP loses its efficiency advantage over conventional range extenders.

As already mentioned, the pmCHP shall condition the passenger compartment to increase passenger comfort during rides. The pmCHP-Software has to monitor the current temperature of the passenger compartment as well as the desired temperature. Furthermore, the pmCHP shall provide heat or cold (depending on the difference between desired and current temperature).

A more exotic requirement is the conditioning of the battery used for driving. Studies have shown that thermal conditioning of the battery can increase the range of an BEV considerably. While this would introduce additional thermal loads to be satisfied by the battery in a normal BEV, a pmCHP-equipped BEV can utilize the pmCHP to cover the thermal loads and profit from efficient co-generation of heat and power. To properly condition the car battery, the pmCHP has to monitor the batteries temperature and regulate it to the optimal temperature.

Summarizing these requirements results in the following list:

BR01 Monitoring of current range and expected driving distance.

BR02 Generation of electrical energy, if it is needed to continue driving.

BR03 Monitoring of the current passenger compartment temperature and desired passenger compartment temperature.

BR04 Conditioning the passenger compartment, if the battery can be charged.

BR05 Monitoring of the car battery temperature.

BR06 Conditioning of the car battery, if it will increase range or the battery can be charged.

C. Usage in the house

The usage of the pmCHP in the house is dominated by its connection to their devices. Integrated into a smart grid, the focus of operation changes to one of cooperation and coordination. Since there are a lot of alternative generators available in the house or the smart grid, the operation of the pmCHP focuses on optimal usage of co-generation effects. Connecting a pmCHP to a potentially global network with a lot of other clients opens the pmCHP to attacks, so security considerations come into play.

The first functional requirement is the proper communication of the pmCHP with other smart grid devices, via the expected smart grid standards like IEC61850. To facilitate coordination, communication is needed. By using the IEC61850 standard interoperability with other smart grid devices is ensured. Since no real modern smart grid is currently in use in Germany or the EU, strict adherence to the predefined standard is needed.

Another requirement is the monitoring of the attached heat storage. In contrast to the car, which has a battery, the house usually has no system for electrical storage (devices like the Tesla Powerwall exist, but are currently not in widespread use). However, most houses have some sort of thermal storage for heating and hot water, which can be used by the pmCHP to store thermal energy. This decouples the time of use from the time of generation and allows more flexibility for pmCHP operation.

The most important functional requirement is the efficient operational planning. The pmCHP-software needs to create efficient operational plans, which use the pmCHP to its maximum potential. To create the operational plans, the software needs to consider current and historical weather data, other energy producing devices within the smart grid, the operational specifics of the pmCHP (long, continuous periods of medium load), and the current price of electricity. The weather data shall be used to predict future loads for proactive generation. Other smart grid devices have to be considered during planning, since there might be no need for the pmCHP to run. Interestingly, the current price of energy also factors into the planning, since the pmCHP consumes fuel to generate heat and electricity. If buying electricity from another source is cheaper than running the pmCHP, it might be more sensible to not run the pmCHP.

In line with the monitoring of the current price of electricity, currently the German government gives grants on sales of electrical energy, if it was produced using the co-generation of heat and power. The pmCHP shall also tap into this stream of revenue by producing and selling electrical energy, if it is not needed in the users house, the heat can be stored and used later, and the profit from the energy sale exceeds the price of fuel.

Since the pmCHP is connected to a network when used in a house, additional functions should be provided by the software. First, the software shall provide the ability for remote maintenance, like software updates, error reporting and state monitoring. This allows the manufacturer to continuously improve upon the pmCHP software and, most importantly, fix security and safety issues in the software quickly.

In a similar fashion, the pmCHP-software needs to provide the ability of emergency control to the electrical grid operator.

Mainly a safety requirement, this allows for remote control of the pmCHP in case of emergencies, like blackouts or times of overproduction, which cannot be handled automatically.

Also, three requirements concerning security are imposed on the software. With hacker attacks on energy grids already being reality, protective measures have to be considered early in development of smart grid devices. The most important requirements for security are the protection of the hardware from malicious control, preventing physical damage to the pmCHP and its surroundings. For example, a hacker with free control over the pmCHP-control could run the pmCHP at a very high load, possibly causing a fire. With software protecting the pmCHP hardware and an update mechanism to change the software, the software itself has to be protected against unauthorized changes. Lastly the information of the user, like times of usage in the car have to be protected from attackers to prevent social or physical damage.

The requirements are summarized as follows:

HR01 Communication with the other smart grid devices.
HR02 Monitoring of the attached heat storage.
HR03 Smart creation of operational plans.
HR04 Sale of electrical energy, if the heat can be stored and profit can be made.
HR05 Provide remote maintenance support.
HR06 Provide emergency remote control to the grid operator.
HR07 Protection of physical components from attacks.
HR08 Protection of the software from unauthorized changes.
HR09 Protection of the users information.

IV. Requirements to software quality

After considering the functional requirements, quality requirements have to be looked at. Quality requirements do not define the softwares behavior like functional requirements, but instead describe the goodness of the software, which often is not directly measurable. Since the device and the projected usage environments are currently in development only a rough definition of quality requirements will be given.

The most important quality requirement is availability, which consists of robustness against errors in the system, time of restoration of service after failure and probability of failure. With the goal of improving the daily life of the user, failure of the software is very problematic as missing energy production can have large impacts to the users daily life. The direct interface to physical components makes failure very dangerous, therefore the availability is of very high importance, with short times of recovery after a failure and long times between failures.

Other very important quality requirements are safety and security, consisting of the security of the software (i.e., resistance to malicious use) and safety due to functional correctness. With pmCHP being part of the smart grid, they automatically become part of critical infrastructure, which needs to be protected extensively against attacks. Also, the direct cyber-physical-interface presented by the software requires high security and safety due to the large impact of malicious use or incorrect functioning of the software.

