IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 2, 2021, accepted April 27, 2021, date of publication May 6, 2021, date of current version May 18, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3078059

RADAR: Data Protection in Cloud-Based Computer Systems at Run Time

ZOLTÁN ÁDÁM MANN, FLORIAN KUNZ, JAN LAUFER, JULIAN BELLENDORF, ANDREAS METZGER, AND KLAUS POHL
paluno—The Ruhr Institute for Software Technology, University of Duisburg-Essen, 45127 Essen, Germany

Corresponding author: Zoltán Ádám Mann (zoltan.mann@paluno.uni-due.de)

This work was supported by the European Union's Horizon 2020 Research and Innovation Programme under Agreement 731678 (RestAssured) and Agreement 871525 (FogProtect).

ABSTRACT The protection of confidential data (e.g., personal data) is a concern of increasing importance. Data processing applications are often deployed in cloud or fog/edge computing systems. Such cloud-based systems may change dynamically during operations, for example because of changes in the users, in the deployed software services, or in the infrastructure. As a result, both the threats to data protection and the availability of data protection mechanisms may change at run time, making efficient data protection very challenging. This paper presents RADAR (Run-time Adaptations for DAta pRotection), an approach for ensuring data protection in dynamically changing cloud-based systems. RADAR analyzes the configuration of the cloud-based system automatically at run time, to detect changes in the threats to data protection or in the availability of data protection mechanisms. If needed, RADAR automatically adapts the cloud-based system to ensure the continued satisfaction of data protection requirements. From multiple possible adaptations that lead to the satisfaction of data protection requirements, RADAR chooses an adaptation that has the lowest negative implication on other goals, like costs and the availability of functionality. RADAR is a comprehensive approach that combines pattern-based detection of problematic system configurations with model-based automatic run-time adaptations and a search algorithm for finding the best adaptation. RADAR is validated using two case studies from the cloud and fog computing domains, and the scalability of the approach is evaluated using a set of controlled experiments.

INDEX TERMS Cloud computing, data protection, edge computing, fog computing, privacy, security, self-adaptation.

I. INTRODUCTION

Many software systems process, store, or transfer data that must be protected from unauthorized access. For example, personal data must be protected in compliance with applicable laws, such as the General Data Protection Regulation (GDPR) of the European Union (EU) [1]. Also non-personal data may require protection, e.g., in the case of trade secrets. Given our society's increasing reliance on data and on the processing of data by software systems, data protection is a concern of increasing importance in the design, development, and operation of software systems [2].

Data processing applications are often deployed in the cloud [3]. While offering easy access to large computational capacity, cloud platforms are associated with special data protection risks, stemming from virtualization, multi-tenancy etc. [4]. The complexity of cloud systems, including many different hardware and software components, services, and different types of stakeholders, offers a large attack surface, thus making it especially challenging to protect data stored and/or processed in the cloud [5]. Recent trends to distribute cloud-like services to the network edge, often referred to as fog computing or edge computing, increase the difficulties of data protection even further [6].

An important property of cloud-based systems (such as cloud and fog/edge computing systems) is that they **dynamically change during run time**. For example, new hardware or software components may be deployed at any time, existing hardware or software components may be changed or removed, tenants may join or leave, applications are automatically scaled to cope with varying workloads etc. While this flexibility is a major advantage of cloud and fog/edge

The associate editor coordinating the review of this manuscript and approving it for publication was Kuo-Hui Yeh.

services [7], it is a challenge for data protection. On the one hand, the *threats to data protection* may change during run time. For example, the migration of a software component from a trusted to a non-trusted environment (e.g., from a private to a public cloud) may significantly increase the risk that an unauthorized party gains access to the processed data. On the other hand, the *availability of data protection mechanisms* may also change during run time. For example, new machines supporting secure hardware enclaves may become available [8]. For these reasons, the **optimal choice of data protection mechanisms cannot be fully determined at design time**, since it depends on the specific cloud/fog/edge configuration which may evolve at run time.

There are many different techniques for protecting data, such as encryption, anonymization, and access control [9]. Yet, the problem of effectively and efficiently protecting data in complex systems is largely unsolved [10]. One particular deficit is that most existing data protection mechanisms offer only local protection, e.g., protecting the transfer of data between two components of the system. On the other hand, successful data protection must be end-to-end in several aspects: throughout the whole system, across the whole data lifecycle, taking into consideration all stakeholders [11]. To achieve **end-to-end data protection**, multiple data protection mechanisms have to be **combined**.

However, combining all conceivable data protection mechanisms is typically not feasible. Several **existing data protection mechanisms have some limitations or drawbacks**. For example, a way to achieve secure data processing is by using special hardware supporting secure hardware enclaves [8]. However, such special machines may not be available in all computing environments, or as scarce resource they may be more expensive to rent than other machines without special security features. Another alternative is to use anonymized data to protect the privacy of the involved data subjects [12]. However, anonymization degrades the utility of the data: for example, if the names and addresses of persons in a dataset are removed or overwritten with fictive values, this limits the kinds of queries that can be answered based on the dataset. Therefore, selecting the most appropriate data protection mechanisms in a given situation is not straight-forward, and should consider not only the data protection requirements, but also other system goals like costs and the availability of functionality.

To address these challenges, we present a novel approach for ensuring the **continued fulfillment of data protection requirements at run time**, in an automatic way. Our approach, called RADAR (Run-time Adaptations for DAta pRotection), takes into account

- the current configuration of the cloud/fog/edge system;
- the threats to data protection stemming from the system configuration;
- the availability of different data protection mechanisms in the given system configuration;
- the ability of the available data protection mechanisms to mitigate the detected threats;
- the impact of the data protection mechanisms on other goals, such as costs and the availability of functionality.

We use techniques from the field of models@run.time [13] to reason about the current configuration of the cloud/fog/edge system, pattern matching [14] to detect problematic configurations (i.e., configurations that threaten data protection), run-time adaptations to mitigate the detected problematic configurations [15], and optimization to find the best adaptation taking into account the impact on data protection and other goals, like functionality and costs.

RADAR is the first comprehensive approach to ensure data protection at run time while also optimizing other system goals in cloud/fog systems. In particular, RADAR possesses the following set of features which differentiates it from all existing approaches that we are aware of:

- Most existing approaches only address design-time activities (e.g., [16]) or only the detection of threats at run time (e.g., [17]). In contrast, RADAR spans the *whole range of activities* from capturing problematic configurations and potential adaptations at design time to automatic detection and automatic mitigation of threats at run time.
- Most existing approaches only support specific types of threats and mitigation actions (e.g., [18]). In contrast, RADAR can handle *arbitrarily complex problematic configurations* and *arbitrarily complex adaptations* of the system configuration.
- Most existing approaches focus on handling specific attacks on system security (e.g., [19]). In contrast, RADAR focuses on detecting and mitigating system *configurations with high risks* of violating data protection requirements.
- Most existing approaches focus only on data protection or security (e.g., [20]). In contrast, RADAR also takes into account *other goals* like costs and the availability of functionality, in two ways. First, from the possible adaptations for mitigating a data protection threat, RADAR chooses one that optimizes the other goals. Second, RADAR ensures that adaptations aimed at optimizing other goals are only carried out if they do not threaten data protection.
- Most existing approaches use off-the-shelf solvers for finding adaptations (e.g., [21]). In contrast, we experiment with different *search algorithms*, and devise a best-first search algorithm that is especially effective for this challenging algorithmic problem, leading to improved scalability.

The paper is organized as follows. Section II presents two examples to motivate our work. Section III identifies a set of requirements that the approach should fulfill. Section IV analyzes related work. Section V presents our proposed approach on a conceptual level, followed by details of a prototypical technical realization in Section VI. Section VII describes the experience from the practical evaluation of the approach. Section VIII discusses the findings and puts the results in perspective, while Section IX concludes the paper. The Appendix presents a formalization of the approach and details of the technologies used in the prototypical implementation.

II. MOTIVATING EXAMPLES

To motivate our work, we describe two example systems, one from the cloud computing domain and one from the fog computing domain.

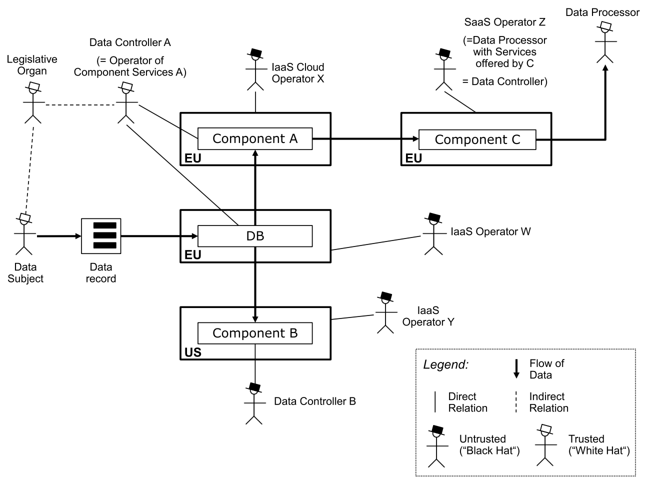


FIGURE 1. Example cloud system, based on [22].

A. CLOUD EXAMPLE

Figure 1 shows schematically the configuration of a cloud system of an industry partner in the project RestAssured,[1] based on [22]. In the shown example, personal data ("Data record") about users within the EU ("Data Subject"[2]) is processed by a company ("Data Controller[3] A"). The "Data Subject" explicitly agreed to the processing, as he or she trusts "Data Controller A." "Data Controller A" stores the "Data record" in an unencrypted database ("DB") operated by "IaaS[4] Operator W." The application ("Component A") of "Data Controller A" is operated on the infrastructure of "IaaS Operator X." The "Data Subject" trusts neither "IaaS Operator W" nor "IaaS Operator X." Another actor ("Data Processor"[5]) connects to "Component A" by using an application ("Component C") and receives the "Data record." The "Data Processor" is trusted by the "Data Subject." "Component C" is run by the untrusted "SaaS[6] Operator Z." In addition, another company ("Data Controller B") accesses the same database ("DB") using an application ("Component B"). The "Data Subject" trusts neither "Data Controller B" nor "IaaS Operator Y" that operates the infrastructure for "Component B."

This configuration is associated with multiple threats to data protection. While the "Data Subject" trusts both "Data Controller A" (the operator of the software that processes the data of the "Data Subject") and the "Data Processor" that in the end receives the data of the "Data Subject," other, untrusted actors along the path ("IaaS Operator W," "IaaS Operator X," "SaaS Operator Z") may have the opportunity to gain unauthorized access to the data. Another problem is that "Component B," which also has access to the data of the "Data Subject," is operated in the US. The GDPR prohibits the processing of personal data of EU citizens outside the EU (except in some explicitly allowed cases), thus this situation represents a compliance threat.

It is important to note that these threats arise from the interplay of multiple entities, and not from a property of a single entity. Moreover, such data protection threats can arise and vanish dynamically at run time, for example due to operator changes, changes in trust levels, or changes in the connections between components.

To address data protection threats arising at run time, adaptations may be carried out. Examples of such adaptations include turning on the encryption of data, migration of software components, or disabling the data processing functionalities affected by the threat. However, such adaptations may adversely impact other system goals. For instance, disabling data processing functionalities limits the functionality available to users and hence should be used only as a last resort. Also, IaaS operators may charge different fees for the use of their infrastructure; hence, different configurations may incur different costs (e.g., a migration between IaaS clouds may lead to a change in costs). Therefore, the choice of the adaptation is a crucial decision. From multiple possible system configurations that satisfy data protection requirements, the one with the lowest costs that does not restrict functionality should be selected.

B. FOG EXAMPLE

Figure 2 shows schematically the configuration of a fog computing system of an industry partner (Nokia) in the project FogProtect,[7] based on [23]. This example is a case study from the smart manufacturing domain, called "Factory in a Box" (FiaB). FiaB is a complete production environment, integrated in a standard 20-feet freight container. The computing capabilities within FiaB are considered as a fog node. This fog node is connected on the one hand with a number of end devices ("Sensors," "Robot," "AR/VR glasses"), and on the

[1] https://restassuredh2020.eu/

[2] Data Subject is a term defined by the GDPR: "A data subject is any person whose personal data is being collected, held or processed" (https://eugdprcompliant.com/what-is-data-subject/).

[3] Data Controller is a term defined by the GDPR: "The data controller determines the purposes for which and the means by which personal data is processed" (https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/obligations/controller-processor/what-data-controller-or-data-processor_en).

[4] Infrastructure as a Service

[5] Data Processor is a term defined by the GDPR: "The data processor processes personal data only on behalf of the controller. The data processor is usually a third party external to the company" (https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/obligations/controller-processor/what-data-controller-or-data-processor_en).

[6] Software as a Service

[7] https://fogprotect.eu/

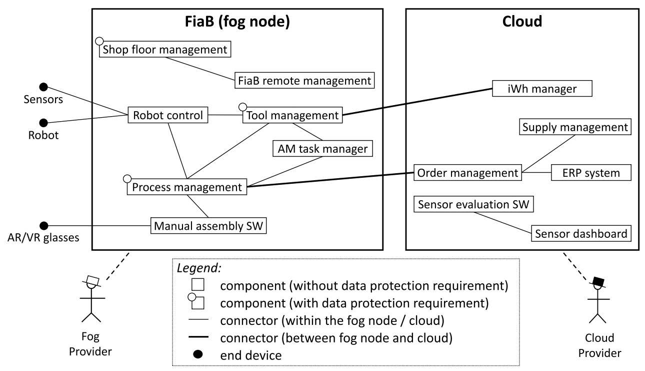


FIGURE 2. Example fog system, based on [23].

TABLE 1. Requirements.

R1:	At design time, it should be possible to capture the problematic configurations, i.e., configurations threatening data protection.
R2:	At design time, it should be possible to capture the possible types of adaptations of the system.
R3:	During run time, it should be detected automatically if the system gets into a problematic configuration.
R4:	During run time, problematic configurations should be automatically mitigated by appropriate adaptations.
R5:	From possible run-time adaptations mitigating the data protection threats, the best one should be chosen, taking into account costs and functionality.
R6:	In non-problematic configurations, adaptations to improve costs and functionality should be carried out if and only if they do not lead to problematic configurations.

other hand with a central cloud using a public network. This heterogeneous infrastructure hosts a number of applications. The applications consist of a varying number of components that can be placed either in the fog node or in the cloud.

Some components may process sensitive data. For example, the FiaB can produce tailored products for individual customers (so-called lot-size-1 production), for which personal data may have to be processed. Placing such components into the public cloud would pose a threat to data protection. Hence, if a component processes sensitive personal data, it has to be placed in the fog node.

During run time, many system parameters can change dynamically, e.g., the resource requirements of components or whether a component processes personal data. After such changes, adaptations may be necessary, in particular to ensure that the data protection constraints are still met. For example, components can be migrated between the fog node and the cloud. However, such adaptations may impact other goals. For instance, since the fog node has only limited computing power, migrating a component to the fog node may make it necessary to switch off computation-intensive features, thus limiting the available functionality. Hence it is important to select the adaptation that restricts the functionality as little as possible while ensuring data protection.

III. REQUIREMENTS

We identified several requirements that an approach to ensuring data protection at run time should satisfy. The requirements are summarized in Table 1 and explained in the following.

As we have seen in the motivating examples, problematic system configurations (i.e., configurations threatening data protection) may occur at run time. Our central requirements are hence to detect such problematic configurations (**R3**) and resolve them (**R4**) automatically at run time. Some aspects need to be highlighted in connection with these requirements:

- We are interested in detecting and mitigating problematic configurations, as opposed to detecting and mitigating the occurrence of specific attacks. Although the detection and mitigation of the occurrence of specific attacks is also an important field of research (including intrusion detection [24] and intrusion protection systems [19]), detecting and mitigating problematic configurations is associated with significant advantages. By detecting and mitigating problematic configurations, attacks can be proactively prevented, as opposed to reacting when it may be too late. In addition, data protection regulations like the GDPR require the mitigation of threats before they materialize as specific attacks.
- The large complexity of modern computing systems is a major source of data protection challenges. In line with that, we are focusing on threats to data protection stemming from the structure, properties, and interplay of different entities, which is captured by the *configuration* of the system.

To support the automatic detection of problematic configurations at run time, as stipulated by R3, it has to be determined what the problematic configurations are. According to current best practices, this requires a risk assessment process, in which security and privacy experts assess the kinds of assets to protect, the relevant threats and potential vulnerabilities, and the business impact of potential misbehaviors [25], [26]. Hence this process must take place at design time. The result of this process should be captured in a form that will be usable by appropriate algorithms at run time. This is formulated by **R1**. It should be noted that not *every* possible threat is to be mitigated. Only those configurations should be captured as problematic configurations, which would be associated with an unacceptably high risk of data protection violation.

To support the automatic mitigation of problematic configurations at run time, as stipulated by R4, the space of possible adaptations that can be used for mitigation has to be defined. This, too, has to be a design-time process in which software and infrastructure experts define what types of adaptations are possible, including in particular the possible run-time adaptations (activation, deactivation, reconfiguration) of data protection mechanisms. This is formulated by **R2**.

In a specific problematic configuration, there can be multiple possible adaptations for mitigating the problem. Different data protection mechanisms may have very different implications on the available functionality and on costs. Hence it is important to choose the best possible adaptation taking into account all these factors, as captured by **R5**.

