# Runtime Model-Based Privacy Checks of Big Data Cloud Services

Eric Schmieders(✉), Andreas Metzger, and Klaus Pohl

paluno (The Ruhr Institute for Software Technology),
University of Duisburg-Essen, Essen, Germany
{eric.schmieders,andreas.metzger,klaus.pohl}@paluno.uni-due.de

**Abstract.** Cloud services have to comply with privacy policies when storing or processing data. As cloud services become increasingly data-intensive, e.g., in the case of big data analytics, data privacy concerns become more critical and challenging to address. In particular, data may only be processed at certain geo-locations. However, the actual geo-locations of the many storage and compute nodes involved in big data processing is dynamically selected during runtime. In addition, the execution of concrete data processing tasks may change data classifications from, e.g., personal to anonymized data. Thus, privacy policy checks for big data cloud services have to consider information about the actual nodes and data processing tasks at runtime. The proposed approach R-PRIS monitors cloud services to derive and maintain typed runtime models providing the aforementioned information. R-PRIS checks the typed runtime models against privacy policies by employing a data-classification-aware search. The evaluation of R-PRIS, performed on Amazon Web Services (including Hadoop), indicates that the approach may efficiently and timely detect privacy violations in big data cloud services.

**Keywords:** Privacy · Big data · Cloud services · Runtime checking

## 1 Introduction

Cloud services have to comply with privacy policies when storing, transferring, and processing data. For instance, the EU Data Protection Directive[1] (DPD) as well as the US Health Insurance Portability and Accountability Act[2] (HIPAA) only permit processing personal data within countries that implement sufficient data protection mechanisms. Moreover, privacy policies, such as the ones proposed by NIST 800-122 or FIPS 199[3], distinguish between different data classifications. Data classifications indicate the data's identifiability or sensitivity, which requires to treat the classified data accordingly.

1 http://eur-lex.europa.eu/.
2 http://www.hhs.gov/ocr/privacy/.
3 http://csrc.nist.gov/.

As cloud services become increasingly data-intensive – being used for large-scale and real-time big data analytics tasks for instance [3] – the implementation of such privacy policies becomes ever more challenging. Data and processing tasks are distributed among a vast number of storage and compute nodes to cope with the high volume of data and to ensure the high velocity of data processing (e.g., when using the MapReduce programming model). In addition, data classifications may dynamically change based on the data processing tasks executed by the various compute nodes (e.g., a task may aggregate personal customer data into anonymized sales statistics). On top of that, storage and compute nodes may be dynamically deployed, replicated, and migrated to achieve performance, availability, and cost goals of cloud providers. Given all this complexity and dynamism, each of the involved nodes still has to comply with privacy policies during the entire cloud service life-cycle.

Existing approaches for checking privacy policies have not addressed cloud elasticity or data classification changes (e.g., [6,7,10,13]). In our previous work [16,17], we introduced R-PRIS, a privacy compliance checking approach for dynamic cloud services. In this paper we extend R-PRIS to cope with the aforementioned challenges imposed by *data-intensive* services. In particular, the extended R-PRIS approach is able to consider data classification changes. This is important, as disregarding data classification changes may lead to a high rate of false positive violations. Such a high positive rate would limit applicability for big data cloud services. For instance, it would prohibit migrating many of the storage and compute nodes that do not process privacy-relevant data.

The extended R-PRIS approach utilizes a data-classification-aware search strategy based on typed runtime models. To this end, we propose typed runtime models for reflecting data placement and data classification changes. In order to facilitate the specification of fine-grained privacy polices, R-PRIS allows for defining data classifications, impact levels, and their relations. Using this information, R-PRIS monitors the cloud services, automatically updates the typed runtime model, and checks the model against privacy policies.

We evaluated the applicability and performance of R-PRIS for a data-intensive cloud service hosted on Amazon Web Services (AWS), leveraging Hadoop clusters for data processing. Our results indicate that R-PRIS is able to efficiently and timely detect privacy violations in big data cloud services.

The remainder of the paper is structured as follows: Sect. 2 identifies relevant classes of privacy policy violations. Using these classes of violations as basis, Sect. 3 describes the R-PRIS approach. Section 4 investigates the realizability of the approach by means of a proof of concept implementation. Section 5 describes the setup and results of experiments to evaluate the performance of R-PRIS. Related work is discussed in Sect. 6.

## 2 Privacy Policy Violations in Big Data Cloud Services

Cloud providers allow for specifying the geo-locations at which virtual machines shall be executed. However, misconfigured cloud infrastructures, software failures, and incorrect geo-location specifications might lead to virtual machine placements that result in privacy policy violations.

As a simple example for such situations, take two interacting service components $v1$ and $v2$ that process personal data. Both components are deployed on separate virtual machines that are initially hosted on cloud data centers within the EU. During runtime, the virtual machine hosting $v1$ is migrated to a data center outside of the EU. After migration, $v1$ and $v2$ continue to exchange personal data. However, as this now implies transfer of data beyond EU borders, this may violate data geo-location policies.