Also, rather important is the maintainability of the software, consisting of the ease of change and extension as well as conceptual integrity and testability. Since the software will be designed in a very early state of development of the pmCHP and the smart grid, the underlying requirements are not final and might change during the future. A good maintainability allows for easy, rapid change of the softwares components, which is critical in unclear scenarios. Adherence to the established smart grid standards also improves maintainability through ease of integration. According to the German 'Normungsroadmap 2.0' [9] three standards are primarily used in the Smart Grid.

- IEC 61850: Substation automation and protection
- IEC 61968/61970: Application level energy management system interfaces, CIM (Common Information Model)
- IEC 62351: Information security for power system control operations

We will concentrate on the IEC 61850.

However, performance is not as important as the other quality requirements, with the exception of the emergency shutdown and other hardware-related parts of the software. While parts of the software 'close' to the hardware are confronted with hard real-time requirements, those parts are considered to be very small. Other parts like the operational planning are not subject to hard deadlines, a late operational plan can always be executed a bit later.

Usability of the software is not very important, since the software is planned to be very autonomous, with little amounts of user interaction.

To summarize the following importances are derived for different quality aspects:

- Availability
  - High error tolerance.
  - High meant time between failures.
  - Low time of recovery after failure.
- Security/Safety
  - Protection against software manipulation.
  - Protection against abuse of the softwares functions.
  - High safety through functional correctness.
- Maintainability
  - High extendability.
  - High conceptual integrity.
  - High testability.
  - High standard compliance (esp. IEC 61850)
- Performance
  - Hard real-time requirements to the emergency shutdown / other close to hardware functions.
  - Soft real-time requirements elsewhere.
- Usability
  - Low usability requirements for the software.

V. Architectures

Based upon the aforementioned requirements, architectures can be developed. Instead of directly choosing an architectural style, we decided to first build a rough sketch and compare three different architectures based upon the sketch. By first designing the principal components of the software, we were

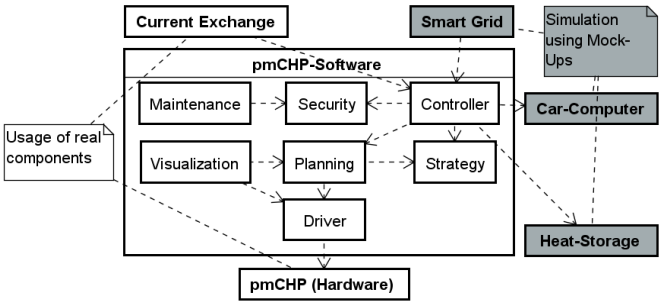

Figure 1. Rough sketch of the pmCHP-Software

able to keep conceptual integrity and continuity between the architectures, which allows proper comparison.

Figure 1 shows the principal components of the pmCHP software. Generally, the energy production will be handled on the basis of energy requests, which specify how much energy of which type is needed. The pmCHP can generate own requests and handle requests from external sources, this simplifies the handling of requests, since only a single mechanism will control the pmCHP.

The CONTROLLER coordinates the production of the pmCHP 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, since requests might not fit in the current operational strategy or the generation capabilities of the pmCHP. The CONTROLLER relies on the STRATEGY component for decision making. Valid requests are handed the PLANNING for further processing. The CONTROLLER fulfills a wide number of functional requirements, since it contains the main logic for generating and handling requests. In further designs it will be split into different components, based upon architectural style.

The component STRATEGY provides the framework for the day-planning, as it decides, which operational strategy is used. The decision for the heat-driven/electricity-driven or combined mode is based upon the detected environment of the pmCHP. It fulfills the requirement GR01.

The component PLANNING is responsible for planning the day-to-day-operation of the pmCHP, according to the operational strategy. To create plans, the component uses the energy requests received fro the controller and further information, like the current operational strategy, data about previous operation and forecasts. After a plan was created, they are handed over to the DRIVER for execution.

The component DRIVER provides the interface between hard- and software. Most importantly, it monitors the pmCHP and provides emergency shutdown functions as outlined in requirements GR03 and GR04. Furthermore, the DRIVER transforms the operational plan into control-commands 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 (GR06). Also, errors and warnings can be shown to the user, so corrective action can be taken if needed (GR05).

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. This covers the requirements HR05 and HR06, as outlined in the requirements for the operation in the house.

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. However, these can not be ignored, since the necessary interfaces in the pmCHP software need to be considered when designing the software.

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

A. *Service-oriented architecture (SOA)*

First, we will give an overview of the service-oriented architectural style itself, before we show our implementation of the SOA for pmCHP. The main drive behind SOA is a very loose coupling and high coherence in components [10]. A special form of SOA are the microservices, in which components are designed for easy rapid replacement. Since a SOA is highly flexible between components, the service oriented architecture is a good match for a cloud environment, in which components can be deployed across different locations without having to adopt the component to the deployment location. Very often web services using REST or SOAP are utilized to facilitate inter-component communication.

A general structure of a very comprehensive service oriented architecture is shown in Figure 2. In total, the general SOA consists of five layers and two cross-cutting parts.

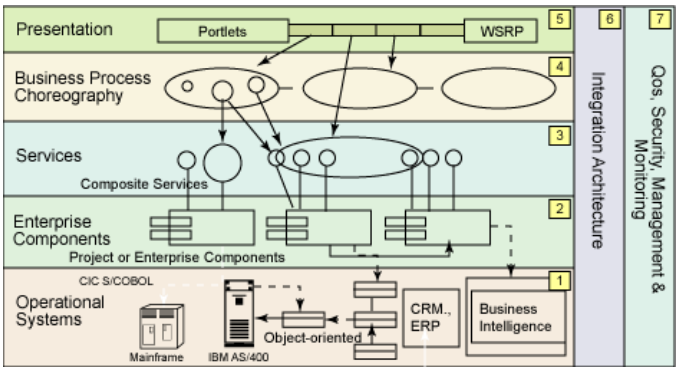

Figure 2. The service oriented architectural style according to [10].

