IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 19, 2020, accepted October 11, 2020, date of publication October 14, 2020, date of current version October 23, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3031189

An Overview of Design Patterns for Self-Adaptive Systems in the Context of the Internet of Things

CHRISTIAN KRUPITZER[1,2], (Member, IEEE), TIMUR TEMIZER[3], THOMAS PRANTL[1], AND CLAUDIA RAIBULET[4], (Member, IEEE)

[1]Software Engineering Group, Julius-Maximilians-Universität, 97074 Würzburg, Germany
[2]Department of Food Informatics, Universität Hohenheim, 70599 Stuttgart, Germany
[3]Chair of Information Systems II, Universität Mannheim, 68161 Mannheim, Germany
[4]Dipartimento di Informatica, Sistemistica e Comunicazione, Universitá degli Studi di Milano-Bicocca, 20126 Milan, Italy

Corresponding author: Christian Krupitzer (christian.krupitzer@uni-hohenheim.de)

This work was supported by the Open Access Publication Fund of the University of Würzburg.

ABSTRACT The Internet of Things (IoT) requires the integration of all available, highly specialized, and heterogeneous devices, ranging from embedded sensor nodes to servers in the cloud. The self-adaptive research domain provides adaptive capabilities that can support the integration in IoT systems. However, developing such systems is a challenging, error-prone, and time-consuming task. In this context, design patterns propose already used and optimized solutions to specific problems in various contexts. Applying design patterns might help to reuse existing knowledge about similar development issues. However, so far, there is a lack of taxonomies on design patterns for self-adaptive systems. To tackle this issue, in this paper, we provide a taxonomy on design patterns for self-adaptive systems that can be transferred to support adaptivity in IoT systems. Besides describing the taxonomy and the design patterns, we discuss their applicability in an Industrial IoT case study.

INDEX TERMS Design patterns, Internet of Things, IoT, self-adaptive systems, software engineering.

I. INTRODUCTION

Design patterns represent well defined and widely applied solutions to specific problems. They were first introduced by Gamma *et al.* [1] in 1994. Since then, the design patterns research has known an increasing trend both in the number of patterns as well as in popularity and application. The reason for this is that software engineers have recognized their advantages immediately, especially, capturing best practices and lessons learned during software development. Design patterns play a central role both in forward and reverse engineering. Using them in forward engineering increases the software quality, its readability, and its documentation. In reverse engineering, design patterns help to understand the software and the rationale behind the development solutions [2].

Usually, when a new research field raises, software engineers tend to reuse available development knowledge. Internet of Things (IoT) and the related concept of Cyber-Physical Systems (CPSs) both interact intelligently with users in a dynamic environment. Further, those systems integrate heterogeneous software and hardware resources as well as data from various resources. Hence, those systems have to adapt to changing environmental conditions and the system's dynamics to preserve their quality of service. Those reactions are aggravated regarding the distributed decision making in IoT and CPSs. For those adaptations as direct reactions to changes in the system environment, IoT systems require context-awareness, autonomy, decision making under uncertainty, and decentralized control.

Self-adaptive systems (SASs) are able to change their behavior at runtime as a response to changes in their environment or in the system itself [3], [4]. Those systems are able to work in dynamic and uncertain environments. They are often divided into a managed subsystem, i.e., software and hardware resources that interact with the users or back-end systems, and a managing subsystem, which is able to control and adapt the managed subsystem. As a de facto standard, the managing subsystem implements the Monitor-Analyze-Plan-Execution-Knowledge (MAPE-K) system model [5] for structuring the required management functionality into (i) monitoring the environment and the system resources,

The associate editor coordinating the review of this manuscript and approving it for publication was Michael Lyu.

(ii) analyzing if an adaptation is required, (iii) planning the necessary adaptation actions, and (iv) executing those actions. Those functionalities can be complemented by a shared (distributed) knowledge repository. Other authors propose similar feedback structures, such as the sense-plan-act control [6], the autonomic control loop [7], or the observer/controller architecture [8].

Both types of systems, SAS and IoT, operate in dynamic environments, resulting in uncertainty about the system environment. Accordingly, the exact requirements for those systems are hard to determine at design time. Hence, adaptive behavior, as know from the SAS domain, is also beneficial for IoT systems. Furthermore, to achieve this adaptive behavior, IoT systems must be context-aware, i.e., those systems must be able to monitor their context/environment and react to changes. Lastly, IoT and SAS systems are systems-of-systems, i.e., composed of several interacting resources resulting in highly distributed systems. Due to the significant commonalities between IoT and SASs – dynamic environments, context-awareness, uncertainty, adaptivity, distributed nature – it seems beneficial to discuss the application of Software Engineering practices from the field of more established SASs community in IoT. The link between SASs and IoT has also been inserted in the research roadmap by Vermesan *et al.* [9], who mentioned the need for ''distributed self-adaptive software for self-optimization, self-configuration, self-healing''. Hence, our work comes to enable and support IoT systems' engineering through design patterns already successfully applied for SASs. Currently, there are available several design patterns for SASs, some of them having overlapping parts, some providing alternative solutions. Therefore, it is not trivial for non-experts of SASs to have an overview of these patterns and choose the appropriate one for the system under development. A taxonomy of design patterns for SASs would be extremely useful because it would generate awareness of their existence in suitable application domains as IoT or CPS, and it would provide an overview of these design patterns. Implicitly, the application of design patterns would improve the quality of the systems and their understanding. To the best of our knowledge, there is no such taxonomy available for self-adaptive specific design patterns.

Based on our experience [10], [11] on design patterns [12], [13] and SASs [4], [14]–[16], in this paper we analyze design patterns for SASs under the lenses of their applicability for IoT systems given the commonalities that both share. Our contributions are threefold:

- **Literature Review:** We do an exhaustive literature review to identify design patterns for SASs.
- **Taxonomy:** We classify the patterns according to their development purpose.
- **Application:** We discuss the potential application of those patterns in IoT systems.

The remainder is structured as follows: Next, we describe related work (Section III). Then, we describe our research methodology (Section IV) as well as our derived taxonomy (Section V). After, we show its application in an Industrial IoT (IIoT) use case (Section VI) and discuss the threats to validity (Section VII). Finally, we conclude the paper with a summary and future work (Section VIII).

II. SELF-ADAPTIVE SYSTEMS IN A NUTSHELL

There is no unique and precise definition of SASs in the scientific literature [4], [17]. Several definitions that outline various facets of SAS have been proposed. For example, Garlan *et al.* mentioned that architectural adaptation focuses on the changes made at run-time in the structure of the components of a system and/or in the interactions among them by using an architectural model of a system [18]. McKinley *et al.* outline that compositional adaptation regards the modifications of a software's structure and behavior made at run-time due to the changes that occurred in its execution environment. This is achieved by exchanging structural and behavior components among them in order to enable software to fit better to its current environment [19]. Bastide *et al.* focus on structural adaptation of a software component that consists of updating its structure while preserving its behavior and services [20]. While behavioral adaptation focuses on the changes made dynamically in the execution of software components in a non intrusive way (e.g., by changing its configuration or by intercepting its requests and replies) as sustained by Gorton *et al.* [21]. Content adaptation is defined through the transformation and manipulation of contents (e.g., images, audio, video, text) based on the application's features or device requiring them [22]. While service adaptation is translated into content as well as a behavioral adaptation [23]. In [4], we add the relevance to explicitly include context-adaptation through the SAS into the reasoning process for adaptation in addition to monitoring the context as a trigger for adaptation.

All these definitions have in common three main characteristics: (1) adaptivity is requested by changes occurred internally inside a system and/or externally in its execution environment; (2) adaptivity consists in changes performed by the system itself in its execution environment, and (3) adaptivity is performed at run-time [15], [24], [25]. Recently, Danny Weyns introduced two basic principles [17] which determine what a SAS is:

- *The external principle*: a SAS can handle changes and uncertainties in its environment, the system itself, and its goals autonomously (i.e., without or with minimal human interference).
- *The internal principle*: a SAS comprises implicitly or explicitly two distinct parts: the first part (see Figure 1- the Managed System) interacts with the environment and is responsible for the domain concerns (i.e., its functionality); the second part (see Figure 1 - the Managing System) interacts with the first part (and monitors its environment) and is responsible for the adaptation.

The managed part of a SAS provides the functionality, i.e., the system's services to its stakeholders. Examples are the

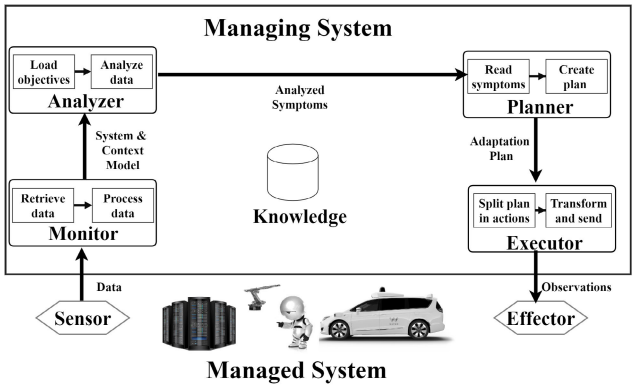

FIGURE 1. A conceptual model of a SAS.

self-driving vehicle or manufacturing systems. The managing part implements the self-adaptive mechanisms. This part should implement implicitly or explicitly four main steps [5]:

- monitor the environment and the system itself;
- analyze the information gathered during the monitoring;
- plan changes if the results of the analyze step indicate a need for adaptation;
- execute the planned changes.

These four steps (also called MAPE or MAPE control feedback loop) may share and/or exploit the knowledge (becoming MAPE-K) built from the monitored environment, the analyzed information, the planned changes, and the result of the execution of the changes. This knowledge may grow in time being enriched with new information about the environment and the applied adaptations.

Engineering the managed part of SAS is a challenging task, especially to the gap between design time and runtime [26]. As SAS operate in highly uncertain and dynamic environments, the set of requirements for those systems are often incomplete or even unknown at design time. Accordingly, those systems have to engineer themselves through adaptation at runtime. Therefore, each of the four steps of the MAPE loop may require a more or less complex subsystem to be engineered. Various approaches and solutions to design and implement these steps to improve the SAS' performance and avoid introducing significant overheads in the interaction of the two parts of a SAS have been proposed. In this context, design patterns for SAS aim to capture the successfully applied solutions for each of these steps, as well as for the coordination, interaction, and management of the various parts of a SAS. While MAPE-K may be considered an architectural pattern because it deals with the structure and interaction of the main elements of a SAS managing part, design patterns have a more narrow scope focusing on the engineering of the MAPE-K steps (or parts of them). Their role is determinant because they can be seen as the building blocks of the MAPE-K loop. Therefore, in this paper we focus our attention on the design patterns defined in the context of SAS, which may be adopted, adapted, and applied for IoT. As both categories of systems — IoT and SAS — operate in dynamic environments leading to uncertainty, both require integrating context-awareness and adaptivity. Accordingly, it seems beneficial to discuss applying principles for adaptivity from the field of the more established SASs community in IoT. In this context, design patterns facilitate the interplay between the managing system as control logic and support the distributed nature of IoT systems and can support developers of IoT systems.