To identify these dynamic changes that need to be reflected in the R-PRIS runtime models, we systematically determine changes of cloud services properties. We focus on changes that stem from the cloud service software architecture and the underlying cloud infrastructure. We analyze whether the identified changes can lead to privacy policy violations. To be specific, we follow three steps in our analysis: (i) we identify architectural, deployment, and functional *properties* of big data cloud services that – if changed – may lead to policy violations (column 1 in Table 1); (ii), we determine concrete types of *changes* of these properties; e.g., due to cloud elasticity mechanisms (column 2); and (iii) we identify the concrete *violations* of privacy policies resulting from these changes (column 3). Table 1 summarizes the results of the analysis.

**Table 1.** Policy violations in big data cloud services

| # | Property | Change | Violation |
|---|---|---|---|
| 1 | Compute node's geo-location | New node; replication; migration | A compute node is instantiated at an excluded geo-location (e.g., a virtual machine hosting a marketing application is instantiated in the US) |
| 2 | Storage nodes geo-location | Data upload; replication; migration | A storage node is assigned to store personal data excluded at the node's geo-location (e.g., a Hadoop node has to process a new chunk of personal data) |
| 3 | Node interactions | New node; migration; replication | A new interaction between nodes is established that involve transfer of personal data to excluded geo-location (e.g., an existing Hadoop cluster is integrated into a cloud service) |
| 4 | Data classifications | Changed processing task | The new task may produce data with different classification (e.g., a Hadoop task determines the average number of purchased items for a specific customer) |

## 3 R-PRIS: Privacy Checks for Big Data Cloud Services

The main idea of R-PRIS (*Runtime model-based PRIvacy checkS*) is to utilize runtime models for performing privacy policy checks. In general, runtime models are dynamically updated abstractions of the reflected systems [18]. The architectural runtime models of R-PRIS contain information about compute and storage node deployments, data flows occurring among these nodes, and data classification changes. The main steps of R-PRIS are depicted in Fig. 1: cloud *monitoring* information is used for *updating* architectural runtime models, which in turn are employed for policy violation *checks*.

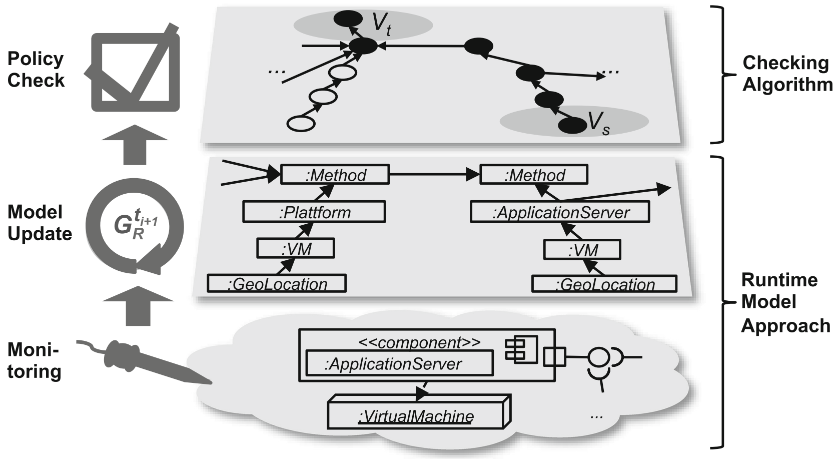


**Fig. 1.** Overview of R-PRIS

This paper focuses on two main new contributions introduced by the extended R-PRIS approach: the use of a typed runtime model to consider data classification changes (Sect. 3.1) and the enhanced privacy policy checks based on this model (Sect. 3.2).

## 3.1 Typed Runtime Model

The typed runtime model employed for taking data classification changes into account is an extension of the architectural runtime model that we introduced in [16,17]. The architectural runtime model $G_R^{t_i}$ reflects the deployment of storage and compute nodes (properties 1 and 2 in Table 1), as well as node interactions (property 3 in Table 1) at time point $t_i$.

In R-PRIS, the updates of $G_R^{t_i}$ are specified as graph transformation rules. Event-condition-action patterns are employed to reason on observed monitoring information as well as to trigger and parametrize the matching graph transformation rules. This results in an updated runtime model $G_R^{t_{i+1}}$ for time $t_{i+1}$.

When extending this architectural runtime model with the data classification information (property 4 in Table 1), we implemented a clear separation of concerns. We separated the dynamically evolving information about node deployments and interactions (architectural runtime model) from the rather stable information about data classifications (data classification model). For the mapping of elements between these two models (e.g., for mapping a *Method* node to its data classification) we utilize the typed graph concept.

The edges and vertices of a typed graph are assigned to types stored in a separate type graph (for an introduction to typed graphs see [4]). We consider the runtime model as the typed graph and the data classifications as the type graph. Similar to the dynamic type checking in type systems, in our work a set of rules defines how typed vertices are to be treated during runtime.