The uppermost layer, also called the presentation layer (5) contains all components that allow interaction with the software system, like the graphical user interface or services that other software systems can use. To access the softwares functionalities, the presentation layer (5) accesses functions from the business process choreography layer (4). In this layer the business processes of the software are modeled and executed. It contains the major functionalities of the software, often

modeled in Business Process Model and Notation (BPMN), allowing domain experts the creation and validation of business processes and therefore functionalities. The business processes from layer 4 access the services in the service layer (3). A service is a defined interface enclosing single functions, independent of their implementation. The services can be modeled very fine grained allowing swift replacement. Generally, a business process accesses many services to complete, each service fulfilling a single step in the process. Services are independent of each other and can be recombined easily in new business processes allowing a high degree of re-usability. The services are implemented in applications contained in the application layer (2). An application might implement multiple services or just a single one, depending on the complexity of the desired functionality. Applications cluster services by semantics or other constraints (e.g., all services regarding a single database might be implemented in a single application). Rather unusual, Figure 2 shows the operational systems, which often are not considered in a cloud environment, since the SOA can be developed platform independent. Working in the background, the integration architecture (6) serves as 'glue' providing functions to connect all parts of the SOA. For example, a service bus is needed to connect the services to the implementation and present the endpoints to the business process choreography. Another important cross cutting part of the SOA are the quality monitoring and management components contained in (7). These allow the implementation of quality aspects like redundancy of services and security.

In the following section we are going to design an architecture based on the aforementioned style. Since the general business processes are already roughly designed a top-down approach is chosen for the design, starting at the business process, which will be decomposed into services grouped into applications, iteratively refining the softwares structure. For the services we follow a microservice-approach, defining services small enough so they can be replaced within a day. This allows for easy maintenance as shown later. Not all business processes and services will be shown, rather we will concentrate on the usage of the pmCHP in BEV or house, leaving, e.g., maintenance out of scope for this article.

1) Processing of energy requests: The main operation of the pmCHP revolves around energy requests. These stem from the smart grid, the car or the pmCHP itself and are the driving force behind the pmCHP-operation. An energy request contains information about the energy that is needed in the environment.

We differentiate between external and internal energy requests as shown in Figure 3. External energy requests are first

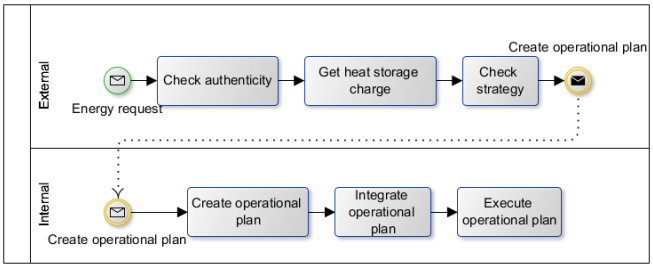

Figure 3. Processing energy requests in the SOA.

checked for authenticity and integrity to ensure no malicious use happens. Afterwards, the current charge of the attached heat storage unit as well as the current operational strategy, has to be checked. If the energy request does not fit the strategy or the generated heat can not be stored in heat-driven-mode, the energy request has to be declined. Otherwise the request meets all requirements and can be turned into an operational plan. This is shown in the internal part of the process, which is also the entry point for internally generated energy requests.

To fulfill the energy request, first a single operational plan containing the single energy request is created. The plan conveys information about the needed energy and the duration of the energy request (e.g., 1 kW of electrical energy for 1 hour). This plan is then integrated into a copy of the current operational plan, which replaces the original when the plan is executed afterwards. If the plan can not be integrated into the current operational plan for whatever reason (e.g., the needed energy exceeds the current production capacity), the energy request is declined and the execution of the original current operational plan continues.

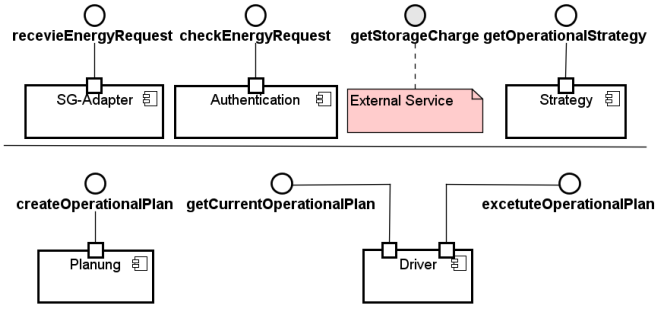

Figure 4. Services for processing energy requests in the SOA.

Figure 4 shows the services used in the business process as well as the applications implementing the services. The services are rather straightforward and fulfill a single responsibility indicated by their name, following the microservice-approach.

2) Energy requests in the BEV: The usage of pmCHP in BEV is a novel deployment of CHP-technology. Here three main business processes can be extracted from the requirements BR01-BR06, as shown in Figure 4. The three processes model the different operational strategies of the pmCHP, using it as a range extender (top), as an air conditioning unit (middle) and as a power conditioning unit (bottom).

In the usage as a range extender the pmCHP checks the currently planned route as well as the current battery charge. If the range is insufficient to complete the planned route, an operational plan has to be created to generate the necessary electrical energy. Using the pmCHP as air conditioner or power conditioning unit works similarly, as the physical attribute of the environment is measured and an operational plan is created to move the temperature to the desired or optimal value.

To create operational plans, the energy requests are sent to the aforementioned internal part of the energy request processing.

The services used for the usage of the pmCHP in the BEV are completely external as all of them have to be implemented

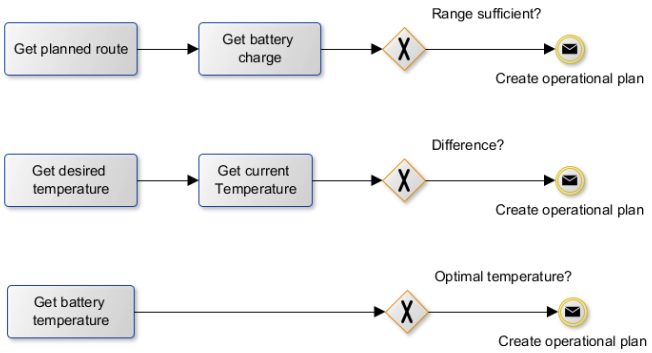

Figure 5. Business processes for the operation in the BEV.