In a self-adaptive system (such as in cloud/fog/edge systems), there can be many other reasons for performing adaptations, not only for ensuring data protection. Other reasons may include for example reducing costs. It is important that data protection is taken into account also in the planning of adaptations for such reasons. It has to be avoided that an adaptation for some reason (e.g., to reduce costs) leads to a problematic configuration in terms of data protection. This is stipulated by **R6**.

Both R5 and R6 relate to other system goals than data protection. Such other goals in cloud-based systems could include energy consumption reduction [27], cost reduction [28], latency minimization [29], and many others [30]. In this paper, we focus on the minimization of costs and the maximization of the available functionality as two representative examples, but the work presented here could be extended to address other goals as well.

Together, R1–R6 represent a challenging set of requirements that an approach for ensuring data protection in dynamic environments has to fulfill.

IV. RELATED WORK

Table 2 summarizes to what extent existing approaches for the detection and mitigation of data protection threats (or more generally, security threats) meet the requirements defined in Section III.

TABLE 2. Overview of previous approaches and their coverage of the requirements from Table 1. "●" means that the approach fully meets the given requirement, whereas "○" means partial fulfillment of the requirement.

Paper	R1	R2	R3	R4	R5	R6
Alebrahim et al. [16]	●	○				
Bürger et al. [31][20]	●	●	●	○		
Gonzalez-Granadillo et al. [32] [33]	○	○		○	○	
Iannucci & Abdelwahed [19]	○	●	○	○	○	
Kritikos et al. [34]		●	○	○	○	
Kunz & Mann [35]	●		●			
Massonet et al. [18]	○		○			
Nguyen et al. [36]	●	●		○		
Nostro et al. [37]	●	○		○		
Palm et al. [38]	●					
Pasquale et al. [39]	●		●	○		
Pasquale et al. [17]	●		●			
Puppala & Pasupuleti [40]	○	●		○	○	
Salehie et al. [41]		●		○	○	
Sartoli & Namin [15]	●	○	●	●		
Schmieders et al. [42]	○		○			
Tsigkanos et al. [21]	●	●	●	●		
This work (RADAR)	●	●	●	●	●	●

Several approaches focus on modeling and analyzing security threats during system design. For instance, Alebrahim *et al.* perform threat analysis based on a model including the physical and virtual entities of a cloud system, its providers and customers, as well as entities like legislators that are indirectly linked to the system [16]. Palm *et al.* use patterns to capture problematic configurations in cloud system models at design time [38].

In contrast to RADAR, such approaches do not address the dynamic appearance of threats at run time, and assume that control and mitigation strategies can be applied manually by human experts.

With the rise of cloud computing, the run-time detection of dynamically arising threats became important. For example, Massonet *et al.* [18] and Schmieders *et al.* [42] propose approaches to detect emerging violations of data location restrictions in cloud systems through appropriate monitoring and analysis techniques. These approaches are limited to one specific type of threat stemming from geo-location violations, and do not mitigate the identified threats.

Pasquale *et al.* address more general threats and capture them with attack scenarios [17]. The approach aims to check whether the conditions for an attack are met to recognize it at an early stage. The approach assumes that the behavior and the goals of an attacker are well known, which may limit its practical applicability. In addition, the approach does not aim at mitigating the identified threats, but only at evidence collection.

In earlier work, we presented our preliminary approach to automatically find problematic configurations in a run-time model [35]. However, also that work did not include the mitigation of detected threats.

Bürger *et al.* present an approach called SecVolution for supporting the evolution of systems to ensure the fulfillment of Essential Security Requirements [20], [31]. Changes in the Security Context Knowledge (e.g., changes in relevant legislation) may trigger the invocation of Security Maintenance Rules. In contrast to our approach, Bürger *et al.* focus only on partial automation, leaving significant work in the evolution process to the developer team.

Pasquale *et al.* argue that the topology of both physical and virtual assets is important to capture the state of a system at run time [39]. If a threat for data security is identified in a possible following state, the system needs to be prevented from reaching this state. Tsigkanos *et al.* build on this idea and use bigraphs to model the topology and the security requirements [21]. Model checking is used to analyze threats and plan adaptations to mitigate the threats. A similar approach is presented by Sartoli and Namin who use answer set programming to reason about security violations and a limited set of possible mitigation actions [15]. All of these approaches were defined for cyber-physical systems, where physical constraints help to limit the possibilities – both for attacks and for adaptations. The problem considered in our paper is more difficult, first because of the lack of such limitations and second because we also consider costs and functionalities together with data protection threats. Nevertheless, the advanced algorithms used by RADAR make it fast for even large systems, whereas the approach of

Tsigkanos *et al.* takes a long time to execute already for simple systems.

Nostro *et al.* use attack paths to identify the actions that insider attackers can perform to reach their goals and the respective mitigation actions to prevent those insider threats [37]. Nguyen *et al.* use Bayesian Attack Graphs to model threats, and propose Moving Target Defense by means of VM migration to mitigate threats [36]. These approaches involve very specific types of adaptation to mitigate a threat, whereas RADAR can use a wide range of adaptations and dynamically decide which one is the most appropriate in a given situation.

Some papers propose methods for selecting the best action, based on various metrics, to mitigate a given security attack. Gonzalez-Granadillo *et al.* propose an approach which selects an appropriate response plan for mitigating identified attacks, taking into account financial and operational impact of the possible mitigation actions [32], [33]. Iannucci and Abdelwahed use multi-agent Markov Decision Processes to model an intrusion response system, with the aim of finding optimal responses, considering costs and response time [19]. Kritikos *et al.* propose to configure an intrusion detection system with security rules taking into account speed, accuracy, and costs [34]. The approach of Puppala and Pasupuleti aims at selecting appropriate countermeasures to attacks captured by an attack graph, also considering the costs of the countermeasures [40]. Salehie *et al.* use a causal network to analyze the implications of asset changes on security concerns and to select appropriate countermeasures, taking into account the impact of the decision on the fulfillment of non-functional requirements like performance or usability [41].

These approaches are focused on specific attacks against system security (e.g., network intrusion) described by independent attack paths, and consider local security measures (e.g., patching or rebooting a server). In contrast, RADAR captures and detects data protection threats stemming from the arbitrarily complex interplay of components, and is able to perform arbitrarily complex adaptations of the system configuration.

Summary of related work. None of the approaches in the literature that we are aware of fulfills all of the requirements defined in Section III. In particular, most of the related papers have at least one of the following limitations:

- They focus on the detection of problematic configurations, but do not mitigate the found problematic configurations.
- They are limited to specific kinds of problematic configurations (e.g., geo-location problems).
- They address specific security attacks (e.g., network intrusion), and not configurations posing threats to data protection.
- They do not consider the data protection implication of adaptations targeting other system goals (e.g., cost improvement).

Our approach, which addresses all requirements from Section III, thus goes significantly beyond the state of the art.

V. THE PROPOSED APPROACH

In this section, we present **RADAR** (**R**un-time **A**daptations for **DA**ta p**R**otection). RADAR is a model-based approach to automatically ensure data protection at run time in cloud and fog systems using adaptations. RADAR uses a run-time model of the system and its environment [13]. The run-time model is a compact representation of the relevant entities, attributes and relations, which allows automated reasoning about data protection threats and possible adaptations.

A. OVERVIEW

As shown in Figure 3, RADAR is divided into two phases, the design time (upper part of the figure) and the deployment / run time (lower part of the figure). The main focus of RADAR is on run time, but some preparatory activities are required at design time.

At **design time**, a *meta-model* is provided, which defines the modeling constructs that can appear in the run-time model. Referring to this meta-model, *problematic configuration patterns* (PCPs) are modeled. Problematic configuration patterns describe configurations that pose threats to data protection, in line with requirement R1. The problematic configuration patterns are identified through a manual data protection risk assessment. To provide means for mitigating problematic configurations, *adaptation rules* are defined. An adaptation rule describes a way to alter the run-time model (and the system itself) to mitigate a problematic configuration, thus fulfilling requirement R2.

At **deployment time**, a *run-time model* is generated which represents the planned configuration of the system. At **run time**, this run-time model is continuously updated by means of monitoring so that it always reflects the current configuration. Both at deployment time and at run time,[8] the run-time model is checked after each update for instances of problematic configuration patterns (*problematic configuration identification*), thus fulfilling requirement R3. Found instances of problematic configuration patterns are called *PCP instances*.

If a PCP instance is detected, a *reconfiguration* is triggered [43], in line with requirement R4. Reconfiguration is carried out using *adaptations*, i.e., instances of the adaptation rules [44]. Since there may be several possible adaptations to eliminate the found PCP instances, these adaptations are compared to select the best one. The potential adaptations are examined using a *functionality analysis* and a *cost analysis*. Depending on the results of the comparison, the best adaptation is selected and applied (i.e., executed in the real system), thus fulfilling requirement R5, and the run-time model is

[8]Figure 3 shows deployment time and run time combined, because the same activities are performed. The difference is only that at deployment time the planned configuration is analyzed and possibly improved, whereas at run time the same is done with the actual configuration.

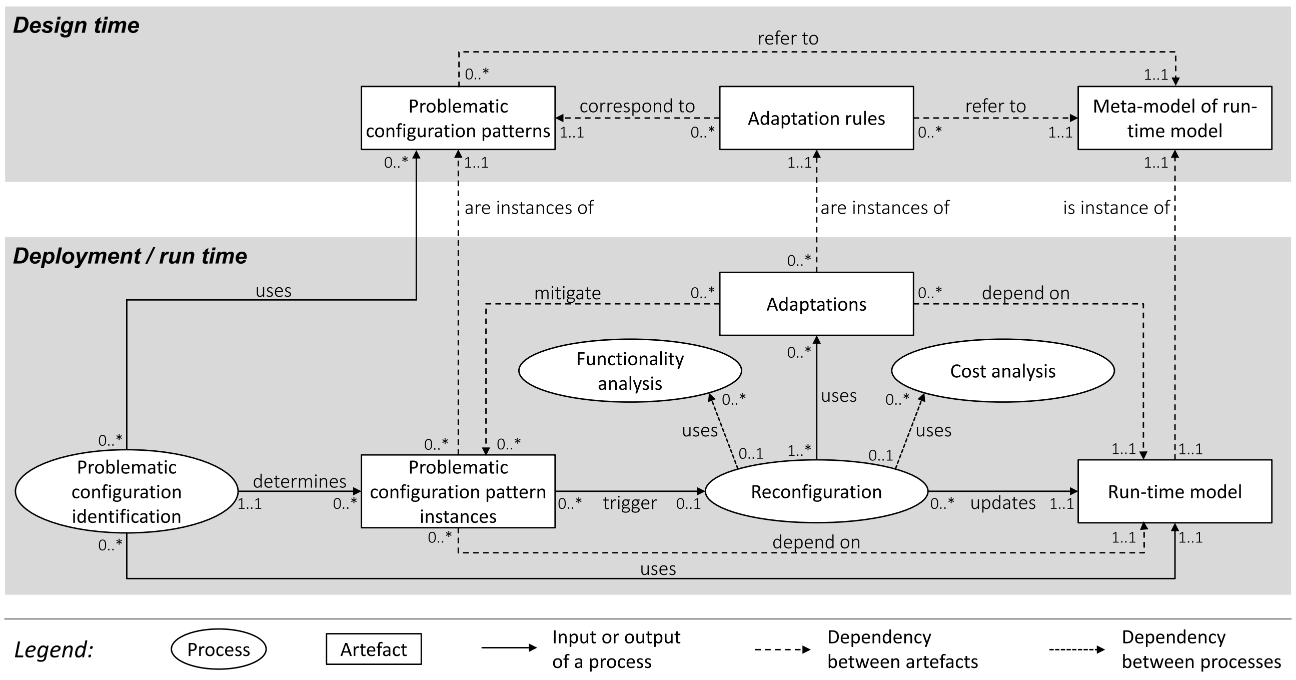


FIGURE 3. Overview of the RADAR approach.

updated accordingly. If no PCP instance is detected in the run-time model, the same reconfiguration algorithm is used to search for adaptations that would improve costs or functionality without introducing PCP instances, thus fulfilling requirement R6.

The individual elements of RADAR are described in detail in the following subsections. In addition, the Appendix contains a formal description of the approach.

B. META-MODEL

The meta-model in RADAR serves as the basis for the run-time model. The meta-model specifies which types of nodes and edges can exist in the run-time model and which attributes the nodes can have. The meta-model can be regarded as a simplified UML class model, whereas the run-time model can be regarded as a simplified UML object model. The problematic configuration patterns also use the types, attributes, and relations from the meta-model. Thus, the meta-model ensures compatibility between the problematic configuration patterns and the run-time model.

We use a meta-model designed for cloud and fog computing. However, RADAR is independent of the specific meta-model, so the given meta-model can be changed or extended if needed to apply RADAR in a specific context.

Our specific meta-model is based on an established standard for modeling cloud systems, namely TOSCA (Topology and Orchestration Specification for Cloud Applications). TOSCA was originally conceived for the specification of cloud system topologies, but has also been extended to other related paradigms like fog computing [45]. In TOSCA, there are two different types of elements making up a topology: nodes and relations. The nodes represent the individual components, while the relations represent the connections between the nodes. The nodes and relations are represented as node templates and relation templates in a topology template. Several objects can be instantiated on the basis of a template.

TOSCA defines a set of normative node types and corresponding relations, and allows the definition of further types. For our purposes, several further node types are needed (see Figure 4). To be able to reason about data protection, it is crucial to model the relevant actors [46]. For example, a data protection violation can occur when one actor has access to data of another actor without the latter's consent. TOSCA does not provide normative node types to represent actors. Thus, we extend TOSCA with new node types to represent actors. These node types inherit from the abstract node type "Actor" and its direct descendants "DataSpecificRole" and "CloudSpecificRole." Subtypes of "DataSpecificRole" represent actor roles relating to data, e.g., producing or processing data. Subtypes of "CloudSpecificRole" represent actor roles relating to a cloud service, in terms of developing, operating, or using a service. These roles are divided into the tree layers IaaS (Infrastructure-as-a-Service), PaaS (Platform-as-a-Service) and SaaS (Software-as-a-Service).

Furthermore, the data that needs protection must also be modeled, requiring further node types. The added node types are the abstract "DataSet" node type from which the "DataFlow" and "StoredDataSet" node types inherit, and the "Record" node type. The "Record" node type represents individual data entries, which are collected within a "DataSet." The difference between "StoredDataSet" and "DataFlow" is that "StoredDataSet" represents a data set

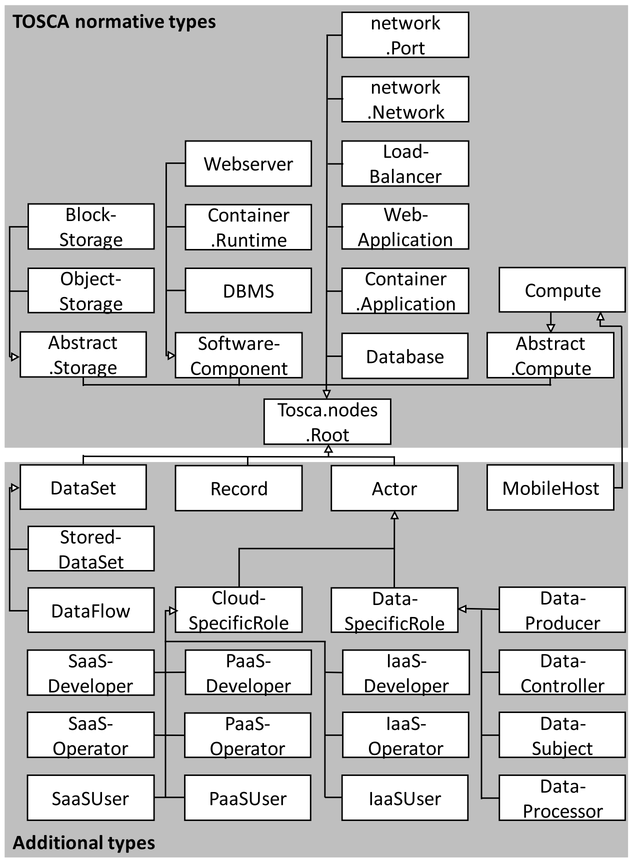


FIGURE 4. TOSCA inheritance tree with additions. (Relations were omitted for clarity.).

persisted in a database, while a "DataFlow" represents data in transit.

With the addition of node types, the addition of relations is also necessary. The relations can be seen in Table 3. The first five relations, which have a gray background, are normative TOSCA relation types. The following relations were added, mostly to represent relations between actors and data or between data and the computing infrastructure.

In our earlier work, we used a proprietary meta-model to represent cloud systems [46]. Because of the advantages of a standard-based modeling approach, we decided to switch to the meta-model based on TOSCA as presented here. The change in the meta-model went smoothly, as the RADAR approach is not dependent on the specifics of the meta-model.

C. PROBLEMATIC CONFIGURATION PATTERNS (PCPs)

Modern computer systems exhibit a complex interplay of different components. Even if each component is secure in itself, the interplay of multiple components may still give rise to a problematic configuration, i.e., a configuration in which data protection is threatened. To address this issue, RADAR uses problematic configuration patterns (PCPs), based on the notion of risk patterns introduced in [22]. A PCP describes

TABLE 3. Relations of the meta-model. (Relations with a gray background color are part of the normative TOSCA notation.).