III. RELATED WORK

This section presents related work in the context of this paper, i.e., overviews on design patterns for adaptive systems / IoT as well as specific definitions of patterns. The relevant literature can be grouped into (i) design patterns for SASs, (ii) design patterns related to IoT systems, as well as (iii) software engineering for IoT. Before presenting this related work, we briefly describe what design patterns are and which are their main objectives.

A. DESIGN PATTERNS

Designing software is not easy; designing good, i.e., qualitative, software is even more challenging. Design patterns are descriptions of communicating entities that are customized and adapted to solve a general design problem in a particular context [27]. The main objective of design patterns is to capture design experience and simplify reuse: developers create new software solutions based on previous experience and on previous successful designs through patterns. Hence, patterns make software flexible, elegant, and reusable. Developers aware and familiar with design patterns may apply them to design problems without having to rediscover them. Patterns are knowledge about design issues and related solutions. Therefore, in this paper, we provide an overview of design patterns successfully applied for SASs that can be adopted in the context of IoT. There are various types of patterns for software design: architectural, design, and idioms. Architectural patterns concern the design of software architectures and have a broader scope than design patterns. Examples of architectural patterns include client-server or model-view-controller. Idioms concern programming languages and have a more narrow scope than design patterns. Examples of idioms for Java include good practices of using the `equals()` or `compareTo()` methods.

B. DESIGN PATTERNS FOR SASs

Puviani *et al.* [28] propose a taxonomy of self-adaptation patterns based on various composition mechanisms focusing on architectural patterns rather than design patterns. The authors base their taxonomy on service components and component ensembles to derive architectural patterns supporting self-adaptation. Juziuk *et al.* [29] present a literature review focusing on design patterns for multi-agent systems. The authors conclude that there is a lack of (i) a standard design pattern description template hampering the use of design patterns amongst system designers and (ii) the description of associations between patterns. However, design patterns for multi-agent systems are applied in

several applications. Musil *et al.* [30] propose new design patterns capturing best practices for self-adaptation in CPSs. Giese *et al.* [31] describe several architectural patterns that describe reflection in self-aware computing systems. Whereas the former approaches target system domains close to SASs, Ramirez *et al.* [32] identified twelve design patterns which concern the main steps of the MAPE-K control loop. These patterns are grouped into three categories: monitoring, decision making, and reconfiguration. Their work integrates structures that the authors may not describe as patterns. Additionally, due to their distributed nature, researchers study decentralized control structures in the field of SASs. One prominent example of decentralized self-adaptation is presented by Weyns *et al.* [33]. They propose a reference model for decentralized self-adaptation. However, they do not focus on the definition of design patterns for self-adaptation.

C. DESIGN PATTERNS FOR IoT

IoT is a paradigm with a significant number of design patterns adopted from various research fields. In the following, we provide an overview of the IoT focused design patterns. Inspired by the agent-based modeling, Jung *et al.* [34] introduce three design patterns to address the heterogeneity of the IoT devices. The three patterns are described in an informal and unstructured way. Inspired by Edge Computing, Qanbari *et al.* [35] introduce four design patterns for IoT concerning the configuration and implementation of applications. They describe each of these four patterns by indicating their name, the problem they address, the appropriate context of the application, the motivations (i.e., use case scenarios), and the solution details with a sketch, i.e., the results of the pattern application. Vega-Barbas *et al.* [36] focus on the human-related aspects of IoT in smart spaces and define five interaction patterns that aim to capture the ''good manners'' of user interaction in IoT. Sithole and Marchall [37] present an exciting work on the attributes extraction for a fine-grained description and differentiation of the IoT patterns. This approach aims to provide an insight into IoT patterns with the objective to quickly and efficiently differentiate them based on various aspects. The authors mention that they have considered 109 IoT patterns (33 from informal Web pages,[1] the others from peer-reviewed and published articles). However, no details on these patterns are provided. Rahman *et al.* [38] propose a generic definition of an IoT pattern together with its formal specification based on the modeling concepts and relations. The authors focus on the managerial conflicts of applying a pattern derived from two types of managerial control, i.e., data control and behavioral control. Reinfurt *et al.* [39] extracted eight IoT design patterns from a large number of IoT solutions. They sustain that these patterns help in understanding the code design principles for developing IoT solutions. The same authors have proposed six security patterns for IoT in [40] and six device energy patterns in [41]. Cruz and Abreu [42] summarize 22 patterns for energy efficiency in mobile applications. The authors argue that these patterns may be of relevance for domains such as IoT and CPS. Pape and Rannenberg [43] show how seven available privacy patterns may be applied to IoT/cloud computing/fog applications by using a smart vehicle case study. The authors underline that the choice of the privacy patterns is use case driven. Summarizing, the cited papers concerning design patterns for IoT neither address adaptivity nor describe design patterns for self-adaptation.

[1] e.g., https://community.arm.com/iot/b/blog/posts/design-patterns-for-an-internet-of-things

D. SOFTWARE ENGINEERING FOR IoT

There are some significant surveys and overviews concerning software engineering for IoT. Weyrich and Ebert [44] underline the available reference architectures and their evolution in industry and academia. Bader *et al.* [45] classify the reference frameworks in IIoT and discuss their concerns. Sethi and Sarangi [46] provide a survey on methods, protocols, and applications in IoT. They propose a taxonomy for research in IoT technologies based on the architectural elements: sensors, communication, middleware, and applications. Di Martino *et al.* [47] review the most commonly used architectural solutions, both standardized and commercial, for IoT systems by focusing on security and interoperability. Mocrii *et al.* [48] address the main technologies, components for communication as well as privacy and security issues for IoT-based smart homes. However, none of those surveys focus on design patterns.

E. INDUSTRIAL IoT

One technological domain within the IoT context that heavily benefits from adaptiveness is the Industrial IoT (IIoT), i.e., smart manufacturing systems, or Industry 4.0, respectively. For those smart manufacturing machines, adaptiveness is a key aspect to achieve the system goals, especially, to enable the production of individualized goods. Several surveys exist in the field (e.g., [49]–[51]). Those surveys mainly target relevant technologies, architectural models, process models, or research challenges. Caesar *et al.* [52] discusses research challenges related to the adaptiveness of systems required in the IIoT. Cha *et al.* [53] analyses and compares meta-model for model-based smart production systems. In [54], the authors present an architecture for event-driven manufacturing information systems following the Industry 4.0 vision. Most closely to our work, Bloom *et al.* [55] presents design patterns for IIoT. They identified six design patterns that subsume the architecture and data flow in IIoT applications. However, to the best of the authors' knowledge, there does not exist design patterns for IIoT systems that focus on the support of adaptiveness.

F. DELINEATION

Generally, literature does not provide a literature review on design patterns for SASs. Moreover, none of the other studies do a large-scale search and comparison of such patterns with the objective of a structured representation of the state of

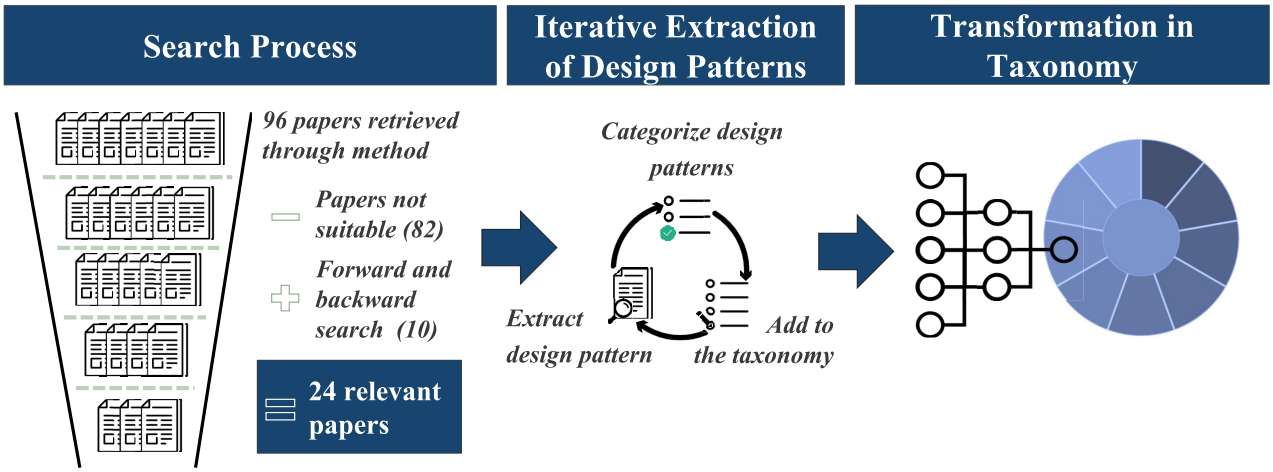

FIGURE 2. Process of literature identification, pattern extraction, and creation of the taxonomy.

the art. If so, than the authors mainly focus on centralized design patterns or the use of the design patterns provided by Gamma *et al.* [27], however, on a limited scope. This paper tries to close this gap and contributes to the existing body of research on design patterns for (self-)adaptive systems with a taxonomy on design patterns, focusing on (but not limited to) decentralized control in SASs which is highly relevant in the context of IoT as those systems are by definition composed of many distributed resources that have to cooperate and adapt. On the other hand, we outlined design patterns that come from various IoT-related research areas such as edge computing, agent-based systems, and human-machine interaction. Consequently, in this paper we review design patterns that support the adaptivity of SASs for supporting the IoT.

IV. RESEARCH METHODOLOGY

We derived our taxonomy following a systematic literature review process [56], [57]. Figure 2 shows a flow diagram of the whole literature selection process. The main search terms used for the initial paper screening process were:

- `design pattern(s) AND self-adaptive system(s)`,
- `design pattern(s) AND adaptive system(s)`,
- `pattern(s) AND self-adaptive system(s)`, and
- `pattern(s) AND adaptive system(s)`.

We included Google Scholar, IEEEXplore, ACM Digital Library, the Web of Science, ScienceDirect, and EBSCOhost as sources. At the first stage, we identified 96 papers potentially relevant for the topic of SASs design patterns after screening their title and abstracts. We intent here as design pattern any description of communicating entities that may be customized to solve a self-adaptive related design problem in a particular context. Through an analysis of the abstracts as well as the introduction sections of the papers, we identified the papers that are directly connected to design patterns for SASs. We explicitly exclude papers which present design patterns in a general context and rather included only design patterns in the context of (self-)adaptive systems as those systems have special characteristics (e.g., uncertainty in the system and the environment or incomplete defined set of requirements as the runtime environment is not known a priori). To ensure the applicability of design patterns in the context of adaptivity, we excluded all papers focusing on design patterns only or SASs in general from the review. Therefore, a remaining set of 14 papers was used as a basis for further review and search. Additionally, we performed a *"go backward"* and *"go forward"* search strategy as introduced by Webster & Watson [57] to identify further relevant papers. This strategy improves the coverage of the significant literature "by reviewing the citations of the articles identified [and finding] articles citing the key articles identified" [57, p. xvi] and helped to identify relevant works from close research domains. With this step of the literature research, we included ten additional papers. Consequently, we extracted design patterns from 24 papers.