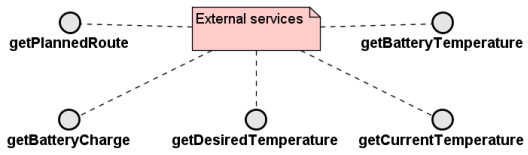

Figure 6. Services for the request creation within the car.

within the cars system software. While they are needed for the correct operation, they are not within the scope of our design.

3) Energy requests in the house: The second cornerstone of pmCHP-operation is the usage of pmCHP in a house, connected to a smart energy grid. Main requirements regarding energy generation are HR01-HR04. In the house, most operational plans are created upon external request through the smart grid, which was shown before. The design of business processes for the internal request generation required for HR04 is left as an exercise for the reader.

4) Overview: Continuing the design as a service oriented architecture, we result in the general architecture presented in Figure 7. Not all processes and services are shown.

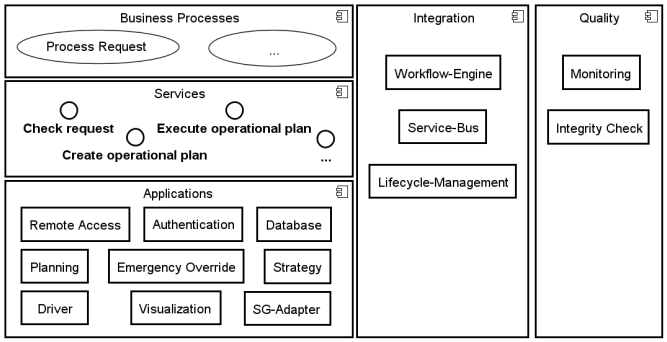

Figure 7. Overview of the complete SOA.

Excluding the presentation layer and the operational systems layer from Figure 2, our architecture conforms to the style presented before. In comparison to the rough sketch in Figure 1, the CONTROLLER was split into different business processes, as expected, while other components like the DRIVER were converted into applications with services. Furthermore, new components were introduced to support the service model, like the WORKFLOW-ENGINE providing orchestration or the SERVICE-BUS, which connects service endpoints to applications.

Overall, to create the SOA we transformed the requirements into business processes, which were decomposed into services. The services were grouped into the applications, which are closely aligned to our previous rough sketch. We achieve comparability to the other architectures by reducing our freedom of design.

B. Event-driven complex event processing (ED-CEP)

In the event-driven complex event processing architecture, processes are mapped to event chains, which start with simple events, later combined them into more complex events. Every event is consumed by an event processor, which either creates a new event, combines multiple events into one or calls an external service.

The general structure of ED-CEP-Architectures is shown in Figure 8.

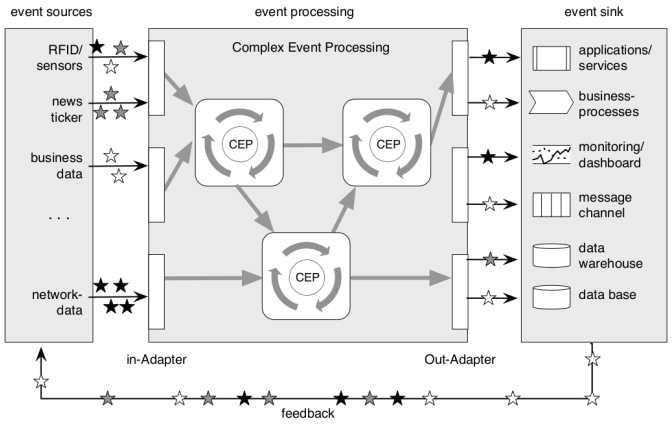

Figure 8. The event-driven complex event processing architecture according to [11].

Events originate in event-sources like sensors or news tickers and are fed into the event processing network (EPN). The EPN contains the event processors, which process different events according to their rule-set. Within the rules, conditions and actions are defined. The conditions can be based on the timing of events, values contained in events or the presence of a combination of events, etc. If a condition is met, the processor executes an action as outlined before. The events 'leave' the EPN through the event sinks, which are special processors that execute actions (e.g., call a certain service of an application) if they receive the corresponding event. Applications in ED-CEP are similar to applications in SOA and group functions that can be accessed by the EPN.

The ED-CEP is similar to the SOA, the functional requirements are modeled within the EPN instead of business processes while a 'framework' takes care of technical functions. It is therefore possible to exchange the orchestration layer of a SOA with an EPN.

To design an ED-CEP architecture, event sources are identified, their events and the transformation of those events are defined. The business processes are modeled as event chains often ending in a service call to an external application.

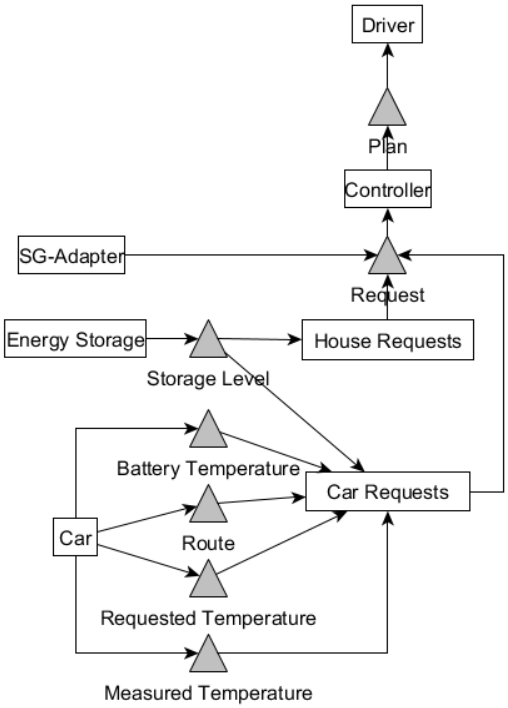

Figure 9. Creation and processing of energy requests in the EPN.