Relation name	Source node type	Target node type
hostedOn	Database	DBMS
hostedOn	SoftwareComponent	Compute
hostedOn	WebApplication	WebServer
attachesTo	BlockStorage	Compute
hostedOn	Container.Application	Container.Runtime
trusts	Actor	Actor
connectedTo	Compute	Network.Network
stores	Database	StoredDataSet
streamsTo	DataFlow	Database
processes	DataProcessor	StoredDataSet
belongsTo	Record	DataSubject
operates	IaaSOperator	Compute
connectedTo	Network.Network	Network.Network
partOf	Record	DataSet
creates	SoftwareComponent	StoredDataSet
accesses	SoftwareComponent	Database
controls	DataController	SoftwareComponent

a pattern which, if found in the run-time model, means that the system is in a problematic configuration. A PCP models an interplay of some components that threatens data protection. Instances of PCPs as sub-structures within the run-time model are to be avoided. If an instance of a PCP arises in the run-time model, then the run-time model and the underlying system have to be changed in such a way that the PCP instance disappears from the run-time model. This pattern-based approach has the advantage that a PCP succinctly defines an infinite class of problematic configurations (namely all those configurations that contain an instance of the given PCP).

To create PCPs, IT security experts identify at design time the constellations of system components that would threaten data protection. For capturing such constellations, we define a graphical modeling language, called PCP language. The PCP language refers to the meta-model to ensure that the PCPs are compatible with the run-time model.

1) GRAPHICAL NOTATION

The PCP language is inspired by the UML object diagram notation and defines the elements that can be used to represent a PCP. This includes objects, attributes, and relations. The syntax of the PCP language is shown in Figure 5. Objects are represented by a rectangle, in which a colon is placed followed by the type of the object. Optionally, a section with attribute descriptions may be given below the object type. Each line of the attribute description section consists of the name of an attribute, followed by either "==" or "!=," and a value.

Relations are denoted by lines connecting pairs of objects. The name of the relation and an arrowhead indicating the direction of the relation are written next to the line. The name of the relation may optionally be preceded by the annotation `<<does not exist>>`.

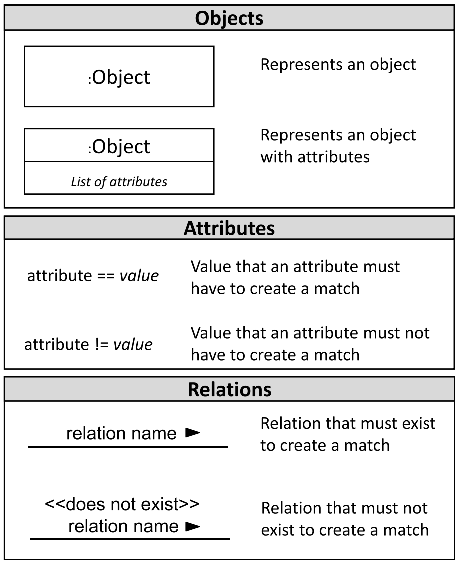

Objects
:Object
Represents an object
:Object
List of attributes
Represents an object with attributes
Attributes
attribute == value
Value that an attribute must have to create a match
attribute != value
Value that an attribute must not have to create a match
Relations
relation name ►
Relation that must exist to create a match
<<does not exist>> relation name ►
Relation that must not exist to create a match

FIGURE 5. Problematic configuration pattern language.

2) SEMANTICS

If a set of objects can be found in the run-time model so that (1) their types match the object types in the PCP, (2) their attribute values are in line with the attribute descriptions in the PCP and (3) their relations are in line with the relation descriptions in the PCP, then these objects in the run-time model form an instance of the PCP. In this case, a match of the PCP has been found in the run-time model.

The condition that the attributes of the objects in the run-time model are in line with the attribute descriptions in the PCP means the following:

- For any attribute description of the form "`attribute == value`" in the PCP, the given attribute of the corresponding object in the run-time model has the given value.
- For any attribute description of the form "`attribute != value`" in the PCP, the given attribute of the corresponding object in the run-time model has a different value from the given one.

Similarly, the condition that the relations of the objects in the run-time model are in line with the relation descriptions in the PCP means the following:

- For any relation description without the annotation "`<<does not exist>>`" in the PCP, the given relation exists between the corresponding objects in the run-time model.
- For any relation description with the annotation "`<<does not exist>>`" in the PCP, the given relation does not exist between the corresponding objects in the run-time model.

The semantics is further detailed in Section V-F, and formalized in the appendix.

3) EXAMPLES

Figures 6–7 show two example problematic configuration patterns. These examples are taken from [22] and fitted to

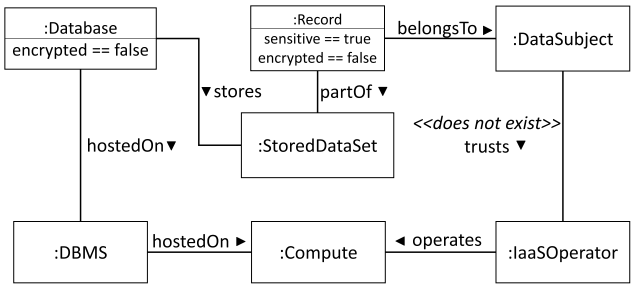


FIGURE 6. Example problematic configuration pattern A.

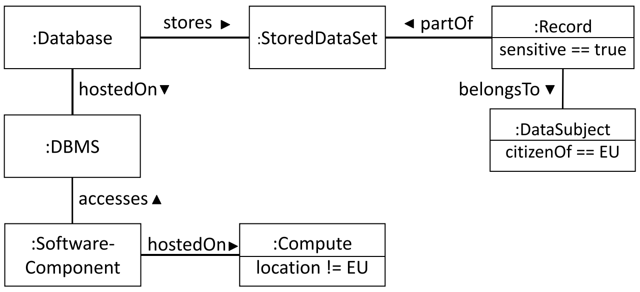


FIGURE 7. Example problematic configuration pattern B.

the current meta-model. Problematic configuration pattern A represents a situation where a data set contains a sensitive, unencrypted record belonging to a data subject. The data set is stored in a database in unencrypted form. The database is hosted by a DBMS, which in turn is hosted on a compute node. The compute node is operated by an IaaS operator who is not trusted by the data subject, as captured by the "`<<does not exist>> trusts`" relation description. Such a constellation of objects is indeed problematic, since the untrusted IaaS operator may gain unauthorized access to the sensitive data stored on the compute node operated by the IaaS operator.

The problematic configuration pattern in Figure 7 (problematic configuration pattern B) represents a situation where a data set contains a sensitive record of a data subject (an EU citizen). The data set is stored in a database, which is hosted on a DBMS which is accessed by a software component. The software component is hosted on a compute node outside the EU, as shown by the attribute description "`location != EU`." Such a constellation is indeed problematic, because the GDPR does not allow in general the processing of data of EU citizens outside the EU [1].

These examples demonstrate how PCPs can capture data protection issues that arise through the interplay of multiple components.

D. ADAPTATION RULES

If one or more PCP instances are found in the run-time model, they should be mitigated. To do this, the run-time model has to be adapted, together with the system that it represents [47]. On the model level, the adaptation may involve adding or removing objects, adding or removing relations, and changing attribute values. We propose the RADAR Adaptation Language for capturing at design time the potential types of adaptations in the form of *adaptation rules*.

Adaptation rules are associated with the PCPs that they may mitigate. It is important to note that a PCP may be mitigated by multiple adaptation rules. It is discussed in Section V-H how to choose the adaptation rule to apply in such cases.

An adaptation rule consists of three parts, similarly to the Event–Condition–Action structure of ECA rules [48]:

- PCP: The PCP to be mitigated by the adaptation rule. If an instance of this PCP is found in the run-time model, this event activates the adaptation rule.
- Precondition: Represents constraints on the run-time model that – in addition to the existence of a PCP instance – are required for the adaptation rule to be applicable.
- Adaptation action: Describes the changes to be carried out.

As a simple example, an adaptation rule could describe the following: if a software component accessing sensitive data is hosted on a server in an insecure environment (PCP), and there is another server available in a secure environment (precondition), then the software component should be migrated from the server in the insecure environment to the server in the secure environment (adaptation action).

RADAR adaptation rules are syntactically similar to the "security maintenance rules" of [20]. But in contrast to security maintenance rules which may include tasks for developers, RADAR adaptation rules only contain actions that can be executed automatically.

1) GRAPHICAL NOTATION

Using the RADAR Adaptation Language, an adaptation rule is captured by a single model using a graphical notation inspired by UML object diagrams. The PCP and precondition parts of the adaptation rule are modeled using the constructs of the PCP language, described in Section V-C.

For modeling adaptation actions, further notation is necessary, which is shown in Figure 8. Adaptation actions may include the creation and deletion of objects or relations, as indicated by the `<<create>>` and `<<delete>>` annotations. In addition, adaptation actions can also include the setting of the values of objects' attributes, using the "attribute = value" notation.

2) SEMANTICS

An adaptation rule expresses the following: if an instance of the PCP is found in the run-time model, and additionally also the precondition matches the run-time model, then the adaptation action can be carried out to change the run-time model, and this change could mitigate the found PCP instance.

Carrying out the adaptation action means the following. If an object is annotated with "`<<create>>`" in the adaptation rule, then the object is added to the run-time model. On the other hand, if the object is annotated with "`<<delete>>`," then the object is removed from the run-time model. Similarly, relations annotated with "`<<create>>`" and "`<<delete>>`" in the adaptation rule are added to respectively removed from the run-time model.

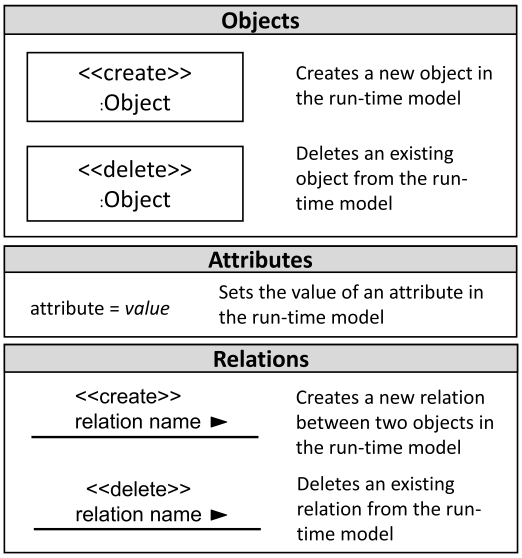

Objects
<<create>> :Object
Creates a new object in the run-time model
<<delete>> :Object
Deletes an existing object from the run-time model
Attributes
attribute = *value*
Sets the value of an attribute in the run-time model
Relations
<<create>> relation name ►
Creates a new relation between two objects in the run-time model
<<delete>> relation name ►
Deletes an existing relation from the run-time model

FIGURE 8. Constructs of the RADAR adaptation language for modeling adaptation actions.

Attributes cannot be added or removed, only their value can be changed. If the adaptation rule contains "attribute name = value" in an object, then the corresponding attribute of the object is set to the new value.

The semantics of adaptation rules is further detailed in Section V-H and formalized in the appendix.

3) EXAMPLE

Figure 9 shows an adaptation rule to mitigate the problematic configuration pattern A from Figure 6. In the upper left part is the problematic configuration pattern that has already been explained. In the upper right part is the precondition, specifying that there has to be another IaaS operator that is trusted by the data subject and also offers a compute node hosting a DBMS. In the bottom part is the adaptation action, specifying that the database is moved from the DBMS hosted on the compute node operated by the untrusted IaaS operator to the DBMS on the compute node operated by the trusted IaaS operator.

E. RUN-TIME MODEL

The run-time model represents the data-protection-related aspects of the configuration of the system and its environment. The run-time model, which is a UML object model, is an instantiation of the meta-model described in Section V-B. The run-time model is created during deployment. During run time, whenever the configuration changes, the run-time model is updated to represent the current

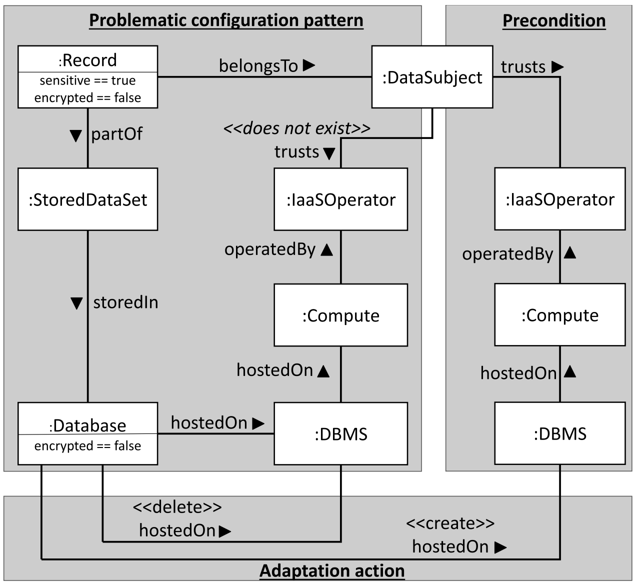


FIGURE 9. **Example adaptation rule.**

configuration. We assume that an appropriate monitoring functionality is in place that ensures that the run-time model correctly mirrors the actual system configuration, in line with similar assumptions in other works on self-adaptation using models at run time [49], [50].

Figure 10 shows an example run-time model, adopted from [22]. This run-time model is a representation of a real cloud system.

In the example there is a "DataSubject" that has some data in form of a "Record," which is part of a "StoredDataSet" and a "DataFlow." The "StoredDataSet" represents the data stored in the "Database," while the "DataFlow" is a representation for data in transit. The "StoredDataSet" is stored in a "Database," which is hosted on a "DBMS," which in turn is hosted on a "Compute" node. The "Compute" node is provided by an "IaaSOperator." The "DBMS" is accessed by multiple "SoftwareComponents." Additionally there are connections between the "DataSubject" and the other actors, showing which actors the "DataSubject" trusts.

F. PROBLEMATIC CONFIGURATION IDENTIFICATION

Using the already introduced concepts (PCP, run-time model), it is possible to detect problematic configurations. The current configuration is problematic, if and only if at least one part of the run-time model *matches* a PCP. In that case, the part of the run-time model matching the PCP is called a *PCP instance*.

A subset of objects in the run-time model matches a PCP, if and only if all of the following conditions are met:

- Each object appearing in the PCP must also exist in the run-time model.
- All attributes appearing in the PCP must have appropriate values in the run-time model.
- Relations appearing without annotation in the PCP must be also present in the run-time model. On the other hand, relations appearing with the `<<must not exist>>` annotation in the PCP must not appear in the run-time model.

It should be noted that attributes not represented in the PCP can have any value in the run-time model, without affecting whether there is a match. Similarly, relations not represented in the PCP may or may not exist in the run-time model without affecting whether there is a match.

In [22] it was proposed to use graph pattern matching to identify PCP instances in the run-time model. The goal of graph pattern matching is to find all matches of a given "pattern graph" in a "data graph" [14]. This is indeed similar to the task of identifying matches of PCPs in the run-time model. However, the above definition of a match includes concepts that are beyond the scope of basic graph pattern matching: types of objects, inheritance, attributes of objects, named relations. Therefore, a more powerful technique is needed to find matches of PCPs in the run-time model, which is presented in Appendix II.

G. PROBLEMATIC CONFIGURATION PATTERN INSTANCE

Problematic configuration pattern instances (PCP instances) represent problems found in the run-time model. They are parts of the run-time model matching a PCP. There can be several PCP instances in the run-time model, even multiple instances of the same PCP.

An example PCP instance is marked in the run-time model shown in Figure 10. The highlighted part of the run-time model is an instance of problematic configuration pattern A from Figure 6.

H. ADAPTATIONS

The identified PCP instances have to be mitigated by appropriate adaptations, i.e., by the application of appropriate adaptation rules. Suppose that a part of the run-time model has been identified as an instance of a PCP. Moreover, suppose that there is an adaptation rule that may mitigate the given PCP. The adaptation rule is *applicable* to the current run-time model, if and only if the run-time model satisfies the precondition of the adaptation rule.

Suppose that the adaptation rule is applicable. Then, *applying* the adaptation rule to the run-time model means performing the actions of the adaptation rule, i.e., creating and deleting objects and relations and setting attributes as prescribed by the adaptation rule.

I. COST ANALYSIS

As part of the reconfiguration process (see Section V-K), different adaptations are compared, and one of the comparison metrics is the cost implication of the adaptations. For this purpose, RADAR analyzes the costs incurred in a given configuration.

Some types of objects are relevant for the cost analysis, while others are not. For example, compute nodes may be rented from IaaS providers incurring costs, thus compute nodes are relevant for the cost analysis.