The approach to develop our taxonomy on design patterns covers four steps. First, we compared the extracted papers using dimensions, such as whether the design patterns were based on SASs or other related concepts, whether the authors introduced a single pattern or composition of patterns, or the issue addressed by the patterns. Additionally, we extracted the relevant design patterns from the identified papers. Second, after identifying different dimensions for comparison and categorization of design patterns, we categorized the patterns along the dimensions. Last, we derived the dimensions for our taxonomy and the relevant properties for the analysis of the design patterns by iteratively refining its focus. In this step, we focus on design patterns for decentralized coordination that can be applied in IoT as IoT systems, by definition, are connected, distributed systems.

V. TAXONOMY ON DESIGN PATTERNS FOR DECENTRALIZED SELF-ADAPTIVE SYSTEMS

This section presents the taxonomy of design patterns for SASs. The literature research returned 24 relevant papers. We included papers that present the application of design patterns

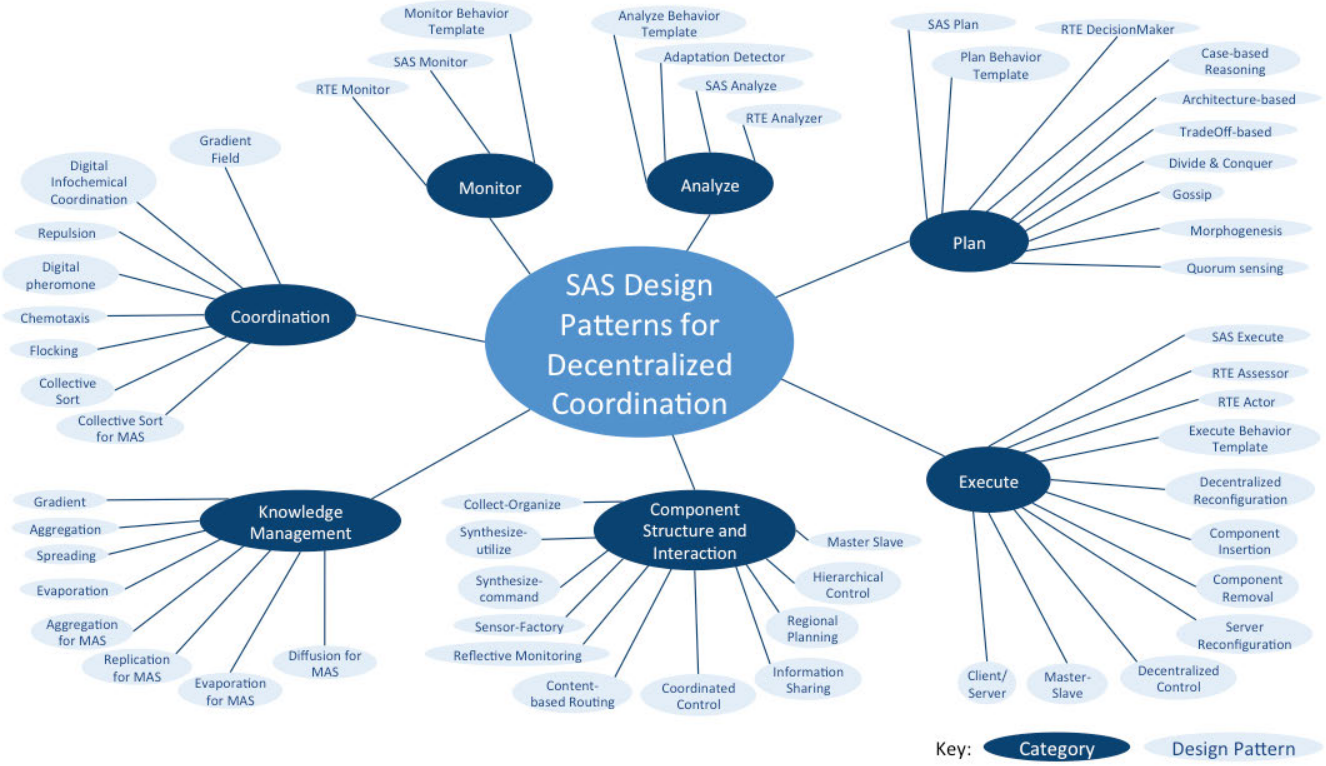

FIGURE 3. **The identified design patterns for self-adaptive systems.**

in a SAS. After carefully studying those papers, we added 55 design patterns out of 11 papers to the taxonomy.

The other papers provided related patterns that, for example, miss a systematic description or have only been applied in specific systems, and hence, we did not include them in the taxonomy. The design patterns included in the taxonomy are not only covering SASs, but also highly related domains, such as (self-organizing) multi-agent systems, self-organizing emergent systems, dynamically adaptive systems, self-adaptive real-time embedded (RTE) systems, and CPSs. Those patterns of related domains are included since they provide solutions that can be applied in IoT. The categorization of design patterns within the taxonomy is based on the functionality that patterns support. Throughout the development of the taxonomy, the scope was further narrowed down to cover SASs design patterns and to integrate the aspect of decentralized control which is highly important for IoT systems. Subsequently, based on the functionality of the managing system, the taxonomy (see Figure 3) includes the following categories: the monitor, analyze, plan, and execute categories provide the basic functionality of an adaptation logic as described by the MAPE-K loop [5] and similar feedback control structures [6]–[8], [58]. As the MAPE functionality is often implemented in dedicated modules, we focus on a separate analysis of the patterns. Additionally, orthogonal to those functionalities, we add the categories component structure and interaction, shared knowledge management, and coordination to reflect the distributed nature of IoT systems. Those aspects support the implementation of the MAPE functionalities. The following subsections provide the relevant design patterns categorized along with the mentioned MAPE functionality as well as the orthogonal aspects. For each category, we provide a short discussion of the relevant design patterns. Within those discussions, we describe overlapping descriptions, similarities, as well as differences in the design patterns.

A. CATEGORY 1: MONITOR

This category includes the design patterns associated with the monitoring step of the MAPE-K control loop [5].

The *SAS Monitor* design pattern aims to establish all the necessary components for the monitoring activity of the Monitor, Analyze, Plan, Execute, Knowledge (MAPE-K) control loop [59]. It decides which properties to monitor in order to detect possible violations that might trigger the adaptation process. The structural and behavioral views of the design pattern are illustrated with UML diagrams.

The *RTE Monitor* design pattern introduced by Said *et al.* [60] is used in the development of a self-adaptive RTE system to support the detection of an irregular RTE system status that resulted from the fluctuations in the internal and external context elements. The pattern enables the observation and monitoring of the system status and context

properties. Said *et al.* provide a structural and behavioral view of the pattern, including class and sequence diagrams.

Iglesia and Weyns [61] describe a *Monitor Behavior Template* that guides through the main monitoring activities: trigger the monitoring step, collect data through sensors, preprocess gathered data whenever possible, update working data, and inform the analyzing step. The *Monitor Behavior Template* is not specified using a pattern description; it is illustrated via multiple state chart diagrams.

Comparison: The two patterns *SAS Monitor* [59] and *RTE Monitor* [60] are almost identical: each is composed of 8 classes, 5 of them having identical names and identical semantics. The difference between the two patterns consists of the use of the *GoF Observer* design pattern. *SAS Monitor* monitors the significant context variables, while *RTE Monitor* explicitly addresses hardware- and software-specific sensors. *Monitor Behaviour Template* [61] models the behavioral aspects of the monitoring step. It is very similar to the behavior captured by the UML sequence diagrams of the *SAS Monitor* and *RTE Monitor* patterns. The main difference between this template and the previously mentioned two patterns consists in the addition of the data preprocessing activity. To summarize, these three patterns may be considered as *variants* (according to [2], [13]) of the same design pattern.

B. CATEGORY 2: ANALYZE

This category presents design patterns associated with analyzing. Those design patterns focus on evaluating the system state and its context to determine adaptation needs.

The *SAS Analyze* design pattern [59] is utilized to analyze previously collected data in order to identify if an adaptation is required. The pattern integrates a symptom-based analysis and aggregation of those symptoms to situations. Also, this pattern uses two observers to interact with the classes of the monitoring and planning steps of the MAPE-K loop. The structural and behavioral views of the design pattern are illustrated with UML class and sequence diagrams.

The *RTE Analyzer* design pattern [60] focuses on the system's stability by keeping the number of adaptation requests low. The pattern consists of the *Analyzer* for verifying system constraints and generating adaptation requests, while an *AdaptationRequest* captures the analysis results. The authors propose a class diagram for the structural view as well as a UML sequence diagram showing the behavior.

Analyze Behavior Template compares the required resources to the used ones [61]. The analysis component's behavior from the MAPE-K loop has three primary states (over satisfied, satisfied, and unsatisfied) that cover the essential steps of triggering the analysis, processing the analysis, and indicating related plan behavior(s). The *Analyze Behavior Template* is illustrated via multiple state chart diagrams showing possible actions that can be performed.

The *Adaptation Detector* design pattern [62] determines when a reconfiguration of the system is required by retrieving and analyzing the provided sensor data. A *HealthIndicator* class captures if an adaptation is required based on the Observer classes' feedback that interprets the sensor data. The pattern also includes a *Trigger* class to indicate the cause for adaptation. Ramirez specifies the design pattern through a UML class diagram for the structural view and a UML sequence diagram for the behavioral view.

Comparison: The two patterns *SAS Analyzer* [59] and *RTE Analyzer* [60] are both composed of 2 core classes: *Analyzer* and *AdaptationRequest*. However, the *SAS Analyzer* models the symptoms explicitly as a unit of analysis, provides a repository with symptom descriptions, and includes links to the adjacent MAPE-K steps using the Observer pattern. . *Analyze Behaviour Template* [61] indicates the behavioral aspects of the analyzing step. It is very similar to the behavior captured by the UML sequence diagrams of the *SAS Analyzer* and *RTE Analyzer* patterns. To summarize, these three patterns may be considered as variants for analysis. The fourth pattern, *Adaptation Detector* [62], includes classes specific to the monitoring (*Sensor* and *Threshold*) and to the planning (*Trigger*); hence, it has a broader scope than the other patterns.

C. CATEGORY 3: PLAN

Based on the analysis, if adaptation is required, the planning step includes a decision-making mechanism to plan adaptations. Accordingly, this category presents design patterns associated with the planning step of the MAPE-K loop [5].

The *SAS Plan* design pattern [59] specifies the necessary adaptation actions as well as the execution order by integrating a policy engine. In addition, this pattern uses two Observers to interact with the classes of the MAPE-K loop's monitoring and planning steps. The structural and behavioral views of the design pattern are illustrated using UML diagrams.