Similar to the SOA we will start by modeling the processing of energy requests as well as their creation in the car.

Figure 9 shows a part of the EPN with event sources marked as triangles and processors as rectangles. On the left side are the event sources involved.

The first event source is the SMART GRID-ADAPTER, which connects the system to the smart grid. Energy requests are received as messages over the network and transformed into *Request* events, which are processed by the CONTROLLER. The CONTROLLER contains the rules, which can be extracted from the business processes shown in the section about the SOA. For example, a (very simple) rule to process an external request might look like this:

```
IF(request.energyNeeded < pmCHP.availablePower)
  CREATE new Plan(Request);
```

The CONTROLLER then generates a new *Plan* event, which is consumed (executed) by the DRIVER.

For the usage in the BEV or the house energy requests are created by the EPN through other chains of events and processors, shown in the bottom half of Figure 9. The CAR is an event source for four different events, containing information about the cars current operational status, which are processed by the CAR REQUESTS processor. Again, the internal modeling of the processor rules can be extracted from the business processes used in the SOA and is therefore not shown here. (Each operational mode is modeled as a single rule, comparing the different events to each other and generating *Request* events if necessary.)

The complete event processing network for the pmCHP-software is shown in Figure 10.

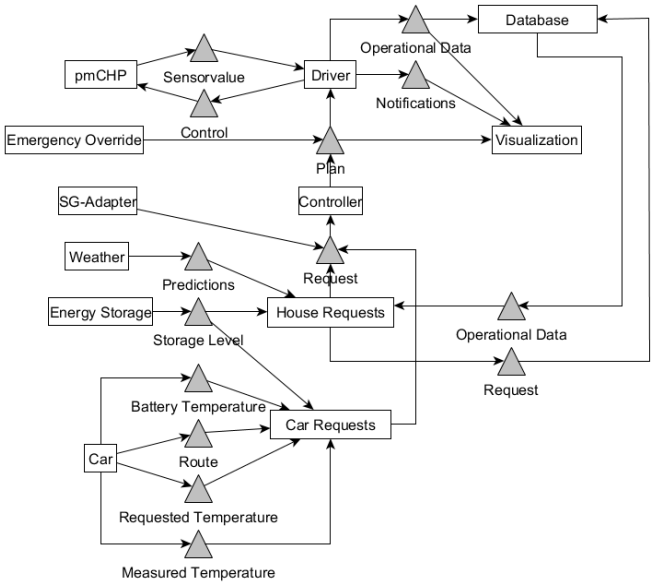

Figure 10. Complete EPN of the ED-CEP architecture.

As opposed to the SOA, there is no clear separation of the business processes but rather a complex network of events and processors with behavior hidden in the internal rules of the processors.

Additionally, further components are required as a framework for the EPN, shown in Figure 11.

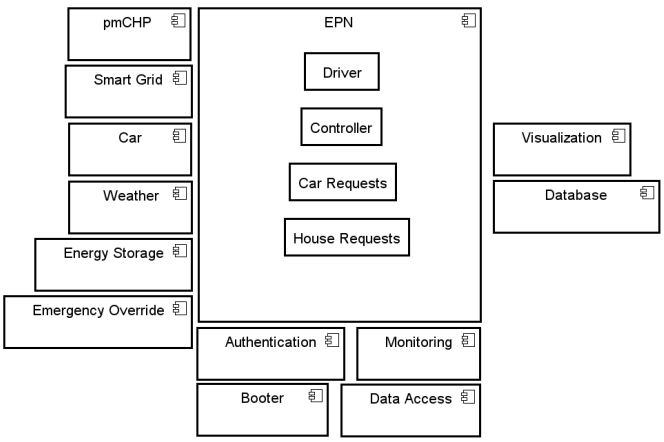

Figure 11. Overview of the ED-CEP architecture.

With the event sources on the left and the event sinks on the right, four components remain in the bottom. These four provide cross cutting functionality, which concerns the EPN or provides functions to the EPN. For example, the MONITORING monitors EPN performance and reliability and takes action if necessary to sustain availability.

Overall, in the ED-CEP business processes are mapped to event chains containing events and rules on how to transform the events into different events or service calls. The modeling of processes as chain of rules allows flexibility in program logic that is directly visible in program structure, as processes

are modeled as EPN components.

The similarity to the rough sketch of Figure 1 can be seen in the overview, as components like the CONTROLLER can be clearly identified. Also, a similarity to the SOA is visible, as the EPN can be seen as an orchestration layer for service calls. This confirms the coherent modeling of both architectures under the same constraints, allowing later comparison under non-functional aspects.

C. Layered Architecture

Lastly, we will present an architecture designed using the layered style. The layered style is one of the oldest architectural styles, dating back to a publication by Dijkstra in 1969 [12]. Generally speaking, the software is divided into multiple layers of increasing abstraction, from the concrete physical system towards the ideal software system. Layers can be observed in other styles as well, as already shown in SOA (Figure 2).

The general style of a layered architecture is shown in Figure 12.