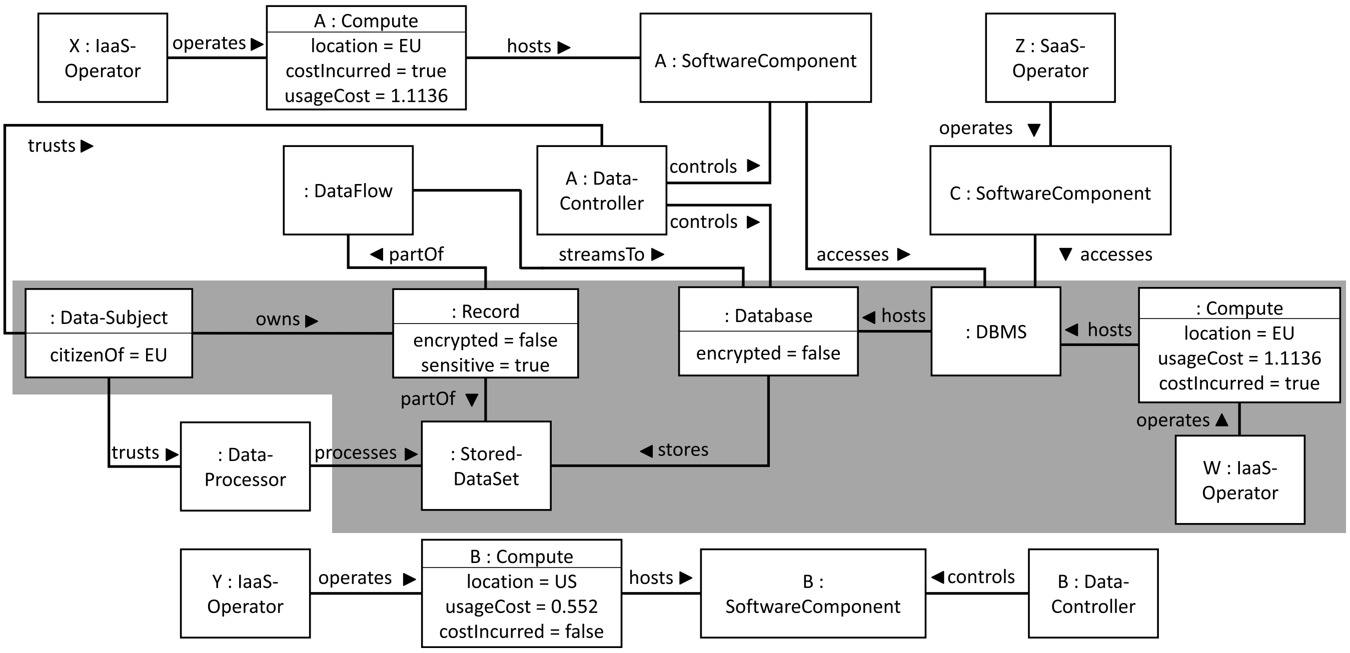


FIGURE 10. **Example run-time model. An instance of PCP A (see Figure 6) in the run-time model is highlighted with gray background.**

The types relevant for the cost analysis are associated with two cost-related attributes:

- The attribute `costIncurred` is Boolean and has the value `true` if and only if the given object is used and hence incurs costs.
- The attribute `usageCost` is a non-negative number specifying the costs incurred if the given object is used.

The total cost incurred in the configuration defined by a given run-time model is calculated by adding up the usage costs (`usageCost`) of the cost-relevant objects that are in use (`costIncurred == true`).

In the example of Figure 10, there are two Compute nodes with `costIncurred == true`, hence usage cost is the sum of their costs: $1.1136 + 1.1136 = 2.2272$.

J. FUNCTIONALITY ANALYSIS

Adaptations for protecting data may impair the availability of functionalities to users, for example by switching off some functionalities altogether or by making them inaccessible for some users. While such adaptations may be effective in protecting data and in some cases they may be the only way to protect data, the impairment of available functionality is certainly a drawback that needs to be considered when deciding which adaptation to make. Hence, within the reconfiguration process (see Section V-K), the availability of functionality in run-time models resulting from different adaptation possibilities is compared.

For quantifying the availability of functionality in a run-time model, RADAR uses the following approach:

- Objects of some given types are considered *functionality providers*. For example, we consider objects of the type `SoftwareComponent` as functionality providers.
- Objects of some given types are considered *functionality consumers*. For example, we consider objects of the types `SaaSUser` and `DataSubject` as functionality consumers.
- A functionality consumer can use the functionality provided by a functionality provider if and only if there is a path in the run-time model from the functionality provider to the functionality consumer, consisting of relations from a given set. For example, the relation `accesses` between a `SoftwareComponent` and a `DBMS` may appear in such a path, while the relation `trusts` between two `Actors` may not.
- The number of pairs of functionality provider and functionality consumer for which the functionality consumer can use the functionality provided by the functionality provider is used as a metric to quantify the availability of functionality in the given run-time model.

The functionality analysis proceeds as follows. All nodes of the run-time model are stored in a list. The nodes in the list are checked for nodes of functionality consumer types and functionality provider types. Then, starting from a consumer node, a depth-first search algorithm is used. The algorithm identifies paths of relations that are suitable to access functionalities, between a consumer node and a provider node. Once any node is visited, it is added to a list. This ensures that every node is only visited once. When a source node is reached, the counter of available functions is incremented.

In the example of Figure 10, the `DataSubject` can use the functionality provided by two `SoftwareComponents`. If an adaptation that removes the `Records` of the `DataSubject` from the `Database` were executed, the number of available functionalities would drop from two to zero. Thus, this adaptation should only be performed if there is no better way to ensure data protection.

K. RECONFIGURATION

There can be two reasons for looking for adaptations in the RADAR approach. Either at least one PCP instance has been identified in the run-time model and should be mitigated, or there is no PCP instance and the aim is to look for adaptations to improve the amount of available functionalities and reduce costs. In both cases, RADAR has to decide which adaptations to carry out. This decision is complicated by several factors[9]:

- There may be multiple adaptations to mitigate a given PCP instance. At least one of them should be selected.
- There may be multiple PCP instances in the run-time model, and they all have to be mitigated.
- An adaptation to mitigate a PCP instance may also mitigate another PCP instance, or may create new PCP instances in the run-time model.
- To mitigate all PCP instances in the run-time model, a sequence of adaptations may be needed.
- In a sequence of adaptations, an adaptation rule may be used multiple times.
- In a sequence of adaptations, the order of the adaptations may be important. For example, an adaptation may become applicable only after another adaptation has been carried out.
- The length of the sequence of necessary adaptations is not bounded. (This is because an adaptation may create new PCP instances in the run-time model, requiring new adaptations etc.)
- Different sequences of adaptations may have different impact on costs and available functionality.

RADAR uses an *adaptation planning algorithm* that takes as input the current run-time model, the identified PCP instances in the run-time model, and the set of available adaptation rules. If there is at least one PCP instance in the current run-time model, the aim of the adaptation planning algorithm is to determine a sequence of adaptations that transforms the current run-time model to a new model with no PCP instances, and with optimal impact on costs and available functionality.

If there is no PCP instance in the current run-time model, the aim is to improve the available functionality and/or reduce the costs, without introducing PCP instances. In this case, the planning algorithm uses the set of adaptation rules that are aimed at functionality and cost improvements. These adaptation rules use the same form as the adaptation rules aimed at mitigating data protection threats. The difference is that the PCP part of these adaptation rules can be empty. Otherwise, the adaptation planning algorithm works in both cases in the same way.

[9]The following list is formulated for the case in which the aim is to mitigate found PCP instances, but most items apply in analogous way also to the case in which functionality and costs should be improved

To reason about possible adaptation sequences, the adaptation planning algorithm constructs a search tree (see Figure 11). Every node in the search tree represents a possible run-time model. The root node represents the current run-time model (M^{RT}). A child node represents the run-time model obtained from the run-time model of the parent node through an adaptation. That is, if M and M' are two possible run-time models and M' is a child of M in the search tree, then there is an adaptation rule ad such that ad is applicable in M and applying ad to M results in M' ($ad(M) = M'$). In the example of Figure 11, each arc of the search tree is labeled with the adaptation rule used to create the child node from the parent node.

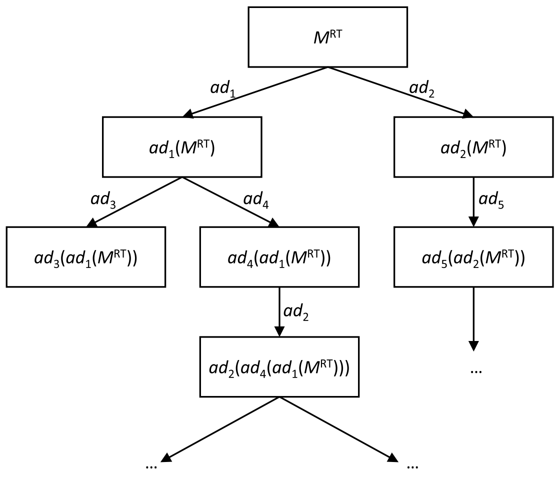


FIGURE 11. **Example search tree of the adaptation planning algorithm.**

The RADAR adaptation planning algorithm is schematically shown in Algorithm 1. The algorithm implements a *bounded backtrack search* in the space of possible run-time models reachable from the current run-time model via sequences of adaptations. During the search, the algorithm iteratively builds up the search tree by exploring the children of already found nodes. The set S consists of the nodes of the search tree that have already been found but not yet processed. At the beginning, S consists of the root node, which is the current run-time model (line 3). In a general step of the algorithm, a node is selected from S (line 5) and is evaluated in terms of PCPs (line 6). If the model is free of PCP instances, then it is also evaluated in terms of costs and available functionality (line 7). In this evaluation, the available functionality is the primary aspect of comparison, while costs act as a tie-breaker if the amount of available functionality is equal. The algorithm keeps track of the best solution found so far, i.e., the possible run-time model with no PCP instances and minimal impact on costs and available functionality (line 8), as well as the adaptation sequence leading to this solution

Algorithm 1 RADAR Adaptation Planning Algorithm

```
 1: best ← NULL                              ▷ best solution found so far
 2: bestPath ← NULL                          ▷ path from root to best solution
 3: $S$ ← {$M^{RT}$}                           ▷ set of nodes that are being explored
 4: while $S \neq \emptyset$ do
 5:     $M$ ← selectNode($S$)
 6:     if $M$ contains no PCP instance then
 7:         if $M$ is better than best then
 8:             best ← $M$
 9:             bestPath ← path from $M^{RT}$ to $M$
10:         end if
11:     end if
12:     $T$ ← generateChildren($M$)
13:     $S \leftarrow S \setminus \{M\} \cup T$
14:     if termination criterion then
15:         break
16:     end if
17: end while
18: return bestPath
```

(line 9). After the current node has been evaluated, it is removed from S and the children of the node are added to S instead (lines 12-13). To avoid an endless search in the potentially unbounded space of possible solutions, the search is aborted when a predefined termination criterion is met (lines 14-16). Such a termination criterion could be for instance a maximum allowed time budget for the execution of the algorithm. At the end, the algorithm returns the adaptation sequence leading to the best found solution (line 18), which is then executed in the real system.

The effectiveness of the algorithm may be significantly influenced by the way the next node to explore is selected (line 5). We experimented with the following strategies:

- Depth-first-search: the node that was put last into S is selected.
- Breadth-first-search: the node that was put first into S is selected.
- Random: a node is randomly selected from S.
- Best-first-search: the node which is associated with the lowest number of PCP instances is selected. From nodes with an equal number of PCP instances, the node with the highest amount of available functionality is selected. If there is still a tie, the node with the least cost is selected. If there is still a tie, the node is selected randomly.

VI. PROTOTYPICAL REALIZATION

This section outlines the main characteristics of our prototypical realization of RADAR.[10] We first give an overview of the software structure, then we describe the interfaces provided or required by RADAR. More information about the used technologies is given in the Appendix.

[10]The prototypical realization is publicly available from `https://git.uni-due.de/fogprotect/radar`

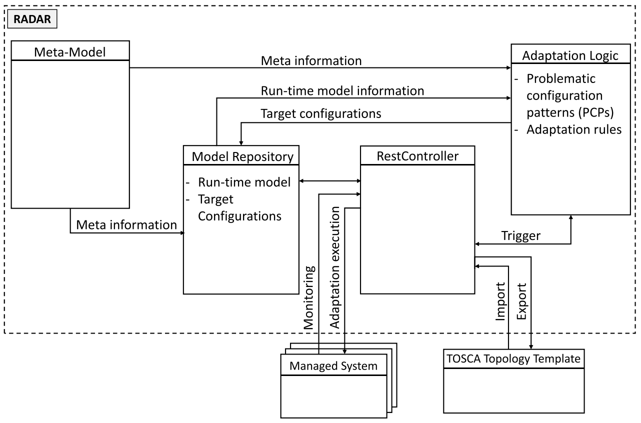


FIGURE 12. Main components of RADAR.

A. OVERVIEW

As shown in Figure 12, the prototypical realization of RADAR is structured into four components.

- The **Meta-Model** (see Section V-B) is based on the Eclipse Modeling Framework (EMF), a modeling framework and tool for creating Java source code based on a structured data model. Information from the Meta-Model is used by the Model Repository and the Adaptation Logic.
- The **Model Repository** contains models of specific system configurations, based on the EMF meta-model. The Model Repository contains two kinds of models:
  - The **run-time model** represents the current configuration of the managed systems at run time. After a managed system registers with RADAR, a representation of the managed system is added to the run-time model. The parts of the run-time model corresponding to individual managed systems are identified by unique IDs.
  - A **target configuration** represents the result of a possible sequence of adaptations that is being evaluated in terms of costs and functionality.
- The **RestController** offers RESTful interfaces between RADAR and other systems. This includes interfaces to monitor managed systems and to execute adaptations on managed systems, as well as an interface to import and export TOSCA topology templates (see Section VI-B2). In particular, changes in the configuration of managed systems are reflected through the monitoring interface, triggering the RestController to update the run-time model appropriately.
- The **Adaptation Logic** is responsible for determining if there is a need for adaptation, and if this is the case, for determining the adaptation to perform. More specifically, if the run-time model is updated, the Adaptation Logic searches for instances of the problematic configuration patterns in the run-time model. In case of a match, the Adaptation Logic generates possible target

configurations using the adaptation rules and evaluates them to find the best sequence of adaptations.

B. INTERFACES

This section contains details on the interfaces provided by the "RestController" (see Figure 12), which relate to the handling of managed systems and the TOSCA export/import. All of these interfaces are RESTful web services and are as such referred to by their URL paths.

1) HANDLING MANAGED SYSTEMS

Figure 13 gives an overview of our approach to monitoring and adaptation execution. In particular, the figure shows that some of the managed systems may be only monitored but not adapted, or only adapted but not monitored, or both monitored and adapted.

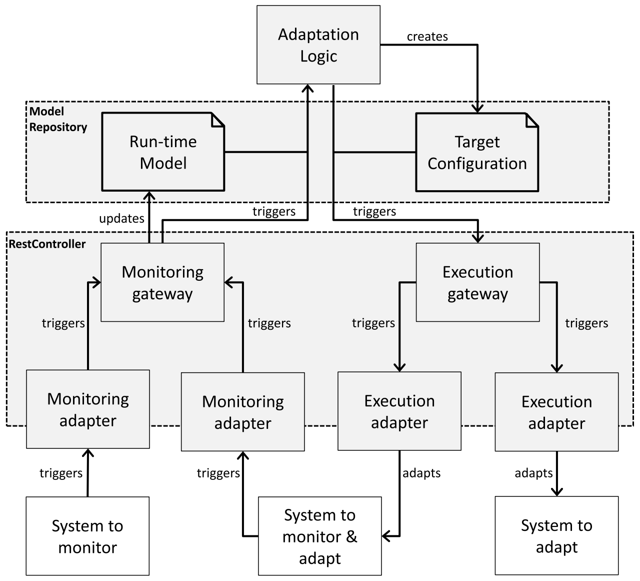


FIGURE 13. Monitoring and adaptation execution.

The monitoring functionality is divided into a general monitoring gateway and system-specific monitoring adapters. The *monitoring gateway* offers a common interface for updating the run-time model. Systems to be monitored will typically not use this interface directly; rather, a system-specific *monitoring adapter* is used to observe the given system and transform monitoring data gained from the system into the format required by the monitoring gateway. We distinguish two different ways of monitoring. In *event-driven monitoring*, the managed system updates the run-time model after any change by invoking the monitoring adapter. In *time-driven monitoring*, the monitoring adapter observes the managed system at certain time intervals to detect changes. Both possibilities can be used simultaneously.

Also the functionality to execute adaptations is divided into a general adaptation execution gateway and system-specific execution adapters. The *execution gateway* offers a generic (i.e., system-independent) interface for the adaptation logic to specify the adaptations that need to be carried out. This interface will typically not be implemented directly by the systems that need to actually execute the adaptations; rather, a system-specific *execution adapter* transforms the adaptations into calls to the API of the system to be adapted. The execution adapter is called by the execution gateway.

The following interfaces are provided by the RestController for handling managed systems:

- /registerManagedSystem
  This web service allows the registration of a new managed system with RADAR. The request body contains a JSON representation of the initial configuration of the new managed system and the URL address of the managed system's execution interface (see below). The web service returns the ID of the managed system within the run-time model in the response body, or -1 if the request failed for any reason.
- /updateRuntimeModel/{id}
  This web service allows the monitoring adapter to update the run-time model. The {id} in the path should be the ID of the managed system (as returned by "/registerManagedSystem"). The part of the run-time model belonging to this ID will be updated with the changed model supplied in the request body as a JSON object.

In addition, the following interface is provided by the execution adapters:

- /executeAdaptation
  This service allows the execution gateway to pass a selected target configuration to the appropriate execution adapter. The target configuration is transmitted as a JSON representation. The execution adapter may convert the information into a different format suitable for invoking the APIs of the managed system to execute the adaptation.