The concept of the typed runtime model will be described more precisely in the following. We use the typed runtime model shown in Fig. 2 as illustrative example. First, we formally define an R-PRIS architectural runtime model as:

**Definition 1.** *Let $G_R^{t_i} = (V_R, E_R, s_R, t_R)$ be a directed graph that models a big data cloud service at a certain point in time $t_i$. $V_R$ are the vertices, i.e. service entities, and $E_R$ the edges, i.e. relations between the cloud service entities. Function $s_R : E \rightarrow V$ specifies the source vertex of edges $E_R$ and function $t_R : E \rightarrow V$ specifies the target vertex of edges $E_R$.*

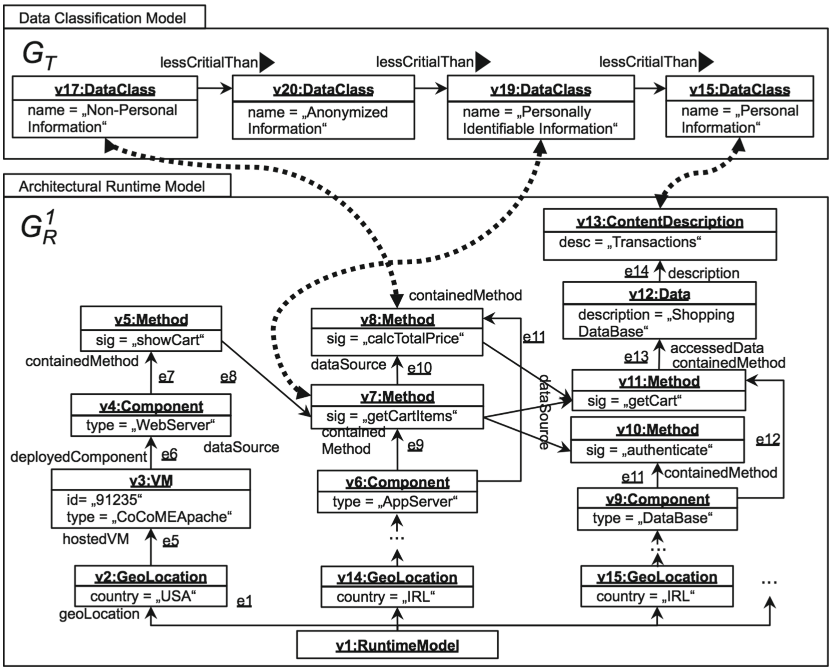


**Fig. 2.** Example of a typed runtime model in R-PRIS

The lower part of Fig. 2 shows an example of an R-PRIS architectural runtime model, where vertices are expressed as UML objects and edges are expressed as

UML associations. The different kinds of objects and associations are defined by a meta model (introduced in [16,17]). The meta model provides concepts for modeling virtual machines, components, and methods to reflect changes of the compute node property. The deployment relationship reflects changes of geo-locations and the data source relationships models changes in interactions.

To express how different data classifications relate to each other in terms of identifiability or sensitivity, we define a data classification model as follows:

**Definition 2.** *Let $G_T = (V_T, E_T, s_T, t_T)$ be the directed graph that represents the data classifications. $V_T$ are the vertices, i.e. data classifications, and $E_T$ the edges, expressing the relations between the data classifications. $s_T : E \rightarrow V$ defines the source vertex of edges $\in E_T$ and $t_T : E \rightarrow V$ defines the target vertex of edges $\in E_T$. Relations between data classifications are transitive*[4]*, i.e., $(x \succ y) \wedge (y \succ z) \rightarrow (x \succ z)$.*

The upper part of Fig. 2 shows an example for a data classification model $G_T$. It includes the data classes *Personal Information*, *Personally Identifiable Information*, etc. The relationship *lessCriticalThan* reflects the criticality relationship among data classifications.

To interrelate $G_R$ and $G_T$ we exploit the concept of typed graphs. Typed graphs employ graph morphisms for interrelating specific elements of two graphs. We define the graph morphism for R-PRIS as follows:

**Definition 3.** *Let $m : G_R^{t_i} \rightarrow G_T$ be a graph morphism that maps $G_R^{t_i}$ to $G_T$. Function $m$ is defined as $m = (m_V, m_E)$ with functions $m_V : V_R \rightarrow V_T$ and $m_E : E_R \rightarrow V_T \times V_T$.*

In our approach, $m_V$ and $m_E$ are specified as follows. $m_V$ maps *Content-Description* and *Method* entities to data classes $V_T$ of the data classification model. The mapping from *ContentDescription* describes the 'static' classification of contents (data objects). The mapping from *Method* describes a change of data classification imposed by executing the method (data processing tasks). All other $G_R^{t_i}$ entities are initially mapped to 'no' data classification ($\perp$). Starting from this initial mapping, we will dynamically update the graph morphism during the policy check in order to determine policy violations (see Sect. 3.2).