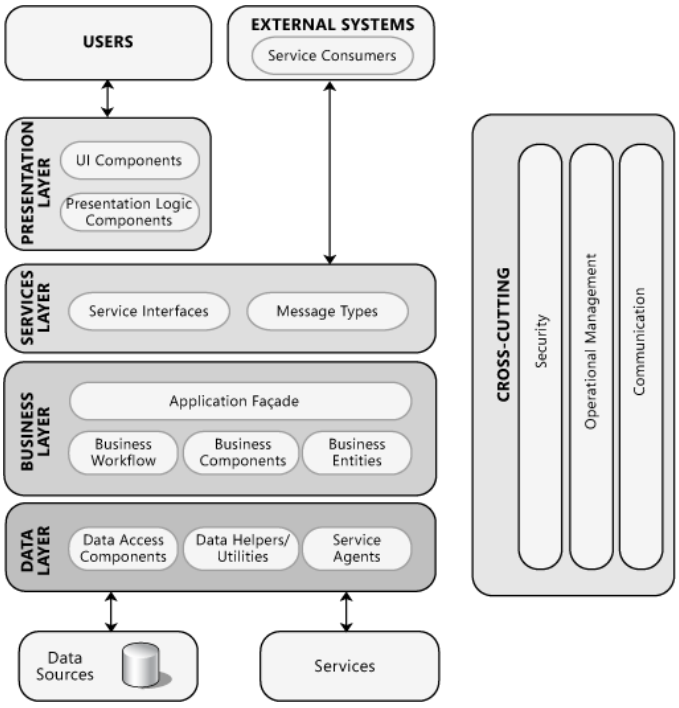

Figure 12. The layered architectural style according to [13].

Each layer can use functions from any layer below it, but not functions from layers above. At the top is the presentation layer, which provides users an interface to the software as well as some logic needed to show the user interface. Below is the service layer, providing interfaces for the presentation layer or external applications. To provide their function, the service layer accesses the business layer, sometimes through the facade, which hides the components of the business layer from external access, decoupling the components further. The whole business logic of the system is implemented within the business layer, comparable to the EPN in the ED-CEP or the business processes in the SOA. The lowest layer, also called the data layer contains technical parts of the system, which allow access to databases or other systems.

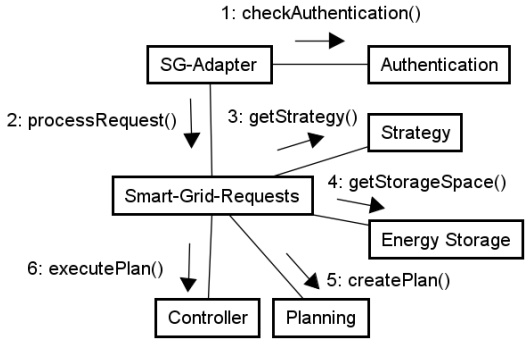

Figure 13. Processing energy requests from the smart grid in the layered architecture.

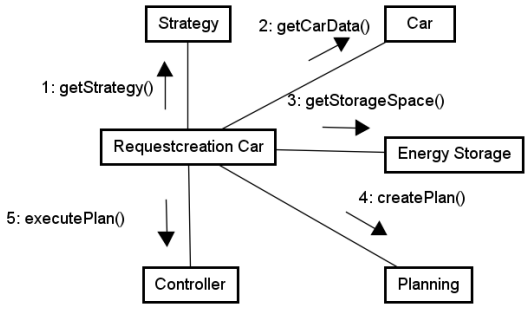

Figure 14. Request-creation for the BEV in the layered architecture.

Some aspects of the software however cannot be attributed to a single layer, since they are needed on all layers, like security. These cross cutting components are often distributed across all layers.

Designing a layered architecture can be done in multitude of ways, we will first model the most important business processes as function calls between components and the order the components into the appropriate layers.

Again, the process of processing an energy requests is modeled first, before we model the creation of energy requests in the car.

Energy requests are received from the smart grid by the SMART GRID-ADAPTER, which is responsible to check the request via the AUTHENTICATION, before handing the request to the SMART-GRID-REQUESTS component, which processes the request. To process the request, first the operational strategy and the current energy storage space is checked, similar to the process shown in the SOA or the ED-CEP. If the request is valid and can be fulfilled an operational plan is created via the PLANNING and sent to the CONTROLLER for execution. The process is modeled as function calls between components.

The same structure can be observed within the request creation for the BEV, shown in Figure 14.

First, the operational strategy is retrieved to determine the functionality needed at the moment (e.g., usage as range-extender or air conditioning). Afterwards the necessary data about the car, like route, temperature, etc. as well as current

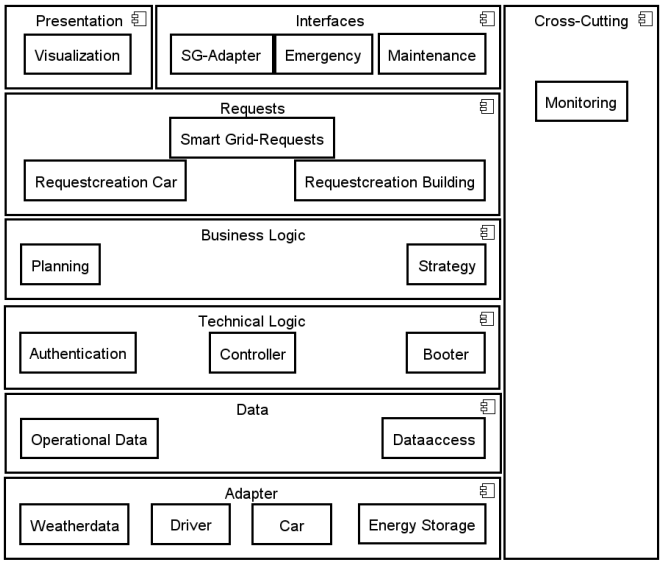

Figure 15. Overview of the layered architecture.

battery charge is retrieved before creating an operational plan to be executed by the controller.

Continuing the process for all business processes the final architecture can be seen in Figure 15.

As shown in the architectural style, the layered architecture contains a presentation and an interface-layer to facilitate access to the software functionalities.

Below those layers, the three logic layers are found. First the request layer, which is responsible of handling and creating energy requests to drive the pmCHP-operation. It is comparable to the business process choreography of the SOA shown in Figure 2. The business logic layer is found in the middle of Figure 15. Containing the PLANNING and STRATEGY components, the business logic layer is responsible for defining the core operation of the pmCHP. It is less abstract than the general request creation occurring above, but not as technical as the layers below, which are system specific. The bottommost logic layer is the technical logic, which contains components that implement system specific logic, like the CONTROLLER, which has to execute operational plans via the driver. Below we find a data layer, allowing access to different data sources within the pmCHP-system. At the bottom of Figure 15 the adapters are found, allowing access to different parts of the pmCHPs environment.