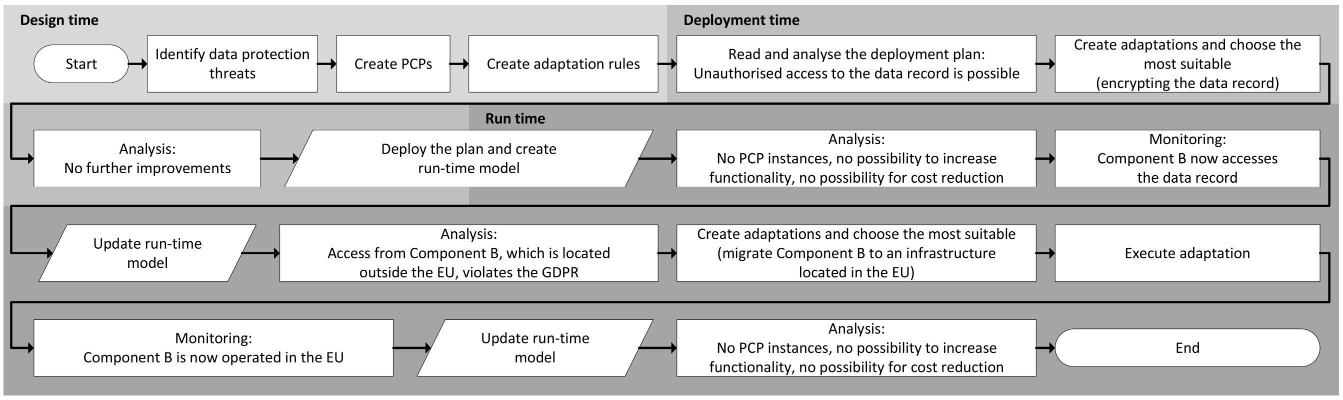
2) TOSCA EXPORT/IMPORT

RADAR provides the possibility to export the run-time model in TOSCA format and to import a TOSCA description into the run-time model. We implemented this functionality based on the TOSCA Simple Profile in YAML.[11] The TOSCA Simple Profile in YAML specifies a rendering of TOSCA in the YAML language.[12] A TOSCA topology template defines the structure of a service in the form of a graph [45].

a: IMPORTING FROM TOSCA INTO THE RUN-TIME MODEL

Since we already support importing JSON representations of models fitting the meta-model (see Section II-A), we only have to convert the topology templates from YAML to JSON. To import a given TOSCA topology template, we assume it is aligned with the RADAR meta-model. Topology templates that are not aligned with the RADAR meta-model can still

[11] http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csprd02/TOSCA-Simple-Profile-YAML-v1.0-csprd02.html
[12] YAML is a human-readable data serialization standard, see https://yaml.org/



Design time
Start
Identify data protection threats
Create PCPs
Create adaptation rules
Deployment time
Read and analyse the deployment plan: Unauthorised access to the data record is possible
Create adaptations and choose the most suitable (encrypting the data record)
Run time
Analysis: No further improvements
Deploy the plan and create run-time model
Analysis: No PCP instances, no possibility to increase functionality, no possibility for cost reduction
Monitoring: Component B now accesses the data record
Update run-time model
Analysis: Access from Component B, which is located outside the EU, violates the GDPR
Create adaptations and choose the most suitable (migrate Component B to an infrastructure located in the EU)
Execute adaptation
Monitoring: Component B is now operated in the EU
Update run-time model
Analysis: No PCP instances, no possibility to increase functionality, no possibility for cost reduction
End

FIGURE 14. **Steps in an exemplary sequence applying RADAR to the cloud example.**

be imported, but may not include all the possibilities offered by the RADAR meta-model. For demonstration purposes the TOSCA import / export interface currently supports a subset of the RADAR meta-model.

Our TOSCA-to-JSON translator transfers the information from the TOSCA Simple Profile in YAML into a graph consisting of JSON elements. To create a JSON string from a graph, the Gson library is used. Gson is an open-source Java serialization / deserialization library to convert Java objects to JSON and back.[13] When importing the topology, both attribute values and relations are transferred. In addition, each object is assigned an ID, which is required in the EMF run-time model. Unlike the TOSCA Simple Profile in YAML, EMF realizes relations using the assigned IDs. The transfer of attribute values and relationships is split into two steps. First an object is created for each element of the TOSCA topology. Furthermore, attribute values (e.g., instance name, type) are transferred and IDs are assigned. In a second run, the relations are transferred by making use of the generated IDs. Following this translation, the JSON object is imported into EMF, thereby creating an initial run-time model.

b: EXPORTING THE RUN-TIME MODEL TO TOSCA

When exporting the run-time model, the JSON string generated by EMF is converted into a graph by using Gson. Attribute values and relations are transferred into YAML format. Since no IDs are needed in TOSCA Simple Profile in YAML, they are not transferred. In this case, the relations can be transferred directly, without requiring a second run. The division into attributes and relations is restored. It is possible to export only the part of the run-time model corresponding to a managed system by specifying its ID. Moreover, it is also possible to export target configurations into a TOSCA topology template the same way.

VII. EVALUATION

In this section, we investigate the practical applicability of the RADAR approach. In Section VII-A, we demonstrate how RADAR can be applied to complex, real-world distributed systems in cloud and fog computing. In Section VII-B, we present empirical results on the performance of RADAR, while Section VII-C demonstrates the use of RADAR over a period of time.

A. CASE STUDY

We applied RADAR to the cloud and fog examples introduced in Section II. In both cases, the meta-model based on TOSCA presented in Section V-B was used.

1) CLOUD EXAMPLE

Figure 14 shows a possible sequence of steps that RADAR performs in the cloud example. At **design time**, PCPs are identified by a data protection assessment carried out by experts. The identified PCPs are modeled using the PCP language. To address the PCPs, adaptation rules are defined with the RADAR adaptation language.

At **deployment time**, a deployment plan is created using TOSCA. Until this point, each step is carried out manually. From this moment, however, the process is automated by RADAR. The deployment plan is imported by RADAR and turned into an initial run-time model. For the cloud example presented in Section II-A, the initial run-time model is the one shown in Figure 10. RADAR analyses this initial run-time model and finds PCP instances capturing the possibility that the untrusted actors "IaaS Operator W," "IaaS Operator X" and "SaaS Operator Z" might gain unauthorised access to sensitive data. For example, a match of the PCP shown in Figure 6 is identified: the data record is not encrypted and contains sensitive data belonging to the Data Subject. Furthermore, the data subject does not trust "IaaS Operator W" who operates the infrastructure on which the database is hosted. Due to the fact that the untrusted IaaS operator might be able to access the unencrypted data, there is a potential data protection violation. The designers may have overlooked this problematic configuration, but RADAR detects the threat.

To address the found PCP instances, RADAR considers applying adaptation rules to reconfigure the initial run-time

[13]https://github.com/google/gson

model. In the particular case, RADAR considers the following possibilities:

- Migrating the database to a compute node operated by a trusted operator. This adaptation rule is not applicable in the given situation, since its pre-condition is not satisfied (there is no other IaaS operator that the Data Subject would trust).
- Terminating the storage of the data in the database. While this adaptation is applicable and would solve the identified data protection threats, it would result in a significant loss of functionality (the components that need access to the database would be limited in their functionality).
- Switching on encryption of the data records. This adaptation is applicable and solves the identified data protection threats without limiting the available functionality.

Hence, RADAR performs a reconfiguration to encrypt the data records. The initial run-time model is adjusted and then re-examined by RADAR for PCP instances and possibilities for functionality improvements or cost reductions. Since neither PCP instances nor improvement possibilities are found, the system is deployed according to the updated initial run-time model.

During **run time**, the run-time model is kept up-to-date by means of monitoring. In the considered scenario, the system changes and the run-time model is updated as shown in Figure 15. A new component B, which is hosted on a compute node located in the US, accesses the DBMS. This access violates the GDPR and is recognized as an instance of the PCP shown in Figure 7. The PCP instance is highlighted in Figure 15.

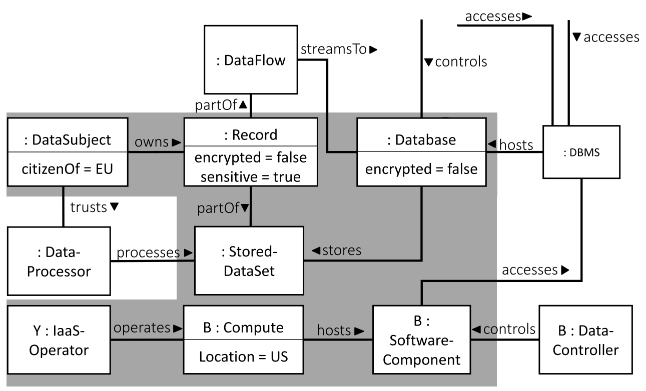


FIGURE 15. Excerpt of the example run-time model after a change in the system. (The identified instance of PCP B (see Figure 7) is highlighted).

To mitigate the found PCP instance, RADAR considers the following possible adaptations:

- Component B's access to the database is disabled. This adaptation is applicable and would resolve the PCP instance, but it would imply a restriction of functionality of component B.
- Component B is migrated to an infrastructure located in the EU. In the considered scenario, two appropriate compute nodes are available in the EU, so that this adaptation is also applicable. This adaptation also resolves the PCP instance and does not reduce the available functionality.

RADAR therefore chooses the second adaptation possibility. It compares the usage costs of both compute nodes, selects the one that incurs less costs and executes the migration.

When the migration has been carried out, monitoring captures the change and updates the run-time model. RADAR checks the updated run-time model again for data protection violations. As there were no further changes to the system beside the adaptation, no PCP instances are identified in the updated run-time model. Furthermore, no possibility of functionality improvement or cost reduction can be identified, meaning that the system is operating in an optimal state.

2) FOG EXAMPLE

A similar sequence of steps as described above for the cloud example is also performed in the fog example scenario. First, PCPs are identified and modeled using the PCP language at **design time**. Afterwards, the adaptation rules are defined with the RADAR adaptation language. At **deployment time**, the deployment plan of the example fog system shown in Figure 2 is imported, turned into a run-time model and analysed by RADAR. Neither PCP instances nor improvement possibilities are found.

In this particular scenario, the system changes during **run time** several times. The changes in the system are detected by RADAR and the run-time model is updated accordingly. Initially, the system processes no personal data. After a change to lot-size-one production, the customer data processed by the order management component and stored in a database hosted on the cloud is now classified as sensitive. Storing and processing sensitive data on a public cloud violates data protection constraints. The violation is recognised as a PCP instance. Since it is essential to process the customer data, a migration of the order management component and the customer data to the fog node is considered as a possible adaptation. This adaptation restricts the functionality of the order management component due to the computing power of the fog node, but the adaptation has to be carried out to prevent the violation of data protection. Since no further changes are found in a second monitoring step apart from the adaptation carried out, no further PCP instances are found in the run-time model. Furthermore, there is no possibility to improve the system by restoring functionality without violating data protection constraints.

After some time, lot-size-one production ends, and thus customer data are no longer classified as sensitive. The run-time model is updated accordingly, and checked for PCP instances. No matches are found. Subsequently, a possibility to improve functionality by migrating the order management component and customer data to the cloud is found, since the processing and storing of the data no longer violates data protection constraints. Accordingly, this adaptation is carried out and the run-time model is updated. As before, no PCP instances are found. Moreover, no further possibility to improve functionality is found, since the system is working in an optimal configuration.

B. PERFORMANCE MEASUREMENTS

RADAR contains some computationally intensive steps, in particular the search for PCP instances in the run-time model and the search for the best adaptation sequence in the given situation. The practical applicability of RADAR could be potentially hampered if these steps take too long. Hence we investigate the performance and scalability of RADAR by means of controlled experiments.

1) EXPERIMENT SETUP

As a basis for the performance tests, we use the run-time model of the cloud example (Section VII-A1, Figure 10). The run-time model contains two PCP instances, namely one instance of PCP A (Figure 6) and one of PCP B (Figure 7). To increase the variety of options for the adaptation algorithm, the run-time model is extended with three additional nodes. Those nodes represent an IaaS operator and two compute nodes. These nodes are not connected to the other nodes shown in Figure 10. The resulting run-time model is called the *basic run-time model*. The basic run-time model is a realistic model with known PCP instances; hence it is a good basis for the controlled experiments.

To investigate the scalability of RADAR, we enlarge the run-time model. Since the model enlargement method may impact the structure of the model, which may in turn influence the performance of RADAR, we experiment with two different methods for model enlargement. The first method uses k copies of the basic run-time model. These copies are interconnected through the shared DataSubject node. The second method adds a defined number of randomly selected nodes and edges to the basic run-time model.

The basic run-time model contains 22 nodes, 23 edges, and 2 PCP instances. The first method produces a run-time model with $1 + 21 \cdot k$ nodes, $23 \cdot k$ edges, and $2 \cdot k$ PCP instances. k is the number of copies of the basic run-time model, which is used as an independent variable to control the size of the run-time model. Since the copies share the DataSubject node, each replication adds only 21 instead of 22 nodes.

The second method produces a run-time model with $22+n$ nodes, $23+m$ edges, and $2+z$ PCP instances, where n represents the number of added nodes, m represents the number of added edges and z represents the number of additional PCP instances. The number n of added nodes is used as independent variable to control the size of the run-time model, while the number m of added edges is random. Relations that must exist according to the meta-model are set in any case. For example, a SoftwareComponent must be hosted on a Compute node. For this reason, the addition of a node may require the addition of some further nodes. If this happens with the nth added node, then the number of nodes actually added will be slightly higher than the originally intended number n. After adding the nodes and the necessary relations, the enlargement algorithm checks which additional relations are possible. Based on this, the enlargement algorithm determines how many additional relations are possible in the currently given run-time model (*max_rels*). Next, a uniform random number between 1 and *max_rels* is chosen, and the chosen number of randomly selected relations are inserted. Through this process, some additional PCP instances might be created; thus, z is not known a priori.

The number of PCPs is fixed to 2, and there are 3 adaptation rules to mitigate each of the PCPs, yielding 6 adaptation rules altogether.

One of the adaptation rules (Figure 9) consists of migrating a database from one DBMS to another one hosted on a different compute node. After enlarging the run-time model with one of the methods explained above, it is possible to migrate a database to a DBMS hosted on a compute node, which is either in another copy of the model (enlargement method one) or a newly added node (enlargement method two). Hence, as the number of nodes increases, also the number of possible adaptations increases. Furthermore, such adaptations significantly change the structure of the run-time model by connecting two previously unconnected parts of the run-time model.

The performance tests were run on a Fujitsu Celsius w550 workstation with an Intel Xeon E3-1275v5 processor with 3.6 GHz clock frequency and with 64 GB DDR4 memory. The machine was running Ubuntu 16.04 with kernel version 4.10.0-42-generic as operating system, and OpenJDK 64-Bit Server VM (build 14.000.1+7) as Java Virtual Machine.

2) RESULTS

We performed one adaptation loop, consisting of searching for PCP instances in the run-time model and then searching for the best sequence of adaptations to mitigate the found PCP instances. The time for the adaptation loop is capped at 10 seconds, i.e., the best found adaptation sequence is returned after 10 seconds. We performed this test for run-time models of increasing size, and with four different algorithms (best-first search, depth-first search, breadth-first search, random search) for searching for the best sequence of adaptations. Each test (i.e., each combination of run-time model size and algorithm) was performed 100 times, and the average results are reported. Since the order of the available adaptation rules may affect the results of the search algorithms, the adaptation rules were ordered randomly for each test.

Table 4 and Table 5 show the results of the experiments. In both tables, the first column displays the number of nodes of the run-time model. The remaining columns each refer to one algorithm and display a penalty value, characterizing the quality of the best solution found by the algorithm. The penalty is calculated by the formula $100 \cdot a + 10 \cdot b + 1 \cdot c$, where a is the number of PCP instances, b is the increase in the number of functionality restrictions, and c is the increase in costs in the found solution. Note that b and c are relative to the run-time model before the adaptation loop, and can also be negative if the adaptation led to an improvement. For all of a, b, c and the penalty, lower numbers are better. The weights in the penalty function correspond to the relative importance of the three considered metrics. The goal of the algorithms is to find an adaptation sequence leading to the lowest penalty. The lowest penalty in each row is highlighted. Furthermore, solutions that do not mitigate all PCP instances ($a > 0$) are marked with an asterisk (*).

TABLE 4. Penalty of the solution achieved by the tested search algorithms for run-time models of increasing size in the first scalability experiment, with a timeout of 10 seconds. Smaller numbers are better. In each row, the minimum is highlighted. Solutions that do not mitigate all PCP instances are marked with an asterisk (*).

Nr. of nodes	Best-First	Depth-First	Breadth-First	Random
22	0.00	0.00	0.00	0.00
43	-0.55	-0.55	-0.55	-0.55
64	-1.10	2.63	299.33*	1.43
85	-1.65	7.63	499.44*	149.99*
106	-2.21	9.88	739.74*	395.08*
127	-2.76	13.35	999.45*	636.47*
148	-3.31	22.84	1199.45*	863.15*
169	104.73*	24.50	1399.57*	1118.30*
190	662.15*	23.07	1599.93*	1347.62*
211	1081.98*	316.42*	1800.00*	1564.36*

The results of the performance test in which the first enlargement method was used are shown in Table 4. For the smallest run-time models, all search algorithms lead to the same good results, but starting from 64 nodes, the results are quite different. The best-first-search algorithm provides the lowest penalty up to a run-time model size of 148 nodes. The negative penalty values show that best-first search could not only mitigate all found PCP instances, but at the same time also the availability of functionalities and/or the costs could be improved. For run-time models that consist of more than 148 nodes, the depth-first-search algorithm provides the lowest penalty. Breadth-first-search always leads to the highest penalty. The random search algorithm is able to provide better results than breadth-first-search, but is clearly outperformed by best-first-search and depth-first-search.