The *RTE DecisionMaker* [60] generates the adaptation decision that best fits the adaptation triggers and offers a component for handling the configurations of adaptable elements. The behavior is provided as a UML sequence diagram, while the structure is shown as a UML class diagram.

The *Plan Behavior Template* focuses on adaptations for adding resources or releasing resources [61]. The pattern is not specified using a pattern description format, rather it is illustrated via multiple state chart diagrams showing possible actions that can be performed on system states.

Ramirez [62] proposed several design patterns to support the planning of adaptation. In order to decide how to adapt the system, the *Case-based Reasoning* design pattern applies rule-based decision making, which can be applied for simple adaptations. The *Divide and Conquer* design pattern aims at systematically breaking down complex adaptation plans into reconfiguration plans that are easier to execute. This pattern is particularly useful if various reconfiguration plans have to be combined or if distributed components share dependencies along with a reconfiguration plan. Following an architecture-based approach, the *Architecture-based* design pattern supports the selection of reconfiguration plans when

they are estimated to change frequently. This pattern manages the architectural model's evolution from a current state to a possible future reconfigured target state. Last, the *TradeOff-based* design pattern chooses the plan which balances the best various objectives. As stated by the authors, the pattern is particularly useful if the reconfiguration requirements are satisfied by numerous reconfiguration plans and, hence, multiple dimensions need to be considered. All design patterns introduced are specified by class and sequence diagrams.

Fernandez-Marquez *et al.* [63] define natural-inspired design patterns. Three of them can be assigned to the plan category: *Gossip, Quorum sensing*, and *Morphogenesis*. *Gossip* aims at reaching a shared agreement about parameter values by spreading information to neighboring agents that aggregate this information with local information to reach an agreement. *Quorum sensing* estimates the density of agents which solely rely on local interactions as in cases where the number of agents collaborating on a specific task needs to be kept to a certain minimum, the density of agents is of high importance. Finally, *Morphogenesis* intends to choose different agent behaviors based on the agent's position. All design patterns are described with flow and interaction diagrams.

Comparison: We identified 10 design patterns for the planning step. *SAS Plan* [59] and *RTE DecisionMaker* [60] have a similar structure and behavior: they consist of a coordination class, a reasoning class, and an adaptation plan. However, both differ in decision making and apply a rule-based approach and a configuration-based approach, respectively. *Plan Behaviour Template* [61] indicates the behavioral aspects of the planning step, being very similar to the behavior captured by the UML sequence diagrams of *SAS Plan* and *RTE DecisionMaker*. *Case-based Reasoning* [62] can be considered as a more detailed variant of the *SAS Plan* pattern. The other six design patterns (from [62] and [63]) differ among them and have minimal commonalities with the four discussed patterns.

D. CATEGORY 4: EXECUTE

This category comprises design patterns associated with the execution step of the MAPE-K loop, responsible for implementing pre-defined adaptation actions. The *SAS Execute* design pattern [59] modifies system parameters or components through effectors and traces the performed adaptation actions. Its components are connected to the planning components using the Observer design pattern. The structural and behavioral views of the design pattern are illustrated with UML class and sequence diagrams.

Said *et al.* [60] present two patterns for executing of adaptations: *RTE Actor* and *RTE Assessor*. *RTE Actor* maps the described adaptation action from the received adaptation plan to components and controls adaptation. *RTE Assessor* supports cost-efficient adaptations by determining the necessary quality of service level through the evaluation and adjustment of the control loop based on the results of statistical analysis and estimations as well as parameter tuning. Both design patterns specify the structural view with a UML class diagram and the behavioral view with a sequence diagram.

Execute Behavior Template splits the execution of adaptation actions in pre-preparation (e.g., locking a resource), execution (e.g., performing adaptation), and post-execution (e.g., unlocking resources). The design pattern is not specified using a pattern description format, rather it is illustrated by exemplary state chart diagrams [61].

Ramirez [62] also define design patterns concerning the execution. The *Decentralized Reconfiguration* design pattern supports the component insertion and removal process of components from a decentralized architecture at runtime. The *Server Reconfiguration* design pattern provides a behavioral template for specifying a server-client architecture's reconfiguration without having to shut down the server. The *Component Removal* design pattern supports the safe removal process of a component at runtime whereas the *Component Insertion* design pattern is responsible for safely inserting and initializing components during runtime. Both design patterns require the available interfaces to enable the components to move to different behavioral states. Ramirez specifies the patterns using class and sequence diagrams.

Comparison: The *SAS Execute* [59] pattern is the simplest one being composed of an executor and effectors. *RTE Actor* [60] adds adaptation plans and adaptation actions. It can be combined with the *RTE Assessor* pattern [60], which aims at cost-efficient adaptation. *Execute Behaviors Template* [61] specifies a pre and a post activity for an adaptation. The design patterns proposed by Ramirez [62] add functionality for adding or removing architectural components at runtime.

E. CATEGORY 5: COMPONENT STRUCTURE AND INTERACTION

SASs are often systems-of-systems; hence, they are distributed. This is also related to decision making. As IoT systems are distributed by nature, decentralized decision making is an important aspect. Accordingly, this category introduces design patterns that deal with such decentralized decision settings and the corresponding interaction between different components. We focus here on distributing the adaptation logic functionality to the different components of the decentralized adaptation logic.

Based on discussions on a Dagstuhl seminar, Weyns *et al.* introduce five design patterns [64][2] for decentralized control focusing on the distribution of decision making components and its interaction. The *Information Sharing* and the *Coordinated Control* patterns support local decision making; however, in both patterns, data is exchanged for coordination in the monitoring functionality or all MAPE-K functionalities, respectively. Contrary, the other design patterns introduced by Weyns *et al.* apply a hybrid model with

[2]In this work, we integrated those patterns and refer to them as design patterns as those patterns also provide a template for designing the decision logic which is the relevant aspect for this work. However, it should be mentioned that in the original publication [64], the authors do not refer to them as design patterns.

partly centralized decision making. *Master-Slave* creates a hierarchical structure between interacting components where one master component controls the analyze and plan activities of the MAPE-K control loop. Monitoring and execution of adaptation decisions remain on the local instances. The *Regional Planning* pattern aggregates the planning functionality into regional planners. In this setting, each region has its own regional planner component. Furthermore, regional planners can collaborate across regions. Finally, *Hierarchical Control* separates concerns of MAPE-K control loops into different layers. While the top layer deals with the overarching adaptation goals of the system, the intermediate layers focus on the lower adaptation layers. Weyns *et al.* combine a standard pattern description format with structural diagrams for describing their design patterns.

Three of the twelve design patterns introduced by Ramirez [62] deal with component structure and interaction: *Sensor-Factory, Reflective Monitoring*, and *Content-based Routing*. *Sensor-Factory* is responsible for probing distributed components by deploying software sensors in a network. *Reflective Monitoring* supports mechanisms that enable the observation of a component's internal state. Furthermore, the proposed mechanisms allow altering the monitoring scheme dynamically. *Content-based Routing* aims at routing messages within a distributed monitoring setting according to the message's content. Ramirez specifies the design patterns by using structural UML diagrams and sequence diagrams.

Musil *et al.* propose three design patterns for self-adaptation, focusing on CPSs: *Synthesize-Utilize*, *Synthesize-Command*, and *Collect-Organize* [30]. Precisely, these multi-adaptation patterns capture information on the used adaptation mechanism, the interactions between those mechanisms across layers, and a definition of layers (i.e., physical, proxy, communication, service middleware, application, and social layer). *Synthesize-Utilize* provides an abstraction for including context information of physical resources. *Synthesize-Command* implements adaptation mechanisms based on the MAPE-K control loop for managing the physical resource(s). *Collect-Organize* consists of an adaptive algorithm, autonomous entities, and self-organizing mechanisms for efficient information sharing and local task coordination. Musil *et al.* describe their patterns using a pattern description and workflow diagrams.

Comparison: 5) Component Structure and Interaction: The five design patterns introduced by Weyns *et al.* [64] focus on the structure and interaction among several MAPE-K control loops, which may operate in parallel to manage self-adaptation. Two of them, i.e., *Coordinated Control* and *Information Sharing*, provide a decentralized approach, while the other three are hybrid. The patterns proposed by Musil *et al.* [30] aim to improve the utility of the services of distributed applications using MAPE-K control loops by exploiting context information. While the patterns proposed by Weyns *et al.* and Musil *et al.* have a global view on the self-adaptive mechanisms (i.e., MAPE-K control loop level), Ramirez *et al.* [62] focus on patterns which concern a single task of the MAPE-K functionalities, for example, *Sensor Factory* and *Reflective Monitoring* for the first step of the adaptation mechanism. Important in the context of component interaction, especially for IoT systems, are the aspects of security and data privacy. However, the studied patterns do not focus on those aspects. An extensive analysis of security and privacy aspects requires an implementation of the design patterns in real systems. Such an analysis is out of the scope of this paper and part of future work.

F. CATEGORY 6: COORDINATION

Whereas the previous category focuses on a distributed adaptation logic's structural aspects, this category captures distributed decision-making with global behavioral guarantees requires efficient coordination mechanisms. This is a procedural view of the distributed decision-making.

The *Collective Sort for MAS* design pattern proposed by Gardelli *et al.* [65] deals with the collective clustering of information to decrease the overhead in information repositories. The authors describe their pattern in a tabular description format. Another *Collective Sort* design pattern was proposed by Snyder *et al.* [66]. This pattern tackles the problem of scattered data or entities within a system by using heuristics for constantly collecting similar elements and form them into clusters. The authors describe their design pattern using a standard pattern description format.

The *Repulsion, Digital Pheromone, Chemotaxis*, and *Flocking* design patterns proposed by Fernandez-Marquez *et al.* [63] support coordination. *Repulsion* describes the basic process for motion coordination in a large agent-based system and enables agents to modify the position in response to changes in the environment. *Digital Pheromone* describes a swarm coordination mechanism that uses indirect communication based on gradients created by digital pheromones. *Chemotaxis* focuses on decentralized motion coordination and addresses the problem of discovering specific sources of events. *Flocking* deals with swarm formation by providing rules that specify how groups of agents move in the environment and also maintaining the connections between agents. The authors describe their design patterns with flow and interaction diagrams that show agent behavior and interactions.

The *Gradient Field* pattern supports coordination inspired by physical and biological processes [67]. Within a gradient field, different agents observe numerous gradient parts of neighboring locations and can move to those locations following the gradient field-specific wave format. The design pattern is outlined by textual specifications and in UML.

Kasinger *et al.* [68] present the *Digital Infochemical Coordination* pattern for the coordination of self-organizing emergent systems based on infochemical coordination. It is specified following a standard pattern description format and providing a conceptual model as a UML class diagram.