The layered architecture contains a lot of similar components when compared to the SOA or the ED-CEP with the most distinct difference being the spread of the business logic across the three logic layers. All three architectures are designed to be functionally identical, fulfilling all functional requirements allowing the comparison of the architectures without obvious deficiencies being visible at this point.

VI. COMPARISON OF ARCHITECTURES

Comparisons of architectures are a useful tool to increase software quality at an early stage of the development cycle [14]. 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.)

A. Scenario-based comparison

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 where the system has to take certain countermeasures to provide uninterrupted operation. Availability is the most important quality in an energy providing system [8].

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 where 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.

B. Application of scenarios

To evaluate the architectures, the aforementioned scenarios are applied to the architectures. For each scenario, necessary changes to the architecture are tracked, differentiating between easy and complicated changes. Easy changes are changes that are most likely to be completed within a day, while complicated changes will most likely take longer than three days. With the scenarios being drawn from different use cases within the softwares life, changes to a lot of components to accommodate a single scenario indicate strong coupling. If only a single component has to be changed for a scenario loose coupling is visible, if no change to any component is necessary the scenario showcases a previously unspoken requirement that was accidentally fulfilled.

Furthermore, scenario interactions are tracked. Two scenarios interact, if they require changes to the same component, indicating a low coherence of the component. If a component is only responsible for a single thing, no two different scenarios will not interact if the scenarios are separate (not two variations of the same situation). A high amount of scenario interactions indicate a badly defined component.

a) Availability: First, we will evaluate the architectures using the scenarios regarding software availability, starting with Scenario Ava01.

Ava01 demands the monitoring of the DRIVER component within the software. A component to monitor the driver can already be found within all three architectures, therefore no additional components are needed. Changes to the MONITORING component are the simple inclusion of the monitoring and restarting of the driver, which we classify as an easy change.

Ava02 requires the Driver to ignore incorrect measurements caused by normal sensor jitter. Again, the no new component has to be introduced, since all architectures already contain a DRIVER, which can be changed. However, only in the ED-CEP this change can be done easily, since the rule-based logic allows for the easy definition of complex conditions. In the SOA or the layered architecture the change is rather complex, requiring a lot of work to correctly filter incorrect data.

Ava03 describes the disconnect of the pmCHP from the Software, an error has to be displayed. All three Architectures already display the status of the system to the user, therefore no change to the architectures has to be made.

Ava04 puts a heavy load of requests on the software and demands continuous service. A simply threshold can be introduced into the corresponding component of the three architectures, discarding all requests beyond that threshold. In the SOA, the WORKFLOW ENGINE creates new processes for every request, a simple change adds a threshold to concurrent processes. In ED-CEP or the layered architecture, the threshold can easily be added to the SMART GRID ADAPTER.

b) Security: Next on the list are the scenarios concerning software security.

Sec01 aims to prevent abuse of software functions, authentication of requests prevents unauthorized or malicious use of the pmCHP. All three architectures already ensure authenticity of requests, no change is needed.

Sec02 is rooted in the softwares update mechanism required by the requirement HR05, allowing software changes in the form of updates. To ensure software integrity the BOOTER within ED-CEP and layered architecture as well as the INTEGRITY CHECK in the SOA are already used to check integrity on startup. No changes are necessary.

Sec03 requires internal shielding of software functions to prevent access to critical functions from unauthorized components. In the SOA all calls to services are done via the SERVICE BUS, which can easily be extended with an access control list, a single easy change is counted. The ED-CEP requires a lot more modification. First, all events consumed by the DRIVER as well as the DRIVER itself have to be modified to allow for authorization checks using a token based system. Additionally, the EMERGENCY CONTROL as well as the PLANNING components have to be modified to use the new token system. Therefore, we count four changes, one of which

is a difficult change of the DRIVER. Similarly, the layered architecture requires extensive changes to the DRIVER as well as easy changes to the VISUALIZATION, CONTROLLER and the MAINTENANCE component.

c) Safety: After ensuring availability and security, safety is our primary concern.

Saf01 describes an emergency shutdown of the pmCHP, which is already part of the DRIVER within each architecture. No changes are necessary.

Saf02 requires the DRIVER to check of every command it receives, to prevent damage to the pmCHP. While the Driver already exists, this was not part of the original functionality, therefore, changes are necessary. To add the checks to the DRIVER is relatively easy within the CEP, since complex conditions can easily be translated into rules. Adding checks to SOA or the layered architecture however is rather complicated since no rule language is available.

d) Maintainability: Maintainability is important over the softwares life, as changes might arise, which are not expected at the moment of its design.

Mai01 plans the integration of the pmCHP into the smart market, adding new features. For this a new component BILLING is needed, which monitors the execution of requests and bills the produced energy to the consumer. In the SOA the process to process an energy request has to be extended by the billing of the customer. A new service BILLING has to be added, which needs to be mapped by the SERVICE BUS. The BILLING service accesses process variables already tracked by the MONITORING to fulfill its function. Three easy changes are required. The CEP architecture requires more extensive changes. The *Request* events have to be extended with data about the consumer to be billed. These changes have to be integrated into the CONTROLLER, *Plan* and the DRIVER. Furthermore, the DRIVER needs to create *Bill* events, which are consumed by the BILLING processor to bill the customer. Five components need to be changed and two new components need to be introduced. The layered architecture requires similar changes, since the customer data needs to be introduced to the SMART GRID-ADAPTER and the SMART GRID-REQUESTS component. Additionally, the SMART GRID-REQUESTS component needs to monitor the execution of the operational plan, billing the customer after the energy was provided. To access billing functions, a new adapter BILLING is introduced. This results in two changes and a single new component.