The results of the performance test in which the second enlargement method was used are shown in Table 5. Recall from Section VII-B1 that the second enlargement method does not always add exactly the same number of nodes; therefore, the average number of nodes in Table 5 is non-integer. Again, the breadth-first-search algorithm leads to the highest penalty. Random search leads to acceptable results until 122 nodes, but its performance quickly deteriorates afterwards. Best-first-search provides the lowest penalty in most cases up to a model size of 222 nodes (in most cases again with negative penalty values), while the depth-first-search algorithm leads to a slightly higher penalty. For models with more than 222 nodes, depth-first-search provides the lowest penalty.

The two experiments provide similar insights about the performance of the algorithms. The breadth-first-search algorithm performs worst, because it processes all possibilities

TABLE 5. Results of the second scalability experiment; notation as in Table 4.

Nr. of nodes	Best-First	Depth-First	Breadth-First	Random
22.00	0.00	0.00	0.00	0.00
42.09	-0.49	-0.49	15.51	-0.49
62.09	-0.57	-0.53	183.45*	-0.53
82.09	-0.77	-0.63	414.29*	-0.73
102.11	-0.76	0.15	714.44*	-0.47
122.08	-0.79	0.71	782.49*	-0.63
142.07	-0.77	2.15	966.63*	15.84
162.09	-0.82	2.64	1132.69*	29.36
182.14	23.16	3.01	1205.65*	13.71
202.08	-0.82	3.00	1502.72*	245.36*
222.08	-0.81	3.24	1629.73*	337.18*
242.10	122.18*	30.86	1733.71*	442.25*
262.11	230.18*	59.56	1963.67*	1001.93*
282.07	535.29*	302.10*	2012.71*	961.98*

on the first level of the search tree before it continues on the second level, and so on. Even in run-time models of moderate size, 10 seconds are not enough to reach a level where all PCP instances are mitigated. Compared to the other algorithms, breadth-first search also has the highest memory consumption, since it has to keep an exponential number of models in memory. Random search performs somewhat better, but for bigger models it is consistently outperformed by the more target-oriented algorithms. Depth-first-search is the most efficient way to go deep enough in the search tree for finding a solution that mitigates all PCP instances, but the solutions found this way may be far from optimal in terms of functionality and cost. Using its more sophisticated search strategy that explicitly considers all relevant metrics, best-first-search consistently finds the best solution among the four algorithms, not only mitigating all PCP instances but also improving functionality and/or costs, up to a certain run-time model size. For the largest run-time models, the 10 seconds are no longer sufficient for best-first search to mitigate all PCP instances, because of the additional time required for assessing different adaptation possibilities.

Comparing the two experiments, it is apparent that the first enlargement method leads to more difficult problem instances. Indeed, for each algorithm, the point from which the algorithm cannot mitigate all PCP instances is reached sooner (i.e., with models of smaller size) with the first enlargement method than with the second one. This is probably due to the fact that, for models of equal size, the first enlargement method leads to a higher number of PCP instances than the second. More PCP instances mean that a longer path needs to be found in the search tree, which makes the problem more difficult for all search algorithms. It can also be seen though that the best-first search and depth-first search algorithms can better cope with this increase in complexity than the other two algorithms.

From the experiments we can conclude that RADAR, using either the best-first search or the depth-first search algorithm, can quickly find a good sequence of adaptations to mitigate multiple data protection threats in large run-time models.

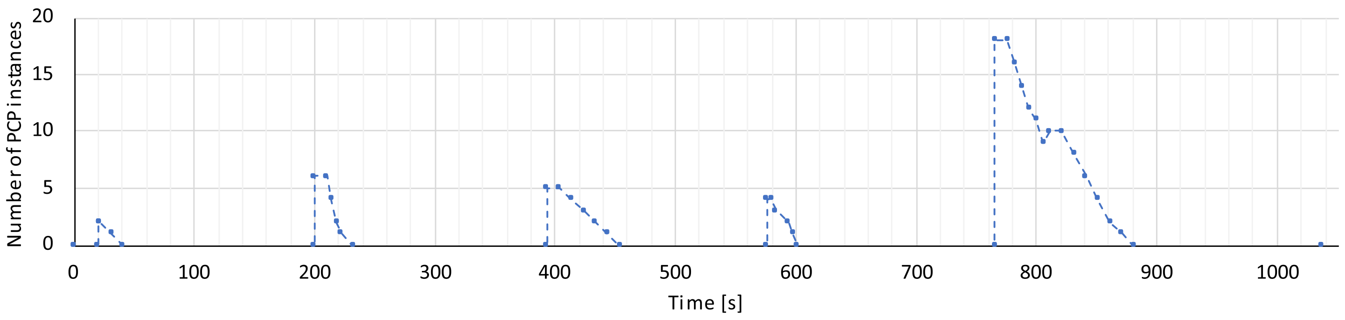


FIGURE 16. **Applying RADAR over a period of time.**

For example, the depth-first-search algorithm could find a solution to mitigate all the 18 PCP instances in a run-time model with 190 nodes in only 10 seconds.

C. EFFECTS OF THE USAGE OF RADAR OVER TIME

In a further experiment, we analyze how RADAR deals with PCP instances that occur at run time, i.e., how RADAR detects PCP instances and resolves them by adapting the system. For this purpose, we use an initial run-time model resulting from applying the first model enlargement method described in Section VII-B, and applying RADAR initially, so that no PCP instances are present in the model. The resulting model consists of 337 nodes.

Starting from this initial run-time model, we perform a sequence of changes. The changes happen at random points in time. Each change introduces a randomly chosen number of threats in the range from 2 to 20. After each change, RADAR tries to detect and mitigate the introduced risks.

Figure 16 shows how the number of PCP instances in the run-time model evolves over time. As can be seen, after each change, RADAR successfully detects and mitigates the introduced PCP instances until the number of PCP instances returns to 0. The detection of the PCP instances takes less than a second in each case. If there are few PCP instances (less than five PCP instances for the given model size of 337 nodes) in the run-time model, RADAR is able to select the best adaptation sequence in less than five seconds. Otherwise, the search for the best adaptation sequence is interrupted by the timeout of 10 seconds. Even in such cases, RADAR manages to identify an adaptation sequence that is useful in mitigating the threats. After the sequence of adaptations has been selected, the execution of the adaptations starts. The duration of the execution depends on the number and kind of the adaptations to be performed. Moreover, sometimes it is possible to execute multiple adaptations simultaneously to speed up the execution process.

If the run-time model contains too many PCP instances at once, then it can happen that RADAR does not find within the 10 seconds time budget an adaptation sequence that would mitigate all PCP instances. For a model size of 337 nodes, this happens if the number of PCP instances is 15 or more. Even in this case, RADAR selects an adaptation sequence that mitigates as many PCP instances as possible. Afterwards, RADAR starts another search for adaptations until all PCP instances are mitigated. This behavior can be observed in Figure 16 around 800 seconds.

When all adaptations have been executed, the number of PCP instances returns to 0 in each case. Thus, RADAR can ensure long-term data protection even in the face of continuously emerging threats.

VIII. DISCUSSION

This section describes some further considerations, such as to which extent RADAR meets the defined requirements and the threats to the validity of our empirical findings.

A. SATISFACTION OF THE REQUIREMENTS

We defined six requirements in Section III that an effective run-time data protection approach should fulfill.

RADAR provides a language for the definition of problematic configuration patterns (Section V-C) and a language for the definition of adaptation rules (Section V-D), thus fulfilling requirements **R1** and **R2**. As described in Section V-F, RADAR automatically detects PCP instances, and Section V-H shows how RADAR mitigates the found PCP instances automatically using adaptations, thus fulfilling requirements **R3** and **R4**. RADAR takes functionality and costs into consideration to find the best adaptation if there is more than one possibility to mitigate a data protection risk (see Section V-K), thus fulfilling requirement **R5**. Section V-K also shows how adaptations for other reasons (e.g., to minimize costs) are supported by RADAR, while also ensuring the continued compliance with data protection, thus fulfilling requirement **R6**. Finally, the case studies of Section VII-A demonstrate how all this works in practice.

Thus, RADAR fully supports the defined requirements.

B. RELATION TO EXISTING CLOUD SERVICES

In our earlier work, we analyzed the data protection impact of vulnerabilities described in the cloud literature [38]. We modeled the resulting threats to data protection in the form of PCPs. The covered vulnerabilities included potential signature wrapping attacks on the control interfaces of Amazon Web Services, side-channel attacks between colocated

Amazon EC2 instances, and privilege exploitation attacks. Thus, that study showed that PCPs can be used to capture a variety of real-world data protection threats in existing cloud services.

RADAR may be implemented on top of existing cloud services, or could be integrated into a cloud service. In any case, it is important that the relevant cloud services offer appropriate monitoring and adaptation capabilities to facilitate the fulfillment of requirements R3 and R4, respectively. Existing cloud services typically offer such interfaces [51], [52]. Section VI-B1 shows how RADAR can be connected to the monitoring and adaptation interfaces of cloud services.

C. THE SECURITY OF RADAR

RADAR ensures the protection of confidential data by detecting and mitigating configurations that would threaten data protection. However, RADAR also introduces an additional attack surface. For example, an attacker may try to compromise the RADAR adaptation logic, or may try to perform a man-in-the-middle attack on the connection between RADAR and the adapted systems. If such an attack succeeds, the attacker may be able to overtake control over the cloud system and thus gain access to the processed data.

To avoid such attacks, RADAR should be protected by standard security measures. For example, the RADAR adaptation logic may be run in a trusted execution environment, and the connections between RADAR and the adapted systems may be protected by authentication, encryption, and digital signatures.

In contrast to the managed cloud services, RADAR is not supposed to change frequently during run time. This is why standard security techniques are appropriate for protecting RADAR, without the need to recursively apply a RADAR-like adaptive approach to protect RADAR itself.

D. RELATION TO SECURITY GOALS

Data protection is transversal to traditional security goals like confidentiality, integrity, and availability [53]. Indeed, data protection contains aspects of confidentiality, integrity and availability, but also other aspects, such as privacy [54]. For example, the GDPR requires that personal data be protected from unauthorized access (confidentiality) and that the correctness of data has to be ensured (which requires integrity), but it also poses different types of requirements, such as geo-location limitations, which go beyond traditional security goals.

Hence, also RADAR is not limited to some of the traditional security goals. RADAR can capture problematic configurations that would threaten data protection. Problematic configurations may or may not relate to some of the traditional security goals. The design-time process of identifying PCPs should take into account the traditional security goals of confidentiality, integrity, and availability, but also all other relevant requirements of applicable data protection laws and policies.

E. THREATS TO VALIDITY

The *internal validity* of our experiments may be influenced by the order in which the adaptation rules are provided to the adaptation planning algorithm. To minimize the bias caused by the order of adaptation rules, the adaptation rules were ordered randomly for each test. Also other random effects (such as an algorithm immediately finding the best adaptation sequence by chance) could threaten internal validity. To address this, we performed the performance experiments (cf. Section VII-B) 100 times.

The *external validity* is influenced by the selection of the example systems to which we apply our approach. To ensure a realistic setting for our evaluation, we applied our approach to two examples from different domains defined in cooperation with industry partners. Another threat to external validity is a potential influence of the results by the size of the run-time model. To avoid bias caused by run-time models with an unrealistically small number of nodes, we enlarged the run-time model by additional nodes, using two different methods.

IX. CONCLUSION AND FUTURE WORK

This paper presented RADAR, a novel approach for ensuring the continued protection of sensitive data in dynamically changing cloud and fog systems. RADAR uses self-adaptations to automatically react to changes during run time. RADAR chooses the adaptation that ensures data protection with the least negative impact on other system properties like costs and the available functionalities.

To show its practical applicability, we applied RADAR to two realistic case studies in the domains of cloud computing and fog computing. In addition, we experimented with four different algorithms for the computationally most challenging part of RADAR, the search for the best adaptation sequence. Our experiments demonstrated that with the two best-performing search algorithms RADAR can resolve multiple data protection threats in run-time models with hundreds of nodes within ten seconds. Thus, RADAR can be applied to complex IT systems and ensure data protection in such systems on the fly.

There are multiple promising directions for future research. On the one hand, RADAR could be extended so that more complex problematic configurations (e.g., threats arising from a sequence of patterns) can be captured. On the other hand, further algorithmic enhancements could be devised to improve the performance of RADAR (e.g., by using incremental pattern matching algorithms). Also, the adaptation logic of RADAR could be extended so that it can reason about the time and computational cost of searching for the best adaptation, as suggested in [55]. This way, urgent data protection threats could be mitigated more quickly than other threats that do not require such an urgent reaction.

APPENDIX I. FORMALIZATION

This section provides a formal description of RADAR. Table 6 gives an overview of the used notation.

TABLE 6. **Overview of formal notation.**

Notation	Description
B	Set of basic types
Meta-model	
M^{meta}	Meta-model
T	Set of types
n_t	Name of type t
A_t	Set of attributes of type t
n_a	Name of attribute a
t_a	Type of attribute a
R	Set of associations
n_r	Name of association r
s_r	Source type of association r
g_r	Target type of association r
I	Direct inheritance relation
I^*	Direct or indirect inheritance relation
Problematic configuration patterns	
P	Set of PCPs
O_p	Set of objects in PCP p
n_o^p	Name of the type of object o in PCP p
t_o^p	Type of object o in PCP p
A_o^p	Set of attribute descriptions of object o in PCP p
n_a^p	Attribute name in attribute description a of PCP p
v_a^p	Attribute value in attribute description a of PCP p
R_p	Set of relation descriptions in PCP p
s_r^p	Source object of relation description r in PCP p
g_r^p	Target object of relation description r in PCP p
n_r^p	Relation name in relation description r of PCP p
Adaptation rules	
Ad	Set of adaptation rules
O_{ad}	Set of objects in adaptation rule ad
n_o^{ad}	Name of the type of object o in adaptation rule ad
t_o^{ad}	Type of object o in adaptation rule ad
A_o^{ad}	Set of attribute descriptions of object o in adaptation rule ad
n_a^{ad}	Attribute name in attribute description a of adaptation rule ad
v_a^{ad}	Attribute value in attribute description a of adaptation rule ad
R_{ad}	Set of relation descriptions in adaptation rule ad
s_r^{ad}	Source object of relation description r in adaptation rule ad
g_r^{ad}	Target object of relation description r in adaptation rule ad
n_r^{ad}	Relation name in relation description r of adaptation rule ad
ann_r^{ad}	Annotation in relation description r of adaptation rule ad
PCP_{ad}	PCP mitigated by adaptation rule ad
Pre_{ad}	Precondition of adaptation rule ad
Act_{ad}	Adaptation action of adaptation rule ad
Run-time model	
M^{RT}	Current run-time model
O^{RT}	Set of objects
n_o^{RT}	Name of object o
t_o^{RT}	Type of object o
A_o^{RT}	Set of attributes of object o
n_a^{RT}	Name of attribute a
v_a^{RT}	Value of attribute a
R^{RT}	Set of relations
s_r^{RT}	Source object of relation r
g_r^{RT}	Target object of relation r
n_r^{RT}	Name of relation r

A. META-MODEL

The meta-model is given by the following tuple: $M^{\text{meta}} = (T, R, I)$, where T is the set of types, R is the set of associations between types, and I is the direct inheritance relation between types.

Each type $t \in T$ has a name and a set of attributes: $t = (n_t, A_t)$. Here, n_t is a string, that is, $n_t \in \Sigma^+$. Each attribute $a \in A_t$ is characterized by a name and a type: $a = (n_a, t_a)$, where $n_a \in \Sigma^+$ and $t_a \in B$, where B is the set of basic types.

Each association $r \in R$ is characterized by a name, a source type and a target type: $r = (n_r, s_r, g_r)$. Here, $n_r \in \Sigma^+$, and $s_r, g_r \in T$.

Finally, the direct inheritance relation is $I \subseteq T \times T$. A pair $(t_1, t_2) \in I$ means that type t_1 inherits directly from type t_2.

Based on I, we can define its transitive closure $I^* \subseteq T \times T$ as follows: $(t_1, t_k) \in I^*$ if and only if there is a sequence of types $t_1, t_2, \ldots, t_k \in T$ (where k is an arbitrary positive integer) such that for each $1 \le j \le k-1$, $(t_j, t_{j+1}) \in I$. This means that t_1 directly or indirectly inherits from t_k. For any $t \in T$, $(t, t) \in I^*$ by definition.