Based on Definitions 1–3, we can now define the R-PRIS typed runtime model:

**Definition 4.** *The R-PRIS typed runtime model is defined as $G_{typed} = (G_R^{t_i}, G_T, m)$ with $G_R^{t_i}$ as the architectural runtime model instance, $G_T$ the data classification model, and $m$ the graph morphism that maps $G_R^{t_i}$ to $G_T$.*

When processing data, methods can change the classifications of the processed data. For instance, method *getCartItens* ($v8$ in Fig. 2) changes personal information to personally identifiable information. To reflect this, we define data classification changes as:

[4] Which implies that $G_T$ has to be acyclic.

**Definition 5.** *Given a typed runtime model $G_{typed}$, let $v_k \in G_R^{t_i}$ represent a method or a data object and let $v_l \in G_R^{t_i}$ represent a method that accesses data provided by $v_k$. $v_l$ either retains the data classification of the accessed data, i.e. $m(v_k) = m(v_l)$ or changes the classification to a less critical one, i.e. $m(v_k) \succ m(v_l)$. We define the change (in the latter case) as data classification change.*

## 3.2 Privacy Policy Checks

The main idea of our privacy policy check is to express the check as a reachability analysis on the architectural runtime models. In [16] we have formalized and realized this reachability analysis as an st-connectivity problem on the architectural runtime model, thereby considering node deployment and interaction but not data classification changes.

### 3.2.1 Data-Classification-Aware Search

In order to cover data classification changes in big data cloud services, we extend the reachability analysis by the data classification model, i.e., we search the typed runtime model (including the type model, see Sect. 3.1). Let's take a policy $p = (geo, class)$, which prescribes that data classified as *class* must neither be processed nor stored at the specified geo-location *geo*. The reachability analysis then aims to find a 'violation path' that connects the 'forbidden' geo-location vertex for *geo* with the data classification vertex for *class*. In simple terms, a 'violation path' is a sequence of vertices connected by edges that do not exhibit a data classification less critical than *class*. If such a path exists, the cloud service either stores or processes data at the forbidden geo-location, thereby violating the checked policy $p$.

As an example, let's define $geo = USA$ and $class = PersonalInformation$, which means that our cloud service may not store or process personal information in the US. Using the typed runtime model of the cloud service in Fig. 2, the reachability analysis aims to find a 'violation path' that connects the 'forbidden' geo-location vertex *USA* ($v2$) with the data classification vertex *PersonalInformation* ($v15$). As can be seen, such a path cannot be found, because although $v13$ is typed as *PersonalInformation*, the path from $v13$ to $v2$ traverses vertices which change the data classification to less critical levels; e.g., $v7$ changes it to *PersonallyIdentifiableInformation*. This means that the example cloud service (in the current deployment reflected in the typed runtime model) complies with the privacy policy.

To realize the described reachability analysis, the policy check starts with defining $v^{geo} \in G_R^{t_i}$ as start vertex and $v^{class} \in G_R^{t_i}$ as target vertex (with the geo-location and the data classification specified in the policy to be checked). The check then performs a depth first search of the typed runtime model, during which three main mechanisms are employed:

- **Early Termination:** The search terminates the traversal of the current path as soon as it reaches a vertex $v_n$ classified as less critical than *class*. Regardless of the data classifications along any continuation of the current path, data that passes through $v_n$ will always be less critical than what is expressed in the privacy policy to be checked. This does not exclude the existence of other paths from $v^{geo}$ to $v^{class}$. Thus, we terminate the search at $v_n$ and then backtrack to $v_{n-1}$ to explore other paths to reach $v^{class}$.

– **Backtracking:** To facilitate backtracking, and thus not have to start from $v^{geo}$ each time we have terminated a path traversal, we employ the graph morphisms introduced in Sect. 3.1 to store the data classification for all vertices we have traversed thus far. This allows us to continue from $v_{n-1}$ by retrieving the classification we have computed at a certain point of traversal.
– **Classification Traversal:** The traversal of $G_R^{t_i}$ is continued as long as the classification of the successor vertex $v_{n+1}$ is higher or the same as the classification of the current vertex $v_n$. The continuation is decided by expanding the search space to $G_T$ (also in the case of backtracking). If there exists a path from the data classification of $v_n$ to the data classification of $v_{n+1}$, then the search continues. In this case, the method returns data equally or more critical[5] than specified in $p$.

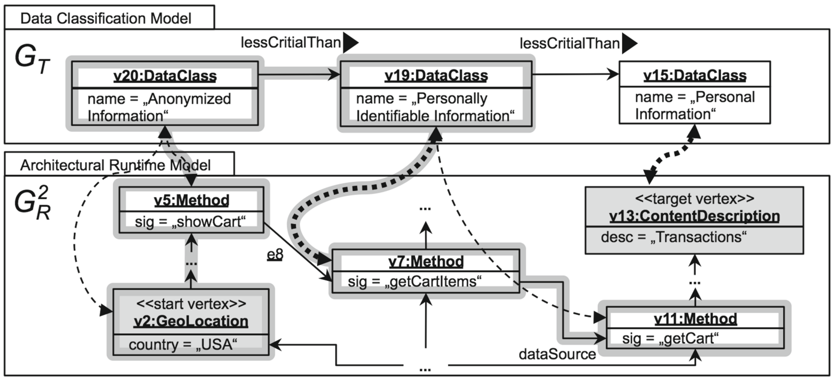


**Fig. 3.** Example traversal through the typed runtime model

### 3.2.2 Performance Optimization

When traversing $G_R^{t_i}$ and dynamically classifying $V_R \in G_R^{t_i}$ there is a situation that requires the utilization of a search heuristic for avoiding the re-visiting of nodes. This is important as revisiting nodes would negatively impact on the approach's time complexity.