Mai02 describes the unit test of the DRIVER, requiring a mock of all functions needed by the DRIVER. This is easily achievable in all architectures, since the physical pmCHP is designed as an external system, no changes are necessary.

Mai03 requires conceptual integrity within the architectures. After reviewing the architectures we conclude that no changes are necessary to the architectures, as similar problems are solved in similar ways in our designs.

e) Performance: The performance of the pmCHP-Software is not as important as the security and safety, since all time-critical functions are contained within the operating system of the pmCHP.

Per01 demands a 'fast enough' shutdown of the pmCHP in critical situations. This is out of scope for our architectures, therefore no changes are necessary.

f) Usability: Ease of use is a desirable trait for all software systems, but is the least important quality concern for the pmCHP-Software, since almost no user interaction is planned.

Usa01 describes a single button startup of the pmCHP as well as a stop upon pressing the same button. This functionality requires the initialization as well as the correct shutdown of all software components. In the SOA this is already provided by the LIFECYCLE-MANAGEMENT, no change is necessary. The CEP and the layered architecture provide a BOOTER, which only takes care of proper initialization and has to be modified to also provide a shutdown functionality. A single change is necessary to CEP and layered architecture (including a renaming of the BOOTER, which no longer only boots the software).

Usa02 needs the display of status information to user at all times. The component VISUALIZATION is already present in all architectures for this exact purpose.

g) Summary: The results of the scenario based evaluation is presented in Tables I and II.

First we will concentrate on Table I, containing the count of changes to the architectures.

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

A lot of easy changes to the ED-CEP architecture and a lot of difficult changes to the layered architecture are evident in the results.

The numerous changes to the ED-CEP are based in its structure, often a lot of small changes to components or new events had to be introduced. Especially the internal shielding required a lot of changes to the architecture. However, changes to the DRIVER often were easy in the ED-CEP but difficult within the SOA or the layered architecture. This is a result of the rule-based nature of the ED-CEP, which is built for easy modeling of complex conditions.

The high amount of difficult changes to layered architecture originate from the scenarios requiring pattern recognition (e.g., Saf01), which is difficult to introduce without specialized tools like the rule language used in the ED-CEP.

The SOA however requires the least amount of changes. Especially the easy extension and modification of services through the use of microservices keeps the necessary changes

small. Also, the SERVICE BUS provides a central place to introduce software specific functions like internal shielding.

Table II shows the scenario interactions that occurred during the application of the scenarios.

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

SOA		ED-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		

A lot of scenario interactions are evident for the ED-CEP architecture. This originates from the spread of functionalities over a high amount of small components, which results in unclear boundaries of responsibility.

However, the SOA and the layered architecture suffer from interactions as well, mostly through the DRIVER, which seems to be responsible for multiple scenarios. Splitting the DRIVER into multiple components might be useful.

VII. EVALUATION

Considering the previous results two rankings can be created. First, the architectures are rated by the changes necessary to accommodate the scenarios. The changes are summed up with difficult changes contributing threefold to the score.

1) SOA using microservices (11 points)
2) ED-CEP (18 points)
3) Layered architecture (19 points)

As previously mentioned the amount of necessary changes to the architecture is an indicator of its coupling with a high amount of points indicating a strong coupling. A strongly coupled architecture is difficult to adapt to new circumstances, since every change touches a lot of parts of the architecture making development complicated. This is considered a negative trait of architecture. Therefore, we consider the SOA the best architecture under this metric with ED-CEP and the layered architecture being similarly bad.

To rank the architectures using the scenario interactions, the interactions are simply summed up.

1) SOA using microservices (4 points)
2) Layers (5 points)
3) ED-CEP (10 points)

The scenario interactions provide an indicator for the quality of the component definition, i.e., how clear the functionalities are defined that a component should provide. A high amount of interactions indicate badly defined components that have no clear responsibility. This leads to feature envy in components that accumulate a lot of different functionalities, making changes complicated and the replacement of components a lot of work. Also, a negative trait of software architecture, badly defined components decrease maintainability and are considered bad. We conclude that the SOA contains the most well defined components, with the layered architecture trailing closely.

Considering both rankings we choose the service-oriented architecture for the integration of the pmCHP into the Smart Grid.

VIII. CONCLUSION

In this article, we first presented the problem of distributed power generation in the smart grid and range extension in the BEV. The problems are tackled by the pmCHP as it is developed at the *University of Applied Sciences and Arts Hannover*. To integrate the pmCHP into a smart grid an architecture had to be chosen.

After presenting the requirements posed to the architecture, we designed three different designs using different architectural styles. This was done to find the optimal architecture to implement. To compare the architectures, we presented scenarios of five different aspects of software quality before applying the scenarios to the architectures.

Considering the evaluation results we conclude that the SOA is the best suited architecture to integrate the pmCHP into the Smart Grid. The SOA provides the highest amount of flexibility when compared to our ED-CEP or layered approaches. Especially the usage of microservices helped to define clear functional boundaries and ensure loose coupling within the architecture, both considered to be good traits within software. We conclude that using the SOA design, we will be able to easily accommodate changes in the project field of usage, e.g., adoption to specific BEVs. However, this result only holds true for our narrowly defined domain and might be different for other use cases, like the usage of pmCHP in planes.

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] R. Pump, A. Koschel, and V. Ahlers, "On microservices in smart grid capable pmchp," SERVICE COMPUTATION 2018, The Tenth International Conference on Advanced Service Computing, pp. 30–35.

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

[3] 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.

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

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

[6] 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.

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

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

[9] V. V. der Elektrotechnik Elektronik und Informationstechnik eV, "Deutsche Normungsroadmap: Energy/Smart Grids 2.0."

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

[11] 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.

[12] 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.

[13] patterns & practices Developer Center, "Microsoft Application Architecture Guide, 2nd Edition," https://msdn.microsoft.com/en-us/library/ff650706.aspx, October 2009, last visited: 26.08.2016.

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