Comparison: The *Collective Sort* design pattern proposed by Snyder *et al.* [66] is a generalization of the *Collective*

Sort for MAS design pattern proposed by Gardelli *et al.* [65]. Snyder *et al.* extend the initial definition of the pattern (which assumes that active agents relocate inactive data items) also to cases where agents themselves may represent entities to be grouped or where different environmental abstractions may be used. Two of the design patterns proposed [63] have no similarities to other patterns (i.e., *Repulsion* and *Flocking*); *Digital Pheromone* may be compared to *Digital Infochemical Coordination* proposed by Kasinger *et al.* [68]. *Chemotaxis* [63] is an extension of the *Gradient* [63] pattern, and it can be compared to the *Gradient Fields* [67].

G. CATEGORY 7: KNOWLEDGE MANAGEMENT

As a third category orthogonal to the MAPE functionality, this category comprises design patterns that cover information and knowledge management to support the shared knowledge management in distributed systems.

Fernandez-Marquez *et al.* [63] propose the *Aggregation, Evaporation*, and *Spreading* design patterns. *Aggregation* describes the process of information fusion by locally applying a fusion operator, such as filtering or merging. *Evaporation* describes a mechanism to prioritize more recent and relevant information from older, potentially outdated information. *Spreading* deals with information diffusion within the system, focusing on direct communication between system parts in order to increase the global knowledge of all system parts. In addition to those basic design patterns, Fernandez-Marquez *et al.* propose the *Gradient* pattern as a composition of the *Aggregation* and *Spreading* patterns. The authors describe their design patterns using flow and interaction diagrams.

Gardelli *et al.* [65] propose the *Aggregation, Replication, Evaporation*, and *Diffusion*[3] design patterns. *Aggregation for MAS* aims at aggregating information in order to achieve coherent global views. *Replication for MAS* tries to lower access time to information and increase robustness by replicating. *Evaporation for MAS* addresses the problem of information flooding in large systems by environmental agents responsible for erasing obsolete information. Last, *Diffusion for MAS* aims at distributing information equally among all nodes within the system. Gardelli *et al.* describe their proposed design patterns in a tabular description format.

Comparison: The design patterns proposed by Fernandez-Marquez*et al.* [63], i.e., *Aggregation*, *Evaporation*, and *Spreading*, represent revised and enhanced versions of the previously defined *Aggregation for MAS*, *Evaporation for MAS*, and *Diffusion for MAS* patterns by Gardelli *et al.* [65]. Hence, the design patterns by Gardelli *et al.* may be seen as variants of those defined by Fernandez-Marquez *et al.* The *Replication for MAS* design pattern has not been revisited by Fernandez-Marquez*et al.* The *Blackboard* design pattern cannot be compared to the above mentioned patterns, having a different objective and coming from an architectural perspective rather than bio-inspired self-organizing systems.

[3] In order to distinguish those patterns from design patterns with the same name, we attach *"for MAS"* in the following.

H. FURTHER DESIGN PATTERNS

In the early stages of our research, we identified patterns that are not relevant for the taxonomy as they either do not support adaptivity in decentralized system settings or are not sufficiently described in a structured approach. For completeness, we mention those patterns in the following.

Taylor *et al.* proposed several architecture styles for runtime software adaptation [69]. The two design patterns *Reactive Stigmergy Service Components Ensemble Pattern* and *P2P AMs Service Components Ensemble Pattern* introduced by Puviani *et al.* [28] support settings with a large number of components interacting with each other in a frequently adjusting environment. Gomaa *et al.* outline patterns for service orchestration of service-oriented architectures [70]. The bio-inspired design patterns introduced by Babaoglu *et al.* describe techniques for information diffusion and handling in dynamic systems [71]. Those architectural styles are not included in the taxonomy as the authors only provide a high-level description.

Giese *et al.* present the *Blackboard* for data exchange in self-aware computing systems [31]. However, the authors do not formally describe the design pattern.

The *Market-Based Control* design pattern presented by De Wolf & Holvoet [67] as well as the *Specialization* design pattern introduced by Snyder *et al.* [66] deal with system optimization. Hence, they propose a system model rather than a design pattern for coping with specific design topics.

Last, the *Centralized Control* pattern by Gomaa & Hussein is not included in the taxonomy due to its focus on centralized control rather than decentralized coordination [72].

VI. APPLICATION OF THE DESIGN PATTERNS IN THE INTERNET OF THINGS

The field of IoT provides a large set of application domains, including smart home/smart buildings [48], smart transportation [73], smart city [74], IIoT [45], smart health [75], or smart energy management/smart grid [42]. Systems in those areas have in common that they can benefit from adaptivity to autonomously react to changes in their environment. Consequently, system development in those domains could highly benefit from the presented patterns as they provide a structured approach to include adaptivity. We outline that there are already some IoT examples using self-adaptivity and the MAPE-K loop: DeltaIoT [76] or Feed me [77]. However, their documentation does not indicate the application of self-adaptive design patterns, and there are no available tools for the automated detection of such patterns yet.

In the following, we present the application of the design patterns within an IIoT/Industry 4.0 use case. Industry 4.0 is a paradigm that fosters the collaboration of smart, autonomous machines (e.g., industrial robots or self-driving transport vehicles) with other machines and humans. The objective is an intelligent production process that offers high flexibility

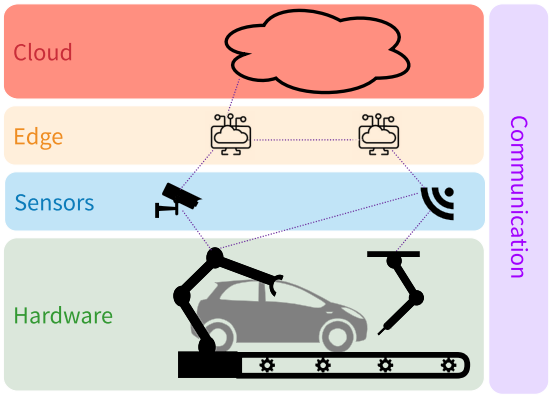

FIGURE 4. Multi-level IIoT system composed of hardware and sensors as well as Edge/Cloud levels for data analytics to control the lower local level.

and results in individualized products. In our IIoT use case, we focus on the application of smart production systems, that are connected and can automatically determine the next production steps as well as organize themselves in groups of connected Cyber-Physical Production Systems (CPPSs) accordingly. This can also include the organization of smart (internal and external) logistics, smart warehousing, as well as machine learning (e.g., for predictive maintenance). In [78], we present a self-adaptive exemplar for such coordinating robots: two robots divide a manufacturing task into subtasks and can re-organize the labor division in case of malfunctions of a robot. Here, we extend that scenario to a multi-layer system, including the groups of coordinating robots, Edge technology to intelligently analyze the robot coordination as well as Cloud technology for a production facility spanning view (see Figure 4). This follows the Cloud-in-the-Loop pattern from [55]. On both levels, Edge and Cloud, intelligent analysis based on machine learning and forecasting algorithms support the decision making. This enables self-adaptation of the production on several levels.

As we have a multi-level system approach, the monitoring part is not limited to a single production robot. Monitoring is necessary on all different levels of the system. Furthermore, for decentralized, local reasoning of the robots requires information dissemination of the instances within a layer. Hence, besides supporting the implementation of local monitoring procedures by the *SAS Monitor* design pattern [59], information dissemination across the layers as well as between instances within a layer can be supported by the knowledge management patterns, especially by *Aggregation, Evaporation*, and *Spreading* [63]. This supports new types of condition monitoring as demanded for IIoT applications [79]. The analysis — for example, to determine the quality of a product, identify the current product, or identify the wear of machine tools — happens on three levels: local (machine level), regional (Edge level), and global (Cloud level). Accordingly, this approach combines several component structure design patterns specified in [64]. On the one hand, the levels follow the *Hierarchical Control* design pattern. This support different types of analysis on the different levels and especially enables the intelligent, AI-based analysis of data [80], e.g., online failure prediction methods [81] for predictive maintenance. On the other hand, within a group of robots, the decision is made cooperatively; hence, the *Coordinated Control*, *Master-Slave*, or *Regional Planning* patterns might suit. The *Sensor-Factory* as well as the *Content-based Routing* design patterns [62] can structure and organize the information flow in distributed monitoring settings. In our IIoT use case, those patterns can be implemented using a stream processing systems like Apache Kafka for data collection as well as data dissemination, the ELK stack for data handling and analysis, and the MQTT protocol for pub/sub communication.

Processes have to guarantee efficient and non-conflicting analysis. Here, the *Adaptation Detector* design pattern [62] can support the identification of adaptation triggers. Further, the *SAS Analyzer* design pattern [59] supports the implementation of a multi-level analysis approach by the possibility of clearly assigning the relevant functionality to the corresponding levels as well as efficient aggregation mechanisms to derive situations from (potentially locally derived) symptoms, e.g., using the approach for situation-awareness from [82].

Analogously, planning should take place at all different levels, i.e., the smart factory, the CPPSs as well as company-wide, hence, between different factories [83]. Planning in our scenario mainly involves identifying the next production steps based on the specification of the current product. As mentioned above, the objective is high flexibility to enable individualized products and support multiple objectives simultaneously [84]. Further, it includes the coordination of the required production steps between the machines. On the local level, *Case-based Reasoning* [62] supported by adaptation rules might be applied. Further, the *Gossip* pattern [63] aims at reaching a shared agreement about parameter values; hence, can support the decision making in a group of robots. For multi-level decision making, the use of the *Divide and Conquer* pattern [62] can divide the responsibilities and different scopes of decision making across the levels, primarily as it supports the combination of several reconfiguration plans. Additionally, the *TradeOff-based* pattern [62] chooses the plan which balances several objectives; hence, it helps to balance trade-off decisions across the levels.

Using the *SAS Execute* design pattern [59] enables the description and implementation of clear interfaces and process workflows for adaptation. This can support the definition of required workflows for the smart factory, i.e., intelligent production processes supported by CPPSs and the required integration of employees with the smart machines [85]. Hence, those adaptation patterns complement the definition of production workflows. The process of adaptation can be supported by the *Execute Behavior Template* which splits the execution of adaptation actions in pre-preparation, execution, and post-execution [61]. This might be especially interesting in this context as those three phases can be distributed across

the levels. The *RTE Actor* pattern [60] maps the described adaptation actions in the adaptation plan to components and controls their adaptation; hence, this can support the central decision making on higher layers and efficient dissemination and execution of the high-level adaptation plans.

Last, several design patterns support the coordination within the different levels: The Digital Pheromones pattern [63] enables the coordination of the production robots, the *Collective Sort for MAS* pattern [65] provides clustering of information on the Edge and Cloud levels, and the *Collective Sort* pattern [66] offers plausibility checks of data. Those patterns can support the application of evaluation models for IIoT as proposed in [86].

In summary, the IIoT use case shows the potential of the design patterns to enable adaptivity in IoT systems on several levels and targets all of the required MAPE-K functionality for adaptivity. A measurement of the effects of the design patterns in a productive system is part of future work.