Let us assume the check passes a vertex $v_s$ classified with $m_V(v_s) = c_1$. Further, the traversal leads over two crossing paths to a vertex $v_t$ with $m_V(v_t) = c_1$. One of these paths includes a $v_1$ with $m_V(v_1) = c_1$ and the other path $m_V(v_2) = c_2$ (with $(c_1, c_2)$ : *lessCriticalThan*). If the path over $c_2$ is traversed

[5] A policy concerning less critical data must also hold for more critical data.

first, then this traversal results in a false negative (the actual violation is not detected). The reasons are (i) that $c_1$ is less critical than $c_2$ such that the checked policy holds and (ii) the path over $c_1$ to $v_1$ is not traversed without visiting nodes twice, which blocks the traversal of the path for $c_1$. In order to tackle this issue without re-visiting nodes, the check visits vertices adjacent to a vertex $v$ with respect to their criticality in an ascending order (by omitting classifications less critical than $m_V(v)$), which avoids the blocking effect.

### 3.2.3 Example Policy Check

Figure 3 shows an excerpt of $G_R^2$, which is an update of $G_R^1$. It shows the morphism-based typings to ease backtracking as thinner arrows, whereas the broader arrows are the initial mappings.

Let us assume, we want to check policy $p$ = *(USA, AnonymizedInformation)*. We dynamically type the start node with the classification specified in $p$, i.e. $v2 \mapsto v20$. The algorithm starts to traverse $G_R^2$ and dynamically resolves $\perp$ with the classifications of the preceding vertices (enabling backtracking). The classification of $v5$ is set to $v20$. The classifications of $v5$ and $v7$ differ. Thus, the search is expanded to $G_T$ (classification traversal). As the classification of $v7$ is reachable from the classification of $v5$ the search continues. $v11$ is dynamically typed with the classification of the preceding note, i.e. $v11 \mapsto v19$. After traversing the data vertex $v12$ and typing it with *PersonallyIdentifyableInformation* the algorithm visits $v13$ (not shown in the figure). The algorithm checks whether there is a path from *PersonallyIdentifyableInformation* to the classification of $v13$ in $G_T$, which is the case. In consequence, data that is classified more critical than $v^{class}$ can be transferred into the USA, which violates $p$.

## 4 Proof of Concept Implementation

To demonstrate the feasibility and applicability of R-PRIS, we developed a prototype implementation that we deployed on actual cloud infrastructures. This R-PRIS prototype is also used during our experimental evaluation in Sect. 5.

### 4.1 Prototype Architecture

The R-PRIS prototype consists of six main components (see Fig. 4): the monitoring probes, the monitoring server, the event processor, the model controller, the policy checker, and a third-party host geo-location service. The probes are deployed on the virtual machines that host the big data cloud service's software components. The monitoring server forwards the parsed monitoring information to the event processor. The event processor invokes a REST service[6] for resolving the VM geo-location. The processor triggers the model controller to execute model transformation rules that modify the runtime model with respect to the observed service changes. We use the Henshin graph transformation API for performing the model updates. Henshin supports runtime model based on Ecore[7].

[6] http://freegeoip.net/json/.
[7] http://www.eclipse.org/modeling/emf/.

For more details on this model-update approach, see [17]. After updating the model, the model controller triggers the policy checker component, which checks the runtime model against the privacy policies (see Sect. 3.2).

The components we developed have been implemented in Java SE 1.7. They are deployed on a dedicated server hosted at our institute (R-PRIS server in Fig. 4) to have maximum control over the prototype and facilitate performance measurements without external influences. The server is equipped with 4 GB of RAM and a single core 2 GHz processor.