B. PROBLEMATIC CONFIGURATION PATTERNS (PCPs)

Let P be the set of all problematic configuration patterns, then $p \in P$ is defined as the pair $p = (O_p, R_p)$. The elements of p are as follows:

- O_p is the set of objects in p. Each $o \in O_p$ is an object, associated with a type name and a – possibly empty – set of attribute descriptions: $o = (n_o^p, A_o^p)$.
  - The type name must be the name of a type in the meta-model, i.e., $n_o^p = n_t$ for some $t \in T$. Let this type t be denoted as t_o^p.
  - If $a \in A_o^p$ is an attribute description, then either $a = (n_a^p, \text{“==”}, v_a^p)$ or $a = (n_a^p, \text{“!=”}, v_a^p)$. In both cases, n_a^p is the name of an attribute in the type t_o^p or one of its super-types, i.e., $n_a^p = n_{a'}$ for some $a' \in A_{t'}$ with $I^*(t_o^p, t')$, and v_a^p is a possible value of that attribute.
- R_p is the set of relation descriptions in p. If $r \in R_p$, then either $r = (s_r^p, g_r^p, n_r^p)$ or $r = (s_r^p, g_r^p, n_r^p, \text{“}$`<<does not exist>>`$\text{”})$, where s_r^p, g_r^p and n_r^p denote the source object, target object, and name of the relation, respectively. In both cases, there must be a relation $r' \in R$ with the same name between the types of the involved objects or their super-types, i.e., $n_{r'} = n_r^p$, $I^*(t_{s_r^p}, s_{r'})$ and $I^*(t_{g_r^p}, g_{r'})$.

C. ADAPTATION RULES

The set of adaptation rules is given by Ad. An adaptation rule $ad \in Ad$ is given by the tuple $ad = (O_{ad}, R_{ad}, PCP_{ad}, Pre_{ad}, Act_{ad})$. Here, O_{ad} is the set of objects in the adaptation rule, R_{ad} is the set of relation descriptions in the adaptation rule, $PCP_{ad} \in P$ is the PCP that ad mitigates, Pre_{ad} is the precondition, and Act_{ad} is the adaptation action.

Let $o \in O_{ad}$ be an object. One possibility is that, as in the PCP language, $o = (n_o^{ad}, A_o^{ad})$, where n_o^{ad} is the type name of o and A_o^{ad} is a set of attribute descriptions, and the type of o is denoted as t_o^{ad}. In addition, the object can be assigned an annotation, so that $o = (n_o^{ad}, A_o^{ad}$, `<<create>>`$)$ or $o = (n_o^{ad}, A_o^{ad}$, `<<delete>>`$)$. Each object $o \in O_{ad}$ belongs to at least one of $PCP_{ad}, Pre_{ad}, Act_{ad}$.

Let $a \in A_o^{ad}$ be an attribute description, then a belongs to exactly one of $PCP_{ad}, Pre_{ad}, Act_{ad}$. If a belongs to either PCP_{ad} or Pre_{ad}, then a has one of the forms defined in the PCP language. If, on the other hand, a belongs to Act_{ad}, then

a has the form $(n_a^{ad}, \text{“=”}, v_a^{ad})$. Here, n_a^{ad} is the name of an attribute in the type t_o^{ad} or one of its super-types, i.e., $n_a^{ad} = n_{a'}$ for some $a' \in A_{t'}$ with $I^*(t_o^{ad}, t')$, and v_a^{ad} is a possible value of that attribute.

Let $r \in R_{ad}$ be a relation description, then r belongs to exactly one of $PCP_{ad}, Pre_{ad}, Act_{ad}$. If r belongs to either PCP_{ad} or Pre_{ad}, then r has one of the forms defined in the PCP language. If, on the other hand, r belongs to Act_{ad}, then r has the form $r = (s_r^{ad}, g_r^{ad}, n_r^{ad}, ann_r^{ad})$. Here, s_r^{ad}, g_r^{ad} and n_r^{ad} denote the source object, target object, and name of the relation, respectively. The annotation ann_r^{ad} is either “`<<create>>`” or “`<<delete>>`.”

D. RUN-TIME MODEL

Formally, the run-time model is defined as $M^{\mathrm{RT}} = (O^{\mathrm{RT}}, R^{\mathrm{RT}})$. The elements of M^{RT} are as follows:

- O^{RT} is the set of objects in the run-time model. Each $o \in O^{\mathrm{RT}}$ is an object, associated with a name, a type name and a set of attributes: $o = (n_o^{\mathrm{RT}}, t_o^{\mathrm{RT}}, A_o^{\mathrm{RT}})$.
  - The name is an arbitrary string: $n_o^{\mathrm{RT}} \in \Sigma^*$.
  - The type of the object is a type in the meta-model: $t_o^{\mathrm{RT}} \in T$.
  - For an attribute $a \in A_o^{\mathrm{RT}}$, $a = (n_a^{\mathrm{RT}}, v_a^{\mathrm{RT}})$. Here, n_a^{RT} is the name of an attribute in the type t_o^{RT} or one of its super-types, i.e., $n_a^{\mathrm{RT}} = n_{a'}$ for some $a' \in A_{t'}$ with $I^*(t_o^{\mathrm{RT}}, t')$, and v_a^{RT} is a possible value of that attribute.
- R^{RT} is the set of relations in the run-time model. If $r \in R^{\mathrm{RT}}$, then $r = (s_r^{\mathrm{RT}}, g_r^{\mathrm{RT}}, n_r^{\mathrm{RT}})$, where $s_r^{\mathrm{RT}}, g_r^{\mathrm{RT}} \in O^{\mathrm{RT}}$ denote the source and target object of the relation and $n_r^{\mathrm{RT}} \in \Sigma^+$ denotes the name of the relation. There must be a relation $r' \in R$ in the meta-model with the same name between the types of the involved objects or their super-types, i.e., $n_{r'} = n_r^{\mathrm{RT}}$, $I^*(t_{s_r^{\mathrm{RT}}}, s_{r'})$ and $I^*(t_{g_r^{\mathrm{RT}}}, g_{r'})$.

E. PROBLEMATIC CONFIGURATION IDENTIFICATION

A subset of objects $O \subseteq O^{\mathrm{RT}}$ in the run-time model matches a PCP $p \in P$, if and only if all of the following conditions are met:

- Each object appearing in the PCP must also exist in the run-time model. More precisely, for each object in the PCP, there must be a corresponding object in the run-time model belonging to the same type as or a (direct or indirect) sub-type of the object in the PCP. That is, there is an injective[14] mapping $f : O_p \to O$, for which $\forall o \in O_p : I^*(t_{f(o)}^{\mathrm{RT}}, t_o^p)$.
- All attributes appearing in the PCP must have appropriate values in the run-time model. If $o \in O_p$ and $a \in A_o^p$ is an attribute description in the PCP of the form $a = (n_a^p, \text{“==”}, v_a^p)$, then the attribute of the object $f(o)$ with the name n_a^p must have the value v_a^p in the run-time model: $a' \in A_{f(o)}^{\mathrm{RT}}, n_{a'}^{\mathrm{RT}} = n_a^p \Rightarrow v_{a'}^{\mathrm{RT}} = v_a^p$. On the other hand, if a has the form $a = (n_a^p, \text{“!=”}, v_a^p)$, then the corresponding attribute of $f(o)$ in the run-time model must have a different value: $a' \in A_{f(o)}^{\mathrm{RT}}, n_{a'}^{\mathrm{RT}} = n_a^p \Rightarrow v_{a'}^{\mathrm{RT}} \neq v_a^p$.
- Relations appearing without annotation in the PCP must be also present in the run-time model. That is, if $r \in R_p$ has the form $r = (s_r^p, g_r^p, n_r^p)$, then there must be a relation $r' = (f(s_r^p), f(g_r^p), n_r^p) \in R^{\mathrm{RT}}$ in the run-time model. On the other hand, relations appearing with the `<<must not exist>>` annotation in the PCP must not appear in the run-time model. That is, if $r \in R_p$ has the form $r = (s_r^p, g_r^p, n_r^p, \texttt{<<must not exist>>})$, then $(f(s_r^p), f(g_r^p), n_r^p) \notin R^{\mathrm{RT}}$ must hold.

[14] A mapping is called injective if the images of distinct elements are also distinct.

F. ADAPTATIONS

Suppose that a part of the run-time model $O \subseteq O^{\mathrm{RT}}$ has been identified as an instance of PCP $p \in P$, and $f : O_p \to O$ denotes the corresponding injective mapping, as described above. Moreover, suppose that there is an adaptation rule $ad \in Ad$ which mitigates the given PCP, that is, $PCP_{ad} = p$.

Adaptation rule ad is *applicable* to the current run-time model, if and only if also the precondition of ad is met, i.e., O can be extended to O' such that $O \subseteq O' \subseteq O^{\mathrm{RT}}$ and O' matches $PCP_{ad} \cup Pre_{ad}$. It should be noted that Pre_{ad} has exactly the same syntactic structure as PCPs, and so the definition of a match is also defined for $PCP_{ad} \cup Pre_{ad}$.

Suppose that ad is applicable. Then, *applying* adaptation rule ad to run-time model M^{RT} leads to a new run-time model denoted as $ad(M^{\mathrm{RT}})$, which is the same as M^{RT} except for the following differences:

- Let $o \in O_{ad}$ be an object in the adaptation rule. If o is of the form $(n_o^{ad}, A_o^{ad}, \texttt{<<create>>})$, then $ad(M^{\mathrm{RT}})$ contains a new object not contained in M^{RT}, with name n_o^{ad} and attributes according to A_o^{ad}. On the other hand, if o is of the form $(n_o^{ad}, A_o^{ad}, \texttt{<<delete>>})$, then $ad(M^{\mathrm{RT}})$ does not contain the object $f(o)$, nor any relations of $f(o)$.
- If $a \in A_o^{ad}$ is an attribute description in ad and has the form $(n_a^{ad}, \text{“=”}, v_a^{ad})$, then the attribute of $f(o)$ with name n_a^{ad} has the value v_a^{ad} in $ad(M^{\mathrm{RT}})$.
- If $r \in R_{ad}$ is a relation description in ad and has the form $(s_r^{ad}, g_r^{ad}, n_r^{ad}, \texttt{<<create>>})$, then $ad(M^{\mathrm{RT}})$ contains a new relation not contained in M^{RT}, with name n_r^{ad} between the objects $f(s_r^{ad})$ and $f(g_r^{ad})$. On the other hand, if r is of the form $(s_r^{ad}, g_r^{ad}, n_r^{ad}, \texttt{<<delete>>})$, then $ad(M^{\mathrm{RT}})$ does not contain a relation with name n_r^{ad} between the objects $f(s_r^{ad})$ and $f(g_r^{ad})$.

G. COST ANALYSIS

Let $TC \subseteq T$ denote the types relevant for the cost analysis. The types in TC are associated with two cost-related attributes:

- The attribute `costIncurred` is Boolean and has the value `true` if and only if the given object is used and hence incurs costs.

- The attribute `usageCost` is a non-negative number specifying the costs incurred if the given object is used.

The total cost incurred in the configuration defined by run-time model M^{RT} is calculated by adding up the usage costs of the cost-relevant objects that are in use:

$$C\left(M^{\text{RT}}\right) = \sum_{o \in O^{\text{RT}},\ t_o^{\text{RT}} \in TC,\ o.\text{costIncurred} = \text{true}} o.\text{usageCost}.$$

H. FUNCTIONALITY ANALYSIS

Formally, a *functionality* is defined as $f = (TS_f, TT_f, RP_f)$. The elements of f are as follows:

- $TS_f \subseteq T$ denotes the set of types that act as providers of the functionality f.
- $TT_f \subseteq T$ denotes the set of types that act as consumers of the functionality f.
- $RP_f \subseteq R$ denotes the set of relations that are suitable to provide the functionality f.

The set of functionalities is denoted as $\mathcal{F}$. An *instance* of a functionality f in the run-time model M^{RT} is a path $p = (o_1, o_2, \ldots, o_k)$ in M^{RT}, such that

- the starting object of p has a type from TS_f, i.e., $t_{o_1}^{\text{RT}} \in TS_f$,
- the end object of p has a type from TT_f, i.e., $t_{o_k}^{\text{RT}} \in TT_f$,
- and each relation along p is in RP_f, i.e., $\forall i \in [1, k-1]$, $\exists r \in RP_f$, $s_r^{\text{RT}} = o_i$, $g_r^{\text{RT}} = o_{i+1}$.

The set of instances of functionality f in the run-time model M^{RT} is denoted by $I\left(f, M^{\text{RT}}\right)$.

The total number of available functionalities in the configuration defined by run-time model M^{RT} is calculated by adding up the number of instances of each functionality:

$$F\left(M^{\text{RT}}\right) = \sum_{f \in \mathcal{F}} \left| I\left(f, M^{\text{RT}}\right) \right|.$$

APPENDIX II. USED TECHNOLOGIES

This section outlines the technologies that we used to construct the meta-model, to store the run-time model and to realize the adaptation planning algorithm.

A. ECLIPSE MODELING FRAMEWORK (EMF)

We implemented the meta-model using EMF. EMF allows to transform structured data models into Java source code [56]. It also provides an editor to create these data models. EMF generates the source code which forms the basis for the run-time model and the target configurations. Both run-time model and target configurations are based on the same meta-model. We also use EMF to serialize the run-time model and the target configurations into files in XMI format[15] for test purposes.

We implemented an additional transformation functionality to convert run-time models and target configurations from EMF to JSON format,[16] as well as to import JSON representations of run-time models and target configurations. JSON is a widely used format for providing and receiving information, and our implementation uses JSON format in the interfaces described in Section VI-B.

As both EMF and the meta-model described in Section V-B are compliant with UML, the meta-model was easily implemented in EMF, with just two adjustments. First, EMF represents a graph as a tree structure, hence a root node had to be added to the meta-model. Second, to represent relations in XMI and JSON, EMF uses IDs that consist of meta-model-specific class names and run-time-generated object identifiers. These IDs are unique for each object of a run-time model / target configuration. We are using the IDs, for example, to identify objects that should be updated due to monitoring events.

B. HENSHIN

The adaptation logic uses the Henshin model transformation library for finding PCP instances and performing adaptations. Henshin provides a model transformation language for EMF [57]. Henshin uses transformation rules that are made of two parts: Left-hand side and right-hand side. The left-hand side of a transformation rule captures the pattern that should be found in a model. The right-hand side of the transformation rule specifies how elements of the model that match the pattern should be transformed. After a pattern is detected and transformed, an output model is returned. In a Henshin rule, annotations are used to specify the role that the objects and relations play in the rule.

We use Henshin transformation rules to identify PCP instances. For this purpose, we use the annotations `<<preserve>>` and `<<forbid>>` in the left-hand side of a transformation rule. All elements annotated with `<<preserve>>` must be present in the run-time model to achieve a match. Elements with the `<<forbid>>` annotation must not exist in the run-time model to achieve a match.

We also use Henshin transformation rules to capture adaptation rules. Possible adaptations are represented by the output model of a transformation and are saved as a target configuration. To encode adaptation actions in Henshin, we use the annotations `<<create>>`, `<<delete>>` and `<<set>>` in the right-hand side of a transformation rule. Nodes and edges annotated with `<<create>>` are added to the model, while those annotated with `<<delete>>` are removed from the model. Setting the value of an attribute is achieved with `<<set>>`.

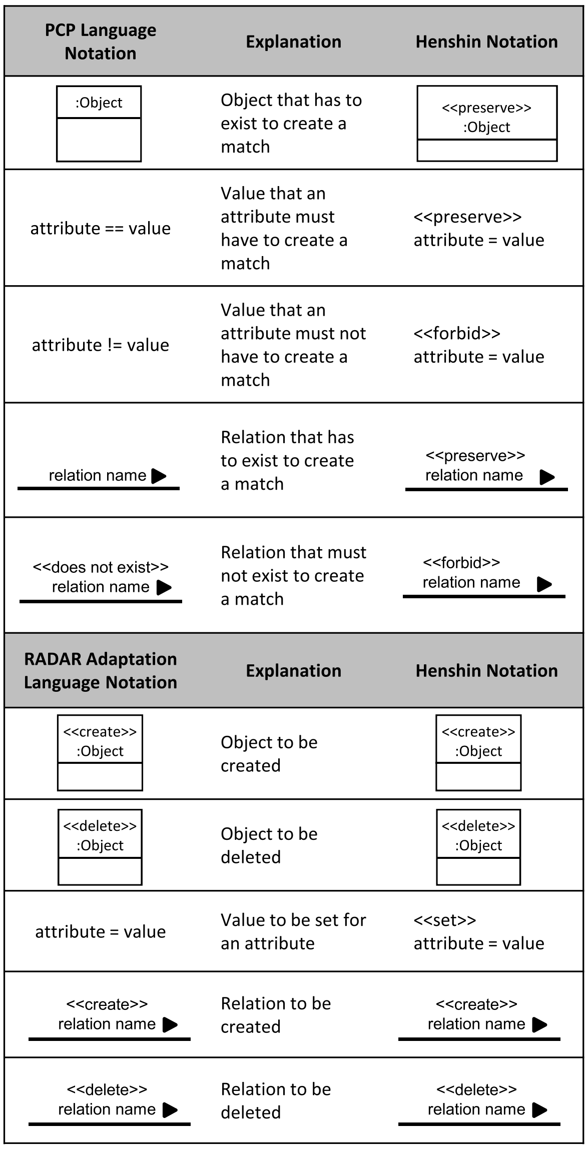
Thus, the PCP language and the RADAR adaptation language described in Section V-C and Section V-D can be mapped to Henshin. The mapping is shown in Figure 17.

C. SPRING BOOT

RADAR is implemented as a Java application using Spring Boot. Spring Boot is part of the open-source Spring Framework [58], and allows us to consistently configure and manage Java objects by using reflection. This simplifies, for

[15]XMI (XML Metadata Interchange) is a standard for exchanging meta-data information via XML.