VII. THREATS TO VALIDITY

To provide an objective and comprehensive overview of the state of the art, we applied a systematic literature review [57] combined with forward and backward search. We analyzed each of the identified papers in detail and extracted the information about the design patterns by using systematic information representations. Nonetheless, the results may be biased due to the manual steps of our methodology. For example, as not all design patterns are described in a systematic approach, the information's extraction might not always be entirely consistent. However, we try to minimize the risk of such effects by carefully discussing doubtful aspects between the authors. Additionally, the set of keywords used in the literature review focuses on the area of SASs. Hence, it might be possible that we miss design patterns from other fields that could be applied in our context.

In this paper, we focus on SASs that have similar characteristics to IoT systems. Still, this can result in not including works that do not confirm the search terms. To overcome this, we apply forward/backward search, to also include papers that do not comply with the search terms.

We are aware of the fact that the discussion of the design patterns in Section V does not follow a systematic approach. On the one hand, this is barely feasible due to the different nature of the patterns as well as the different degrees structured information provided by the authors. On the other hand, we reduced the risk of misinterpretation by careful consideration and the authors' experience in the areas of design patterns and SASs. We are aware that patterns must be applied in the design of a system to see how they can solve design issues. However, design patterns represent design solutions. Hence, they may have potentially an infinite number of implementations, called variants in the software engineering literature. Still, we plan as future work to investigate some of the design patterns in real systems to systematically measure their impact. As mentioned at the beginning of Section VI, there are already some IoT systems (e.g., DeltaIoT, Feed me) that explicitly rely on MAPE-K and maybe also on some of the design patterns presented in this paper. Another future activity may concern reengineering already available IoT systems using design patterns and making a comparison.

We focus on the MAPE-K system model [5] for structuring the taxonomy. This seems to be a limitation, as this is a specific model. However, other authors propose similar feedback structures, such as, the sense-plan-act control [6], the autonomic control loop [7], the observer/controller architecture [8], or based on MIAC/MRAC controllers [58]. All of those contain the same basic functionality of data collection, reasoning, and adaptation. Further, the addition of three orthogonal dimensions—component structure and interaction, shared knowledge management, and coordination—reflect the distributed nature of IoT systems as outlined in many papers such as [87]–[89] and, hence, improves the compliance to the target domain. In addition, in this paper we focused on design patterns; hence other architectural or design mechanisms (e.g., algorithms) have not been discussed.

Another problem might arise due to the fact that different activities are associated with each MAPE activity. However, one has to mention that still some aspects are always present for a specific MAPE activity, as described in the following. The design patterns capture and support those activities without limiting their applicability through a detailed, narrow implementation. For example, the SAS Analyze pattern analyzes the collected data to identify possible issues, i.e., ''symptoms''; however, it does not mention anything about how the analysis is performed: there can be applied various analysis mechanisms ranging from simple comparisons with a reference value or advanced mechanisms based on machine learning. This is valid for several other design patterns; it is the nature of a design pattern to provide not an implementation solution in a specific programming language but a generic applicable concept.

Applying the MAPE model provides a split of the pattern's functionality so that developers can easily identify possible patterns for a specific MAPE functionality. Hence, it is a simplified approach to clustering the patterns. However, as the orthogonal categories of the taxonomy — component structure and interaction shared knowledge management, and coordination — already show the discussion of design patterns might also require a view that spans across the single functions. Further, it might be possible the there are dependencies between patterns, for example, that an analysis pattern relies on a specific type of data created by a monitoring pattern. Hence, the horizontal comparison of patterns might be reduced by our approach chosen for deriving the taxonomy. As we want to present in this paper an overview of patterns, we think that the construction method of the taxonomy is suitable for our purpose. As already mentioned, an in-depth analysis of the patterns — including a study of their dependencies and their applicability of various use cases— is part of our future work. However, a comprehensive

comparison of all patterns in various use cases seems hardly feasible to do the high implementation effort.

In this paper, we focus on the application of design patterns to simplify the development of the managing subsystem's functionality. Of course, further relevant aspects for IoT systems exist that we excluded, especially the security or data privacy in those systems. The reason for the exclusion of security and privacy aspects is that these two aspects pose entirely new challenges to IoT developers and are often in contrast to commonly applied design patterns. Design decisions for IoT systems take into account the decentralized nature of IoT systems and that they typically consist of battery-powered and resource-constrained devices. Accordingly, developers focus on optimizing performance, which is, e.g., clearly evident in most commonly used IoT protocols such as MQTT. These protocols are usually based on the so-called Pub/Sub architecture, which aims to increase IoT devices' performance by implementing communication between IoT devices using additional devices, so-called brokers, which handle the main communication burden. Thereby security mechanisms are almost wholly ignored, as it is the case with MQTT [90]. This is especially crucial, as IoT systems rely on sensors that potentially gather sensitive data. Since conventional IoT design decisions were made with regard to performance, but not security, and IoT systems also place a whole range of new demands on security and privacy mechanisms, security, and privacy, which are not regarded by well-established and state-of-the-art patterns [91]. Further, security or data privacy is often implementation-specific aspects, whereas design patterns incorporate design-specific aspects. Still, for future work, an interesting research question would be to investigate the influence of security — for example, based on existing works related to security in IoT (such as [92]–[95]) — on the analyzed design patterns.

VIII. CONCLUSION AND FUTURE WORK

IoT systems are strongly distributed systems with heterogeneous hardware and software resources. Furthermore, those systems interact with users or backend systems in dynamic environments. Adaptivity can help to overcome those challenges. With this work, we contribute to this topic by providing an overview of design patterns in SASs. We outline that there are at least two main advantages of applying design patterns in the engineering of any system: (1) quality achievement, because design patterns capture efficient solutions to recurring problems in specific contexts, and (2) maintenance support, because design patterns implicitly document the solution applied indicating not only *how* it has been designed, but also *why* it has been designed due to the semantic behind a pattern, i.e., its motivation.

Based on a systematic literature review, we identified 24 relevant papers as a base for taxonomy on design patterns for SASs. This taxonomy is composed of seven categories (monitor, analyze, plan, execute, component structure & interaction, knowledge management, and coordination) with a total of 55 design patterns. We exemplify the applicability of those patterns in an IIoT use case.

With this work, we provide qualitative analysis for integrating design patterns that support adaptivity in IoT systems.

An open issue in SAS concern the context and system models representation and management as the knowledge in the MAPE-K loop. TThese models may grow and become bigger and bigger based on the monitoring context and adaptation strategies' needs. In addition, these models also become very complex and hard to exploit at runtime, as well as hard to maintain. Alternative approaches to models at runtime are emerging, e.g., based on big data technologies and analytics [96].

As future work, we plan to provide a thorough discussion of the applicability on different IoT system domains by comparing the implementations of IoT systems having integrated those patterns to systems without design patterns for measuring the effectiveness of the design patterns. Moreover, a future task will concern the privacy and security issues [97]–[99] addressed in SAS, which may be adapted and applied in IoT though self-protection mechanisms [100].

ACKNOWLEDGMENT

The authors would like to thank Veronika Lesch and Martin Pfannemüller for their valuable feedback.

REFERENCES

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA, USA: Addison-Wesley, 1994.

[2] F. Arcelli Fontana, F. Perin, C. Raibulet, and S. Ravani, "Design pattern detection in java systems: A dynamic analysis based approach," in *Proc. ENASE*, 2010, pp. 163–179.

[3] B. H. Cheng, R. De Lemos, H. Giese, P. Inverardi, and J. Magee, "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*. Berlin, Germany: Springer, 2009, pp. 1–26.

[4] C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems," *Pervas. Mobile Comput.*, vol. 17, pp. 184–206, Feb. 2015.

[5] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[6] J. Kramer and J. Magee, "Self-managed systems: An architectural challenge," in *Proc. Future Softw. Eng. (FOSE)*, May 2007, pp. 259–268.

[7] S. Dobson, S. Denazis, A. Fernández, and D. Gaïti, "A survey of autonomic communications," *ACM Trans. Auto. Adapt. Syst.*, vol. 1, no. 2, pp. 223–259, Dec. 2006.

[8] S. Tomforde, H. Prothmann, J. Branke, J. Hähner, M. Mnif, C. Müller-Schloer, U. Richter, and H. Schmeck, "Observation and control of organic systems," in *Organic Computing–A Paradigm Shift for Complex Systems*. Basel, Switzerland: Springer, 2011, pp. 325–338.

[9] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, and H. Sundmaeker, "Internet of things strategic research roadmap," IERC, Oslo, Norway, Tech. Rep., 2011.

[10] B. Eberhardinger, I. Gerostathopoulos, C. Krupitzer, P. Lewis, and C. Raibulet, "EMSAC-SeAC 2019: Evaluations and measurements in self-aware computing systems workshop and the workshop on self-aware computing," in *Proc. IEEE 4th Int. Workshops Found. Appl. Self Syst.*, Jun. 2019, pp. 21–22.

[11] C. Krupitzer, B. Eberhardinger, I. Gerostathopoulos, and C. Raibulet, "Introduction to the special issue 'Applications in self-aware computing systems and their evaluation,'" *Computers*, vol. 9, no. 1, p. 22, 2020, doi: 10.3390/computers9010022.

[12] F. Arcelli Fontana, S. Maggioni, and C. Raibulet, "Understanding the relevance of micro-structures for design patterns detection," *J. Syst. Softw.*, vol. 84, no. 12, pp. 2334–2347, Dec. 2011.

[13] F. A. Fontana, S. Maggioni, and C. Raibulet, "Design patterns: A survey on their micro-structures: Design patterns: A survey on their micro-structures," *J. Softw., Evol. Process*, vol. 25, no. 1, pp. 27–52, Jan. 2013.

[14] E. Kaddoum, C. Raibulet, J.-P. Georgé, G. Picard, and M.-P. Gleizes, "Criteria for the evaluation of self-* systems," in *Proc. ICSE Workshop Softw. Eng. Adapt. Self-Managing Syst. (SEAMS)*, 2010, pp. 29–38.

[15] C. Raibulet and F. A. Fontana, "Evaluation of self-adaptive systems: A women perspective," in *Proc. ECSA*, 2017, pp. 23–30.

[16] C. Krupitzer, F. M. Roth, C. Becker, M. Weckesser, M. Lochau, and A. Schurr, "FESAS IDE: An integrated development environment for autonomic computing," in *Proc. IEEE Int. Conf. Autonomic Comput. (ICAC)*, Jul. 2016, pp. 15–24.

[17] D. Weyns, "Engineering self-adaptive software systems–An organized tour," in *Proc. IEEE 3rd Int. Workshops Found. Appl. Self* Syst. (FAS*W)*, Sep. 2018, pp. 1–2, doi: 10.1109/FAS-W.2018.00012.

[18] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, Oct. 2004, doi: 10.1109/MC.2004.175.

[19] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng, "Composing adaptive software," *Computer*, vol. 37, no. 7, pp. 56–64, Jul. 2004, doi: 10.1109/MC.2004.48.