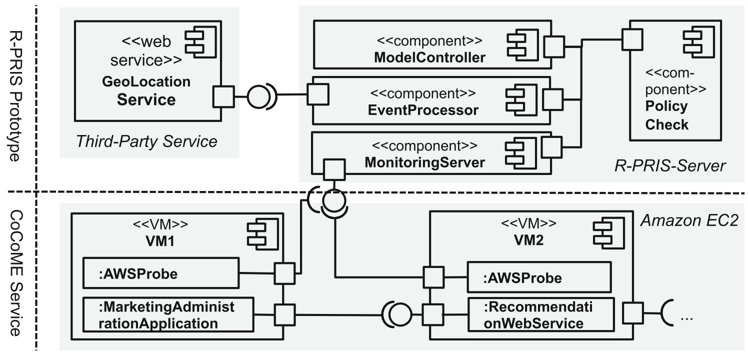


**Fig. 4.** R-PRIS prototype and parts of the example service

## 4.2 Cloud Service and Infrastructure

To test the applicability of R-PRIS, we employ a realistic cloud service that builds on the CoCoME case study [14]. CoCoME represents a typical trading service operated by a supermarket chain. In the CoCoME scenario of interest, the CoCoME service sends personalized recommendation e-mails to online customers by exploiting a Hadoop cluster for big data analytics. The Amazon reference architecture for e-commerce websites[8] is used as the underlying structure for the CoCoME service components. The architecture includes a recommendation web service, a marketing administration application, a Hadoop name node, and Hadoop data nodes. All virtual machines are instrumented with the R-PRIS monitoring probes. The lower half of Fig. 4 shows a subset of these virtual machines and the CoCoMe components they host. We choose Amazon EC2 as a realistic execution environment for the cloud services.

## 4.3 Change Scenarios

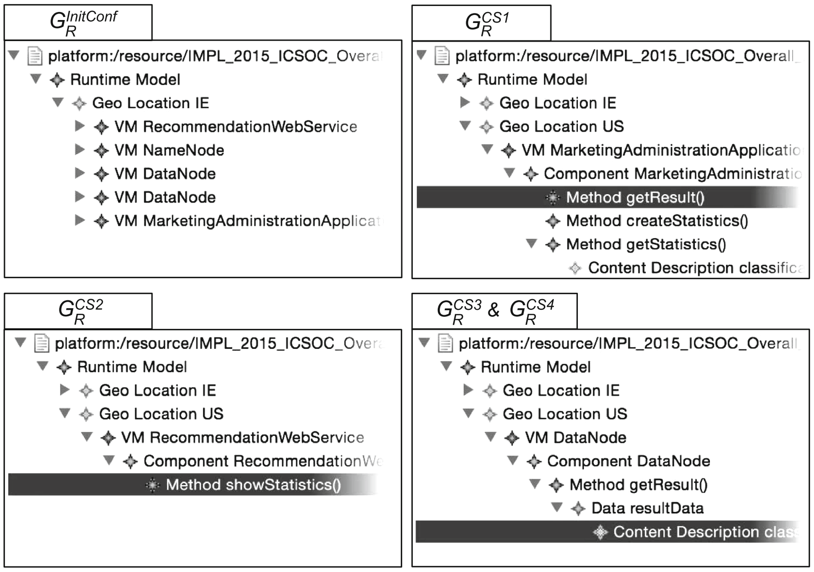
To assess the applicability of R-PRIS, we expose the prototype to four change scenarios. These scenarios cover all changes identified in Sect. 2. For each change,

[8] http://aws.amazon.com/architecture/.

we have defined a scenario with positive (policy violation) and negative (policy compliance) situations.

Figure 5 shows excerpts of the actual runtime models (as Eclipse EMF trees). As a pre-condition for each scenario, the virtual machines of the CoCoME service are deployed on a data center in Ireland (see $G_R^{InitConf}$ in Fig. 5). The other models show situations after executing the change scenarios. In each change scenario, one virtual machine is migrated to the US and thus potentially provokes a policy violation. Employing migration is sufficient in our change scenarios, as from a technical perspective, a migration shuts down a virtual machine at the source location and re-starts it at the target location (which leads to initializing a new node, new interactions, etc.; cf. [1]).

Table 2 shows the covered properties, the provoked violation, the applied changes, and the runtime model updates as well as the policy checker results. As the results from the change scenarios indicate, R-PRIS is able to keep the typed runtime model in sync with the reflected CoCoME service. Moreover, the policy checks correctly determine violations and compliance to CoCoME's privacy policy. In particular, the correct true negative check after the migration of the recommendation web service (case $G_R^{CS2}$) demonstrates that R-PRIS is able to avoid false positives that stem from ignoring data classification changes.



$G_R^{InitConf}$

```
▼ platform:/resource/IMPL_2015_ICSOC_Overa
   ▼ Runtime Model
      ▼ Geo Location IE
         ▶ VM RecommendationWebService
         ▶ VM NameNode
         ▶ VM DataNode
         ▶ VM DataNode
         ▶ VM MarketingAdministrationApplica
```

$G_R^{CS1}$

```
▼ platform:/resource/IMPL_2015_ICSOC_Overall_
   ▼ Runtime Model
      ▶ Geo Location IE
      ▼ Geo Location US
         ▼ VM MarketingAdministrationApplicatio
            ▼ Component MarketingAdministratio
                  Method getResult()
                  Method createStatistics()
               ▼ Method getStatistics()
                     Content Description classific
```

$G_R^{CS2}$

```
▼ platform:/resource/IMPL_2015_ICSOC_Over
   ▼ Runtime Model
      ▶ Geo Location IE
      ▼ Geo Location US
         ▼ VM RecommendationWebService
            ▼ Component RecommendationW
                  Method showStatistics()
```