[16]JavaScript Object Notation, https://www.json.org/



PCP Language Notation	Explanation	Henshin Notation
:Object	Object that has to exist to create a match	<<preserve>> :Object
attribute == value	Value that an attribute must have to create a match	<<preserve>> attribute = value
attribute != value	Value that an attribute must not have to create a match	<<forbid>> attribute = value
relation name ▶	Relation that has to exist to create a match	<<preserve>> relation name ▶
<<does not exist>> relation name ▶	Relation that must not exist to create a match	<<forbid>> relation name ▶
RADAR Adaptation Language Notation	**Explanation**	**Henshin Notation**
<<create>> :Object	Object to be created	<<create>> :Object
<<delete>> :Object	Object to be deleted	<<delete>> :Object
attribute = value	Value to be set for an attribute	<<set>> attribute = value
<<create>> relation name ▶	Relation to be created	<<create>> relation name ▶
<<delete>> relation name ▶	Relation to be deleted	<<delete>> relation name ▶

FIGURE 17. Mapping the PCP language and the RADAR adaptation language to Henshin.

example, the processing of the run-time model. Spring Boot supports the creation of stand-alone Spring-based applications.[17] An embedded Tomcat server as well as Apache Groovy[18] support make it easy to set up server applications with multiple interfaces. These interfaces are needed to provide the functionalities described in Section VI-B. Spring Boot makes it possible to deploy web applications directly as JAR files that contain all the software infrastructure, including a web server.

D. SIRIUS

Sirius[19] is an Eclipse plugin for creating editors for an existing EMF meta-model. Such editors can be used to work with models corresponding to the meta-model. We used Sirius to create an editor that can create and edit models conforming to the RADAR meta-model (see Section V-B). This way, run-time models can be created and visualized, which is useful for testing purposes.

[17]https://spring.io/projects/spring-boot

[18]Apache Groovy is a dynamic object-oriented programming and scripting language, see https://groovy-lang.org/.

[19]https://www.eclipse.org/sirius/doc/

ACNOWLEDGMENT

Useful discussions with project partners in the EU Horizon 2020 Research and Innovation Projects "RestAssured" and "FogProtect" are gratefully acknowledged.

REFERENCES

[1] General Data Protection Regulation, "Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC," *Off. J. Eur. Union*, vol. 59, p. L119, May 2016.

[2] S. Gurses and J. M. del Alamo, "Privacy engineering: Shaping an emerging field of research and practice," *IEEE Secur. Privacy*, vol. 14, no. 2, pp. 40–46, Mar. 2016.

[3] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The rise of 'big data' on cloud computing: Review and open research issues," *Inf. Syst.*, vol. 47, pp. 98–115, Jan. 2015.

[4] V. Chang and M. Ramachandran, "Towards achieving data security with the cloud computing adoption framework," *IEEE Trans. Services Comput.*, vol. 9, no. 1, pp. 138–151, Jan. 2016.

[5] M. Ali, S. U. Khan, and A. V. Vasilakos, "Security in cloud computing: Opportunities and challenges," *Inf. Sci.*, vol. 305, pp. 357–383, Jun. 2015.

[6] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *Future Gener. Comput. Syst.*, vol. 78, pp. 680–698, Jan. 2018.

[7] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in cloud computing: State of the art and research challenges," *IEEE Trans. Services Comput.*, vol. 11, no. 2, pp. 430–447, Mar. 2018.

[8] W. Sun, R. Zhang, W. Lou, and Y. Thomas Hou, "REARGUARD: Secure keyword search using trusted hardware," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2018, pp. 801–809.

[9] J. Heurix, P. Zimmermann, T. Neubauer, and S. Fenz, "A taxonomy for privacy enhancing technologies," *Comput. Secur.*, vol. 53, pp. 1–17, Sep. 2015.

[10] A. Senarath and N. A. G. Arachchilage, "Why developers cannot embed privacy into software systems?: An empirical investigation," in *Proc. 22nd Int. Conf. Eval. Assessment Softw. Eng.*, Jun. 2018, pp. 211–216.

[11] N. Gol Mohammadi, Z. A. Mann, A. Metzger, M. Heisel, and J. Greig, "Towards an end-to-end architecture for run-time data protection in the cloud," in *Proc. 44th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Aug. 2018, pp. 514–518.

[12] B. B. Mehta and U. P. Rao, "Privacy preserving big data publishing: A scalable k-anonymization approach using MapReduce," *IET Softw.*, vol. 11, no. 5, pp. 271–276, 2017.

[13] N. Bencomo, S. Götz, and H. Song, "Models@run.time: A guided tour of the state of the art and research challenges," *Softw. Syst. Model.*, vol. 18, no. 5, pp. 1–34, 2019.

[14] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu, "Graph pattern matching: From intractable to polynomial time," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 264–275, Sep. 2010.

[15] S. Sartoli and A. S. Namin, "A semantic model for action-based adaptive security," in *Proc. Symp. Appl. Comput.*, A. Seffah, B. Penzenstadler, C. Alves, and X. Peng, Eds. New York, NY, USA: ACM, Apr. 2017, pp. 1130–1135.

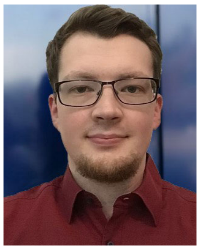
[16] A. Alebrahim, D. Hatebur, S. Faßbender, L. Goeke, and I. Côté, "A pattern-based and tool-supported risk analysis method compliant to ISO 27001 for cloud systems," *Int. J. Secure Softw. Eng.*, vol. 6, no. 1, pp. 24–46, Jan. 2015.

[17] L. Pasquale, S. Hanvey, M. Mcgloin, and B. Nuseibeh, "Adaptive evidence collection in the cloud using attack scenarios," *Comput. Secur.*, vol. 59, pp. 236–254, Jun. 2016.

[18] P. Massonet, S. Naqvi, C. Ponsard, J. Latanicki, B. Rochwerger, and M. Villari, "A monitoring and audit logging architecture for data location compliance in federated cloud infrastructures," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Workshops Phd Forum*, May 2011, pp. 1510–1517.

[19] S. Iannucci and S. Abdelwahed, "Model-based response planning strategies for autonomic intrusion protection," *ACM Trans. Auto. Adapt. Syst.*, vol. 13, no. 1, pp. 1–23, May 2018.
[20] J. Bürger, D. Strüber, S. Gärtner, T. Ruhroth, J. Jürjens, and K. Schneider, "A framework for semi-automated co-evolution of security knowledge and system models," *J. Syst. Softw.*, vol. 139, pp. 142–160, May 2018.
[21] C. Tsigkanos, L. Pasquale, C. Ghezzi, and B. Nuseibeh, "On the interplay between cyber and physical spaces for adaptive security," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 3, pp. 466–480, May 2018.
[22] S. Schoenen, Z. A. Mann, and A. Metzger, "Using risk patterns to identify violations of data protection policies in cloud systems," in *Service-Oriented Computing—ICSOC 2017 Workshops*. Cham, Switzerland: Springer, 2017, pp. 296–307.
[23] Z. A. Mann, A. Metzger, J. Prade, and R. Seidl, "Optimized application deployment in the fog," in *Proc. Int. Conf. Service-Oriented Comput. (ICSOC)*, vol. 11895. Cham, Switzerland: Springer, 2019, pp. 283–298.
[24] A. Le, J. Loo, A. Lasebae, M. Aiash, and Y. Luo, "6LoWPAN: A study on QoS security threats and countermeasures using intrusion detection system approach," *Int. J. Commun. Syst.*, vol. 25, no. 9, pp. 1189–1212, Sep. 2012.
[25] X. Zhang, N. Wuwong, H. Li, and X. Zhang, "Information security risk management framework for the cloud computing environments," in *Proc. 10th IEEE Int. Conf. Comput. Inf. Technol.*, Jun. 2010, pp. 1328–1334.
[26] N. van Dijk, R. Gellert, and K. Rommetveit, "A risk to a right? Beyond data protection risk assessments," *Comput. Law Secur. Rev.*, vol. 32, no. 2, pp. 286–306, Apr. 2016.
[27] A. S. Abohamama and E. Hamouda, "A hybrid energy–aware virtual machine placement algorithm for cloud environments," *Expert Syst. Appl.*, vol. 150, Jul. 2020, Art. no. 113306.
[28] M. Du, Y. Wang, K. Ye, and C. Xu, "Algorithmics of cost-driven computation offloading in the edge-cloud environment," *IEEE Trans. Comput.*, vol. 69, no. 10, pp. 1519–1532, Oct. 2020.
[29] R. O. Aburukba, M. AliKarrar, T. Landolsi, and K. El-Fakih, "Scheduling Internet of Things requests to minimize latency in hybrid fog–cloud computing," *Future Gener. Comput. Syst.*, vol. 111, pp. 539–551, Oct. 2020.
[30] J. Bellendorf and Z. Á. Mann, "Classification of optimization problems in fog computing," *Future Gener. Comput. Syst.*, vol. 107, pp. 158–176, Jun. 2020.
[31] J. Bürger, S. Gärtner, T. Ruhroth, J. Zweihoff, J. Jürjens, and K. Schneider, "Restoring security of long-living systems by co-evolution," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, S. I. Ahamed, C. K. Chang, W. C. Chu, I. Crnkovic, P. Hsiung, G. Huang, and J. Yang, Eds. Washington, DC, USA: IEEE Computer Society, Jul. 2015, pp. 153–158.
[32] G. Gonzalez-Granadillo, E. Alvarez, A. Motzek, M. Merialdo, J. Garcia-Alfaro, and H. Debar, "Towards an automated and dynamic risk management response system," in *Proc. Nordic Conf. Secure IT Syst.*, B. B. Brumley and J. Röning, Eds. Cham, Switzerland: Springer, 2016, pp. 37–53.
[33] G. Gonzalez-Granadillo, S. Dubus, A. Motzek, J. Garcia-Alfaro, E. Alvarez, M. Merialdo, S. Papillon, and H. Debar, "Dynamic risk management response system to handle cyber threats," *Future Gener. Comput. Syst.*, vol. 83, pp. 535–552, Jun. 2018.
[34] K. Kritikos, M. Papoutsakis, S. Ioannidis, and K. Magoutis, "Towards configurable cloud application security," in *Proc. 19th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2019, pp. 684–689.
[35] F. Kunz and Z. A. Mann, "Finding risk patterns in cloud system models," in *Proc. IEEE 12th Int. Conf. Cloud Comput. (CLOUD)*, E. Bertino, C. K. Chang, P. Chen, E. Damiani, M. Goul, and K. Oyama, Eds., Jul. 2019, pp. 251–255.
[36] M. Nguyen, P. Samanta, and S. Debroy, "Analyzing moving target defense for resilient campus private cloud," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*. Washington, DC, USA: IEEE Computer Society, Jul. 2018, pp. 114–121.
[37] N. Nostro, A. Ceccarelli, A. Bondavalli, and F. Brancati, "Insider threat assessment: A model-based methodology," *ACM SIGOPS Operating Syst. Rev.*, vol. 48, no. 2, pp. 3–12, Dec. 2014.
[38] A. Palm, Z. A. Mann, and A. Metzger, "Modeling data protection vulnerabilities of cloud systems using risk patterns," in *Proc. Int. Conf. Syst. Anal. Modeling (SAM)*, F. Khendek and R. Gotzhein, Eds. Cham, Switzerland: Springer, 2018, pp. 1–19.
[39] L. Pasquale, C. Ghezzi, C. Menghi, C. Tsigkanos, and B. Nuseibeh, "Topology aware adaptive security," in *Proc. 9th Int. Symp. Softw. Eng. Adapt. Self-Manag. Syst. (SEAMS)*, 2014, pp. 43–48.
[40] G. Puppala and S. K. Pasupuleti, "Dynamic security risk assessment in cloud computing using IAG," in *Progress in Computing, Analytics and Networking*. Singapore: Springer, 2018, pp. 105–116.
[41] M. Salehie, L. Pasquale, I. Omoronyia, R. Ali, and B. Nuseibeh, "Requirements-driven adaptive security: Protecting variable assets at runtime," in *Proc. 20th IEEE Int. Requirements Eng. Conf. (RE)*. Washington, DC, USA: IEEE Computer Society, Sep. 2012, pp. 111–120.
[42] E. Schmieders, A. Metzger, and K. Pohl, "Runtime model-based privacy checks of big data cloud services," in *Proc. Int. Conf. Service-Oriented Comput. (ICSOC)*. Berlin, Germany: Springer, 2015, pp. 71–86.
[43] F. Durán and G. Salaün, "Robust and reliable reconfiguration of cloud applications," *J. Syst. Softw.*, vol. 122, pp. 524–537, Dec. 2016.
[44] K. Kritikos, C. Zeginis, E. Politaki, and D. Plexousakis, "Towards the modelling of adaptation rules and histories for multi-cloud applications," in *Proc. 9th Int. Conf. Cloud Comput. Services Sci. (CLOSER)*, 2019, pp. 300–307.
[45] J. Bellendorf and Z. Á. Mann, "Specification of cloud topologies and orchestration using TOSCA: A survey," *Computing*, vol. 102, no. 8, pp. 1793–1815, Aug. 2020.
[46] Z. A. Mann, A. Metzger, and S. Schoenen, "Towards a run-time model for data protection in the cloud," in *Modellierung 2018*, I. Schaefer, D. Karagiannis, A. Vogelsang, D. Méndez, and C. Seidl, Eds. Bonn, Germany: Gesellschaft für Informatik, 2018, pp. 71–86.
[47] A. R. Hummaida, N. W. Paton, and R. Sakellariou, "Adaptation in cloud resource configuration: A survey," *J. Cloud Comput.*, vol. 5, no. 1, pp. 1–16, Dec. 2016.
[48] A. Frommgen, R. Rehner, M. Lehn, and A. Buchmann, "Fossa: Learning ECA rules for adaptive distributed systems," in *Proc. IEEE Int. Conf. Autonomic Comput.*, Jul. 2015, pp. 207–210.
[49] T. Vogel and H. Giese, "Model-driven engineering of self-adaptive software with EUREMA," *ACM Trans. Auto. Adapt. Syst.*, vol. 8, no. 4, pp. 18:1–18:33, 2014.
[50] N. Ferry, F. Chauvel, H. Song, A. Rossini, M. Lushpenko, and A. Solberg, "CloudMF: Model-driven management of multi-cloud applications," *ACM Trans. Internet Technol.*, vol. 18, no. 2, pp. 16:1–16:24, 2018.
[51] K. Fatema, V. C. Emeakaroha, P. D. Healy, J. P. Morrison, and T. Lynn, "A survey of cloud monitoring tools: Taxonomy, capabilities and objectives," *J. Parallel Distrib. Comput.*, vol. 74, no. 10, pp. 2918–2933, Oct. 2014.
[52] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *Proc. IEEE 6th Int. Conf. Cloud Comput.*, Jun. 2013, pp. 887–894.
[53] Y. Cherdantseva and J. Hilton, "A reference model of information assurance & security," in *Proc. Int. Conf. Availability, Rel. Secur.*, Sep. 2013, pp. 546–555.
[54] S. Shastri, M. Wasserman, and V. Chidambaram, "GDPR anti-patterns," *Commun. ACM*, vol. 64, no. 2, pp. 59–65, Jan. 2021.
[55] Z. A. Mann and A. Metzger, "Auto-adjusting self-adaptive software systems," in *Proc. IEEE Int. Conf. Autonomic Comput. (ICAC)*, Sep. 2018, pp. 181–186.
[56] F. Budinsky, D. Steinberg, R. Ellersick, T. J. Grose, and E. Merks, *Eclipse Modeling Framework: A Developer's Guide*. Reading, MA, USA: Addison-Wesley, 2004.
[57] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer, "Henshin: Advanced concepts and tools for in-place EMF model transformations," in *Proc. Int. Conf. Model Driven Eng. Lang. Syst. (MODELS)*. Berlin, Germany: Springer, 2010, pp. 121–135.
[58] F. Gutierrez, *Introducing Spring Framework: A Primer*. New York, NY, USA: Apress, 2014.

ZOLTÁN ÁDÁM MANN is currently a Senior Researcher with paluno—The Ruhr Institute for Software Technology, University of Duisburg-Essen, Germany. His research interests include fog computing, software engineering, cloud computing, security and privacy, and optimization algorithms.



FLORIAN KUNZ is currently a Research Assistant with paluno—The Ruhr Institute for Software Technology, University of Duisburg-Essen, Germany. His research interests include adaptive software-systems and data security in cloud and fog computing.

JAN LAUFER is currently a Research Assistant with paluno—The Ruhr Institute for Software Technology, University of Duisburg-Essen, Germany. His research interests include fog computing and data protection threats in fog computing.

JULIAN BELLENDORF is currently a Research Assistant with paluno—The Ruhr Institute for Software Technology, University of Duisburg-Essen, Germany. His research interests include fog computing and data security in the cloud.

ANDREAS METZGER is a Senior Academic Councilor at the University of Duisburg-Essen and the Head of adaptive systems and big data applications at paluno, The Ruhr Institute for Software Technology. He is the Steering Committee Vice-Chair of the European Technology Platform on Software, Services, Cloud and Data (NESSI) and the Deputy Secretary General of the Big Data Value Association (BDVA). His research interests are in software engineering for self-adaptive systems.

KLAUS POHL is a Full Professor for software systems engineering at the University of Duisburg-Essen. He is the Director of paluno–The Ruhr Institute for Software Technology and was the Scientific Founding Director of Lero–The Irish Software Engineering Centre. He is member of the NESSI Board and member of several steering and advisory committees.