[20] G. Bastide, A. Seriai, and M. Oussalah, "Software component re-engineering for their runtime structural adaptation," in *Proc. 31st Annu. Int. Comput. Softw. Appl. Conf.*, vol. 1, Jul. 2007, pp. 109–114, doi: 10.1109/COMPSAC.2007.192.

[21] I. Gorton, Y. Liu, and N. Trivedi, "An extensible and lightweight architecture for adaptive server applications," *Softw., Pract. Exper.*, vol. 38, no. 8, pp. 853–883, Jul. 2008, doi: 10.1002/spe.857.

[22] J. He, T. Gao, W. Hao, I.-L. Yen, and F. Bastani, "A flexible content adaptation system using a rule-based approach," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 1, pp. 127–140, Jan. 2007, doi: 10.1109/TKDE.2007.250590.

[23] O. Choi and Y. Yoon, "A meta data model of context information for dynamic service adaptation on user centric environment," in *Proc. Int. Conf. Multimedia Ubiquitous Eng. (MUE)*, 2007, pp. 108–113, doi: 10.1109/MUE.2007.22.

[24] C. Raibulet, "Hints on quality evaluation of self-systems," in *Proc. IEEE 8th Int. Conf. Self-Adapt. Self-Organizing Syst.*, Sep. 2014, pp. 185–186, doi: 10.1109/SASO.2014.36.

[25] C. Raibulet, F. Arcelli Fontana, and S. Carettoni, "A preliminary analysis of self-adaptive systems according to different issues," *Softw. Qual. J.*, vol. 28, no. 3, pp. 1213–1243, Sep. 2020, doi: 10.1007/s11219-020-09502-5.

[26] S. Tomforde and C. Müller-Schloer, "Incremental design of adaptive systems," *J. Ambient Intell. Smart Environments*, vol. 6, no. 2, pp. 179–198, 2014.

[27] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA, USA: Addison-Wesley, 1994.

[28] M. Puviani, G. Cabri, and F. Zambonelli, "A taxonomy of architectural patterns for self-adaptive systems," in *Proc. Int. C* Conf. Comput. Sci. Softw. Eng. (C3S2E)*, 2013, pp. 77–85.

[29] J. Juziuk, D. Weyns, and T. Holvoet, "Design patterns for multi-agent systems: A systematic literature review," in *Agent-Oriented Software Engineering*. Berlin, Germany: Springer, 2014, pp. 79–99.

[30] A. Musil, J. Musil, D. Weyns, T. Bures, H. Muccini, and M. Sharaf, "Patterns for self-adaptation in cyber-physical systems," in *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*. Basel, Switzerland: Springer, 2017, pp. 331–368.

[31] H. Giese, T. Vogel, A. Diaconescu, S. Götz, and K. L. Bellman, "Generic architectures for individual self-aware computing systems," in *Self-Aware Computing Systems*, S. Kounev, J. O. Kephart, A. Milenkoski, and X. Zhu, Eds. Cham, Switzerland: Springer, 2017, 149–189, doi: 10.1007/978-3-319-47474-8_6.

[32] A. J. Ramirez and B. H. C. Cheng, "Design patterns for developing dynamically adaptive systems," in *Proc. ICSE Workshop Softw. Eng. Adapt. Self-Manag. Syst. (SEAMS)*, 2010, pp. 49–58.

[33] D. Weyns, S. Malek, and J. Andersson, "On decentralized self-adaptation: Lessons from the trenches and challenges for the future," in *Proc. SEAMS*, 2010, pp. 84–93.

[34] E. Jung, I. Cho, and S. M. Kang, "An agent modeling for overcoming the heterogeneity in the IoT with design patterns," in *Proc. MUSIC*, 2013, pp. 69–74.

[35] S. Qanbari, S. Pezeshki, R. Raisi, S. Mahdizadeh, R. Rahimzadeh, N. Behinaein, F. Mahmoudi, S. Ayoubzadeh, P. Fazlali, K. Roshani, A. Yaghini, M. Amiri, A. Farivarmoheb, A. Zamani, and S. Dustdar, "IoT design patterns: Computational constructs to design, build and engineer edge applications," in *Proc. IEEE 1st Int. Conf. Internet–Things Design Implement. (IoTDI)*, Apr. 2016, pp. 277–282.

[36] M. Vega-Barbas, I. Pau, J. C. Augusto, and F. Seoane, "Interaction patterns for smart spaces: A confident interaction design solution for pervasive sensitive IoT services," *IEEE Access*, vol. 6, pp. 1126–1136, 2018.

[37] V. Sithole and L. Marchall, "Attributes extraction for fine-grained differentiation of the Internet of Things patterns," in *Proc. South Afr. Inst. Comput. Scientists Inf. Technologists*, 2019, p. 9.

[38] L. F. Rahman, T. Ozcelebi, and J. J. Lukkien, "Designing IoT systems: Patterns and managerial conflicts," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2019, pp. 542–548.

[39] L. Reinfurt, U. Breitenbücher, M. Falkenthal, F. Leymann, and A. Riegg, "Internet of things patterns for communication and management," in *Transactions on Pattern Languages of Programming*, vol. 4. Berlin, Germany: Springer, 2019, pp. 139–182.

[40] L. Reinfurt, U. Breitenbücher, M. Falkenthal, P. Fremantle, and F. Leymann, "Internet of Things security patterns," in *Proc. PLoP*, 2017, p. 20.

[41] L. Reinfurt, U. Breitenbücher, M. Falkenthal, F. Leymann, and A. Riegg, "Internet of Things patterns for devices," in *Proc. PATTERNS*, 2017.

[42] L. Cruz and R. Abreu, "Catalog of energy patterns for mobile applications," *Empirical Softw. Eng.*, vol. 24, no. 4, pp. 2209–2235, Aug. 2019.

[43] S. Pape and K. Rannenberg, "Applying privacy patterns to the Internet of Things' (IoT) architecture," *Mobile Netw. Appl.*, vol. 24, no. 3, pp. 925–933, 2019.

[44] M. Weyrich and C. Ebert, "Reference architectures for the Internet of things," *IEEE Softw.*, vol. 33, no. 1, pp. 112–116, Jan. 2016.

[45] S. R. Bader, M. Maleshkova, and S. Lohmann, "Structuring reference architectures for the industrial Internet of things," *Future Internet*, vol. 11, no. 7, p. 151, Jul. 2019.

[46] P. Sethi and S. R. Sarangi, "Structuring reference architectures for the industrial Internet of Things," *J. Electr. Comput. Eng.*, vol. 2017, p. 25, 2019.

[47] B. Di Martino, M. Rak, M. Ficco, A. Esposito, S. A. Maisto, and S. Nacchia, "Internet of things reference architectures, security and interoperability: A survey," *Internet Things*, vols. 1–2, pp. 99–112, Sep. 2018.

[48] D. Mocrii, Y. Chen, and P. Musilek, "IoT-based smart homes: A review of system architecture, software, communications, privacy and security," *Internet Things*, vols. 1–2, pp. 81–98, Sep. 2018.

[49] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, opportunities, and directions," *IEEE Trans. Ind. Informat.*, vol. 14, no. 11, pp. 4724–4734, Nov. 2018.

[50] L. Da Xu, W. He, and S. Li, "Internet of Things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[51] Y. Liao, E. de Freitas Rocha Loures, and F. Deschamps, "Industrial Internet of Things: A systematic literature review and insights," *IEEE Internet Things J.*, vol. 5, no. 6, pp. 4515–4525, Dec. 2018.

[52] B. Caesar, F. Grigoleit, and S. Unverdorben, "(Self-)adaptiveness for manufacturing systems: Challenges and approaches," *SICS Softw.-Intensive Cyber-Phys. Syst.*, vol. 34, no. 4, pp. 191–200, Dec. 2019.

[53] S. Cha, B. Vogel-Heuser, and J. Fischer, "Analysis of metamodels for model-based production automation system engineering," *IET Collaborative Intell. Manuf.*, vol. 2, no. 2, pp. 45–55, Jun. 2020.

[54] A. Theorin, K. Bengtsson, J. Provost, M. Lieder, C. Johnsson, T. Lundholm, and B. Lennartson, "An event-driven manufacturing information system architecture for industry 4.0," *Int. J. Prod. Res.*, vol. 55, no. 5, pp. 1297–1311, Mar. 2017, doi: 10.1080/00207543.2016.1201604.

[55] G. Bloom, B. Alsulami, E. Nwafor, and I. C. Bertolotti, "Design patterns for the industrial Internet of things," in *Proc. 14th IEEE Int. Workshop Factory Commun. Syst. (WFCS)*, Jun. 2018, pp. 1–10.

[56] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *J. Syst. Softw.*, vol. 80, no. 4, pp. 571–583, Apr. 2007.

[57] J. Webster and R. T. Watson, "Analyzing the past to prepare for the future: Writing a literature review," *MIS Quart.*, vol. 26 no. 2, pp. 13–23, 2002.