$G_R^{CS3}$ & $G_R^{CS4}$

```
▼ platform:/resource/IMPL_2015_ICSOC_Overall
   ▼ Runtime Model
      ▶ Geo Location IE
      ▼ Geo Location US
         ▼ VM DataNode
            ▼ Component DataNode
               ▼ Method getResult()
                  ▼ Data resultData
                        Content Description clas
```

**Fig. 5.** Screenshots of runtime models (Eclipse EMF) generated by R-PRIS

**Table 2.** Executed change scenarios and observations

| Covered property | Scenario | Migrated VM | Observed R-PRIS Behaviour |
|---|---|---|---|
| Compute node geo-location (1), node interactions (3), data classification (4) | Positive | Marketing administration application | The runtime model reflects the applied change correctly (see $G_R^{CS1}$ in Fig. 5). The check detects a policy violation. The check message includes the violating data flow of personal data that starts in one of the $DataNodes$, leads over the $NameNode$ and is requested by the $getResult()$ method (marked grey in the figure). |
| | Negative | Recommendation web service | The runtime model reflects the change correctly (see $G_R^{CS2}$ in Fig. 5). The check assesses the CoCoME service as policy compliant. The $showStatistics()$ method of the recommendation web service invokes the $getStatistics()$ method of the marketing administration application. $getStatistics()$ accesses personal information from the data nodes, but changes the data's classification from $PersonalInformation$ to $PersonallyIndentifiableInformtion$. The changed classification does not violate the privacy policy. By reflecting the classification change, the check avoids a false positive in comparison to a check that does not take data classification into account. |
| Storage geo-location (2), interactions (3), classifications (4) | Positive | Data node (personal data) | The runtime model reflects the change correctly (see $G_R^{CS3}$ & $G_R^{CS4}$ in Fig. 5). The check detects the policy violation. Storing personal data at excluded geo-locations violates the privacy policy. |
| | Negative | Data node (non-personal data) | The runtime model reflects the change correctly (see $G_R^{CS3}$ & $G_R^{CS4}$ in Fig. 5). The check assesses the CoCoME service to be policy compliant. Only non-personal information is stored in the US. |

## 5 Performance Evaluation

Policy violations need to be detected timely in order to have sufficient time to mitigate and respond to these violations. Thus, as one key criterion of R-PRIS we evaluate its performance. On the one hand, we measure its response time based on the aforementioned change scenarios, thereby determining values for realistic application scenarios (Sect. 5.1). On the other hand, we analyze the runtime complexity by means of the $\mathcal{O}$-notation in order to determine the scalability of the approach (Sect. 5.2).

### 5.1 Experimental Evaluation

We evaluate the performance of R-PRIS by taking dedicated measurements for its three main phases: (phase 1) cloud monitoring, including sending monitoring

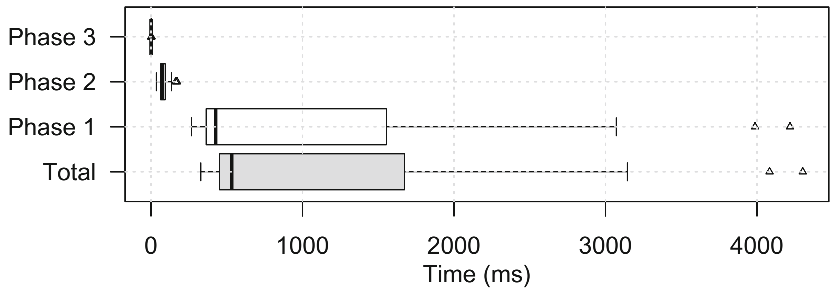


**Fig. 6.** Response times per phase (on Amazon Web Services)

**Table 3.** Measurement per phase and total

| Phase | Average (ms) | Median (ms) | Minimum (ms) | Maximum (ms) |
|---|---|---|---|---|
| 3 | 1 | 1 | 1 | 3 |
| 2 | 86 | 76 | 35 | 172 |
| 1 | 113 | 462 | 267 | 4218 |
| Total | 1119 | 532 | 329 | 4032 |

data to the R-PRIS server and resolving host geo-location using third-party service; (phase 2) runtime model update; and (phase 3) policy check.

We use the prototypical implementation and the change scenarios introduced in Sect. 4. The scenarios include violations (i.e., best and typical cases for reachability analysis) and non-violations (i.e., worst cases for reachability analysis as the entire graph has to be traversed). During the execution we repeated the four change scenarios five times each. Thus, we measure 20 response times for each phase during the experiment. The results are shown in Fig. 6 and in Table 3.

We consider the measured worst case response time of around 4 seconds promising. However, in order to speed up the response further, we analyzed the time consumption in phase 1 as this phase predominates the overall duration.

Further investigation showed that the comparatively high response times as well as the outliers in phase 1 stem from employing a third-party service for resolving the host-geolocation. Thus, we choose an alternative cloud infrastructure that offers 'built-in' geo-location APIs. We have repeated the measurements for phase 1 on the Azure cloud[9], which offers a REST API for geo-locating virtual machines. The measured results for phase 1 on Azure are: $avg$ = 488 ms, $med$ = 441 ms, and $max$ = 924 ms. This shows a clear reduction of the overall worst case response time, but exhibit low impact on the $med$ value. The reason might be that the quality of the built-in geo-location API is generally more stable than the third-party service.