[58] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, "Engineering self-adaptive systems through feedback loops," in *Software Engineering for Self-Adaptive Systems*. Berlin, Germany: Springer, 2009, pp. 48–70.
[59] Y. Abuseta and K. Swesi, "Design patterns for self adaptive systems engineering," 2015, *arXiv:1508.01330*. [Online]. Available: http://arxiv.org/abs/1508.01330
[60] M. Ben Said, Y. Hadj Kacem, M. Kerboeuf, N. Ben Amor, and M. Abid, "Design patterns for self-adaptive RTE systems specification," *Int. J. Reconfigurable Comput.*, vol. 2014, pp. 1–21, 2014.
[61] D. G. D. L. Iglesia and D. Weyns, "MAPE-K formal templates to rigorously design behaviors for self-adaptive systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 10, no. 3, p. 15, 2015.
[62] A. J. Ramirez, *Design Patterns for Developing Dynamically Adaptive Systems*. East Lansing, MI, USA: Michigan State Univ., 2008.
[63] J. L. Fernandez-Marquez, G. Di Marzo Serugendo, S. Montagna, M. Viroli, and J. L. Arcos, "Description and composition of bio-inspired design patterns: A complete overview," *Natural Comput.*, vol. 12, no. 1, pp. 43–67, Mar. 2013.
[64] D. Weyns, B. Schmerl, V. Grassi, S. Malek, and R. Mirandola, "On patterns for decentralized control in self-adaptive systems," in *Software Engineering for Self-Adaptive Systems II*. Berlin, Germany: Springer, 2013, pp. 76–107.
[65] L. Gardelli, M. Viroli, and A. Omicini, "Design patterns for self-organising systems," in *Multi-Agent Systems and Applications V*. Berlin, Germany: Springer, 2007, pp. 123–132.
[66] P. L. Snyder, G. Valetto, J. L. Fernandez-Marquez, and G. D. M. Serugendo, "Augmenting the repertoire of design patterns for self-organized software by reverse engineering a bio-inspired P2P system," in *Proc. IEEE 6th Int. Conf. Self-Adapt. Self-Organizing Syst.*, Sep. 2012, pp. 199–204.
[67] T. De Wolf and T. Holvoet, "Design patterns for decentralised coordination in self-organising emergent systems," in *Engineering Self-Organising Systems*. Berlin, Germany: Springer, 2007, pp. 28–49.
[68] H. Kasinger, B. Bauer, and J. Denzinger, "Design pattern for self-organizing emergent systems based on digital infochemicals," in *Proc. 6th IEEE Conf. Workshops Eng. Autonomic Auto. Syst.*, Apr. 2009, pp. 45–55.
[69] R. N. Taylor, N. Medvidovic, and P. Oreizy, "Architectural styles for runtime software adaptation," in *Proc. Joint Work. IEEE/IFIP Conf. Softw. Archit. Eur. Conf. Softw. Archit.*, Sep. 2009, pp. 171–180.
[70] H. Gomaa, K. Hashimoto, M. Kim, S. Malek, and D. A. Menascé, "Software adaptation patterns for service-oriented architectures," in *Proc. ACM Symp. Appl. Comput. (SAC)*, 2010, pp. 462–469.
[71] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, and A. Montresor, "Design patterns from biology for distributed computing," *ACM Trans. Auton. Adapt. Syst.*, vol. 1, no. 1, pp. 26–66, 2006.
[72] H. Gomaa and M. Hussein, "Software reconfiguration patterns for dynamic evolution of software architectures," in *Proc. 4th Work. IEEE/IFIP Conf. Softw. Archit. (WICSA)*, Jun. 2004, pp. 79–88.
[73] C. Krupitzer, M. Segata, M. Breitbach, S. El-Tawab, S. Tomforde, and C. Becker, "Towards infrastructure-aided self-organized hybrid platooning," in *Proc. IEEE Global Conf. Internet Things (GCIoT)*, Dec. 2018, pp. 1–6.
[74] R. Lea and M. Blackstock, "City hub: A cloud-based IoT platform for smart cities," in *Proc. IEEE 6th Int. Conf. Cloud Comput. Technol. Sci.*, Dec. 2014, pp. 799–804.
[75] C. Krupitzer, T. Sztyler, J. Edinger, M. Breitbach, H. Stuckenschmidt, and C. Becker, "Hips do lie! A position-aware mobile fall detection system," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. (PerCom)*, Mar. 2018, pp. 95–104.
[76] M. U. Iftikhar, G. S. Ramachandran, P. Bollansée, D. Weyns, and D. Hughes, "Deltaiot: A real world exemplar for self-adaptive Internet of Things (artifact)," *DARTS*, vol. 3, no. 1, p. 4, 2017.
[77] A. Bennaceur, C. McCormick, J. García-Galán, C. Perera, A. Smith, A. Zisman, and B. Nuseibeh, "Feed me, feed me: An exemplar for engineering adaptive software," in *Proc. IEEE/ACM 11th Int. Symp. Softw. Eng. Adapt. Self-Manag. Syst. (SEAMS)*, May 2016, pp. 89–95.
[78] C. Krupitzer, G. Drechsel, D. Mateja, A. Pollklasener, F. Schrage, T. Sturm, A. Tomasovic, and C. Becker, "Using spreadsheet-defined rules for reasoning in self-adaptive systems," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2018, pp. 462–467.
[79] A. Maier, S. Schriegel, and O. Niggemann, *Big Data and Machine Learning for the Smart Factory—Solutions for Condition Monitoring, Diagnosis and Optimization*. Cham, Switzerland: Springer, 2017, pp. 473–485.
[80] S. Goetz, G. Keitzel, and F. Klocke, *Going Smart—CPPS for Digital Production*. Cham, Switzerland: Springer, 2017, pp. 401–422.
[81] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Comput. Surv.*, vol. 42, no. 3, pp. 1–42, Mar. 2010, doi: 10.1145/1670679.1670680.
[82] E. M. Fredericks, I. Gerostathopoulos, C. Krupitzer, and T. Vogel, "Planning as optimization: Dynamically discovering optimal configurations for runtime situations," in *Proc. IEEE 13th Int. Conf. Self-Adapt. Self-Organizing Syst. (SASO)*, Jun. 2019.
[83] D. Zuehlke, "Smartfactory—From vision to reality in factory technologies," *IFAC Proc. Volumes*, vol. 41, no. 2, pp. 14101–14108, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667016412565
[84] M. Saggiomo, Y.-S. Gloy, and T. Gries, *Applying Multi-objective Optimization Algorithms to a Weaving Machine as Cyber-Physical Production System*. Cham, Switzerland: Springer, 2017, pp. 505–517.
[85] M. Palviainen, J. Mäntyjärvi, J. Ronkainen, and M. Tuomikoski, *Towards User-Driven Cyber-Physical Systems–Strategies to Support User Intervention in Provisioning of Information and Capabilities of Cyber-Physical Systems*. Cham, Switzerland: Springer, 2017, pp. 575–593.
[86] M. Weyrich, M. Klein, J.-P. Schmidt, N. Jazdi, K. D. Bettenhausen, F. Buschmann, C. Rubner, M. Pirker, and K. Wurm, *Evaluation Model for Assessment of Cyber-Physical Production Systems*. Cham, Switzerland: Springer, 2017, pp. 169–199.
[87] M. Weisbach, N. Taing, M. Wutzler, T. Springer, A. Schill, and S. Clarke, "Decentralized coordination of dynamic software updates in the Internet of things," in *Proc. IEEE 3rd World Forum Internet Things (WF-IoT)*, Reston, VA, USA, Dec. 2016, pp. 171–176, doi: 10.1109/WF-IoT.2016.7845450.
[88] M. WeiBbach, P. Chrszon, T. Springer, and A. Schill, "Decentrally coordinated execution of adaptations in distributed self-adaptive software systems," in *Proc. IEEE 11th Int. Conf. Self-Adapt. Self-Organizing Syst. (SASO)*, Tucson, AZ, USA, Sep. 2017, pp. 111–120, doi: 10.1109/saso.2017.20.
[89] M. Weißbach and T. Springer, "Coordinated execution of adaptation operations in distributed role-based software systems," in *Proc. Symp. Appl. Comput.*, Marrakech, Morocco, A. Seffah, B. Penzenstadler, C. Alves, and X. Peng, Eds. New York, NY, USA: ACM, Feb. 2017, pp. 45–50, doi: 10.1145/3019612.3019624.
[90] T. Prantl, P. Ten, L. Iffländer, A. Dmitrenko, S. Kounev, and C. Krupitzer, "Evaluating the performance of a state-of-the-art group-oriented encryption scheme for dynamic groups in an IoT scenario," in *Proc. IEEE Symp. Model., Anal., Simul. Comput. Telecommun. Syst. (MASCOTS)*, 2020.
[91] P. Thomas, "SIMPL: Secure IoT management platform," in *Proc. ITSec*, 2020, pp. 1–2.
[92] C. Choi and J. Choi, "Ontology-based security context reasoning for power IoT-cloud security service," *IEEE Access*, vol. 7, pp. 110510–110517, 2019.
[93] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A survey on IoT security: Application areas, security threats, and solution architectures," *IEEE Access*, vol. 7, pp. 82721–82743, 2019.
[94] S. Zahra, M. Alam, Q. Javaid, A. Wahid, N. Javaid, S. U. R. Malik, and M. Khurram Khan, "Fog computing over IoT: A secure deployment and formal verification," *IEEE Access*, vol. 5, pp. 27132–27144, 2017.
[95] G. George and S. M. Thampi, "A graph-based security framework for securing industrial IoT networks from vulnerability exploitations," *IEEE Access*, vol. 6, pp. 43586–43601, 2018.
[96] S. Schmid, I. Gerostathopoulos, C. Prehofer, and T. Bures, "Self-adaptation based on big data analytics: A model problem and tool," in *Proc. IEEE/ACM 12th Int. Symp. Softw. Eng. Adapt. Self-Manag. Syst. (SEAMS)*, Buenos Aires, Argentina, May 2017, pp. 102–108, doi: 10.1109/SEAMS.2017.20.
[97] A. Hassan, R. Hamza, H. Yan, and P. Li, "An efficient outsourced privacy preserving machine learning scheme with public verifiability," *IEEE Access*, vol. 7, pp. 146322–146330, 2019, doi: 10.1109/ACCESS.2019.2946202.
[98] Y. Yu, D. Barthaud, B. A. Price, A. K. Bandara, A. Zisman, and B. Nuseibeh, "LiveBox: A self-adaptive forensic-ready service for drones," *IEEE Access*, vol. 7, pp. 148401–148412, 2019, doi: 10.1109/ACCESS.2019.2942033.

[99] C. Perera, M. Barhamgi, A. K. Bandara, M. Ajmal, B. Price, and B. Nuseibeh, "Designing privacy-aware Internet of things applications," *Inf. Sci.*, vol. 512, pp. 238–257, Feb. 2020, doi: 10.1016/j.ins.2019.09.061.

[100] E. Yuan, N. Esfahani, and S. Malek, "A systematic survey of self-protecting software systems," *ACM Trans. Auton. Adapt. Syst.*, vol. 8, no. 4, p. 17, 2014, doi: 10.1145/2555611.

CHRISTIAN KRUPITZER (Member, IEEE) received the bachelor's, master's, and Ph.D. degrees from the University of Mannheim, Germany, in 2010, 2012, and 2018, respectively. Since October 2020, he is a Tenure Track Professor and leads the Department of Food Informatics, University of Hohenheim, Stuttgart, Germany. His research interests include applying principles of adaptive systems and machine learning for the IIoT (focusing on food production), intelligent transportation, and sports. He is involved in the organization of workshops and conferences, such as IEEE PerCom and IEEE ACSOS (former ICAC/SASO), and reviews for conferences and journals, i.e., ACM TAAS, IEEE IoTJ, or *Elsevier FGCS*.

TIMUR TEMIZER received the B.Sc. and M.Sc. degrees in business informatics from the University of Mannheim, Germany, in 2015 and 2018, respectively. As a part of his studies, he has focused on design patterns for SASs and their application in a smart highway system. He currently works as an IT Consultant with BCG Platinion across countries in Europe, Middle East, and Asia. His main work and research interests include agile delivery of software engineering projects, and the impact of changing IT skills on IT departments and organizations.

THOMAS PRANTL received the bachelor's and master's degrees from the University of Würzburg. He is currently a Doctoral Researcher with the Security Performance Research Group at the Software Engineering Chair, University of Würzburg.

CLAUDIA RAIBULET (Member, IEEE) received the master's degree in computer science from the POLITEHNICA University of Bucarest, Romania, in 1997, and the Ph.D. degree from the Politecnico di Torino, Italy, in 2002. She is currently an Assistant Professor with the Universitá degli Studi di Milano-Bicocca, Italy. Her research interests include software architectures, object-oriented methodologies, model-driven solutions, SASs, mobile systems, distributed systems, reverse engineering, software architecture reconstruction, and design patterns. She has coauthored more than eighty research articles published in international journals, conferences, and workshops. She is involved in referee activities for various international journals, as well as in organizing and program committees for international conferences and workshops.