[9] http://azure.microsoft.com.

A further performance improvement may be possible if cloud management APIs were available that emit monitoring events as soon as cloud migrations are triggered. In this case, the time that it takes for performing cloud migration or starting a new virtual machine (which may well be in the order of tens or hundreds of seconds [11]), may be used to run the policy checks and stop the migration if it turns out to violate the policy.

### 5.2 Runtime Complexity Analysis

The above experiments delivered concrete response times for a cloud service that was deployed on a small cloud cluster. To assess the scalability of R-PRIS with respect to performance, we perform a complexity analysis of the approach.

The worst case runtime complexity for depth first search is given by $\mathcal{O}(|V| + |E|)$. As the approach visits the vertices of $V_R$ once at most (see Sect. 3.2.2), the worst case complexity of traversing $G_R^{t_i}$ is $\mathcal{O}(|V_R| + |E_R|)$. However, in our approach we perform two interwoven depth first searches. For every pair of adjacent vertices the typed graph is traversed, such that $|E_R|$ resolves to $\mathcal{O}(|E_R| \cdot (|V_T| + |E_T|))$. The overall worst case complexity of the policy check is thus given by $f(G_{typed}) \in \mathcal{O}(|V_R| + |E_R| \cdot (|V_T| + |E_T|))$.

The actual complexity of $f(G_{typed})$ is quite low, when taking knowledge about the application context into account. The data classifications and their relations represented in $G_T$ are derived from standards. Thus, $|V_T|$ and $|E_T|$ have low values (in the order of 10) and, more importantly, are considered to be constant (in case of FIPS, $|V_T| = 3$ and $|E_T| = 2$). According to the $\mathcal{O}$ simplification rules, constants are to be neglected when analyzing worst case complexity. Thus, the checking function's complexity reduces to $f(G_{typed}) \in \mathcal{O}(|V_R| + |E_R|)$ and scales linearly with the size of the runtime model.

## 6 Related Work

In this section we discuss how R-PRIS relates to existing privacy checks of cloud services and existing runtime model approaches.

**Privacy Policy Checks:** Research on privacy policy compliance of big data and cloud services mainly focusses on policy violation prevention and compliance monitoring. In policy violation prevention, privacy-by-design principles guide the design and implementations of privacy aware architectures. For instance, the approach presented in [6] equips cloud services with mechanisms that permit or grant data access after matching the client characteristics with privacy policies. However, changes of data geo-locations imposed by migration or replication of the component storing the data are not considered. Data transfers between the client services and further services are not covered. Transitive data transfers that may lead to policy violations thus remain undetected.

Compliance monitoring approaches such as [7, 10] employ cloud services audits during runtime. For instance, the approach in [7] correlates ping round-trip times of the audited service with geographical information. This allows to determine whether the service interface resides at specific geo-locations. However, the software components behind the service interfaces might be migrated or replicated, while the service interface remains at the same geo-location. For instance, Hadoop data nodes might be replicated to different locations while the request handling master node remains invariant. Further, policy checks may exploit elasticity events [1], by checking elasticity events against policies. Although this would enable local checks, it would not cover the analysis of data classification changes across software components.

**Runtime Models:** Runtime models provide global views on cloud services. Behavioral runtime models utilize, e.g., sequence-models [12], workflow models [9,15], and Markov-chains [5]. These models include activities and interactions of the reflected applications but do not provide information about computing nodes, their geo-locations, and the processed data. In contrast, architectural runtime models, e.g., [2,8], combine behavioral aspects of the system with structural information. These models do not provide information on the geo-location, processed data, and changes of data classifications. However, our runtime model approach [17] provides the required information as being designed for supplying the policy check with the necessary reflections of real world systems.

## 7 Conclusion and Future Work

We addressed the challenges involved in checking the compliance of big data cloud services against data geo-location policies. In particular, we have addressed the problem of considering different data classifications in order to avoid false positive violations. The main ideas underlying our approach were (1) using typed runtime models that reflect cloud services and data classifications, as well as (2) exploiting efficient reachability analyses on these runtime models to detect policy violations. Our proof of concept implementation and the experimental evidence indicates that the proposed approach is able to correctly identify policy violations with reasonably fast response times. In future work, we plan to investigate the applicability of R-PRIS to a real life example. Further, we plan to complement the approach with pro-active policy violation detection. To this end, we envision the assessments of adaptation plans (e.g., expressed in terms of prescriptive runtime models) before their execution.

**Acknowledgements.** This work was partially supported by the DFG (German Res. Found.) under Priority Programme “SPP1593” (grant PO 607/3-1).

## References

1. Aceto, G., Botta, A., de Donato, W., Pescapè, A.: Cloud monitoring: A survey. Comput. Netw. **57**(9), 2093–2115 (2013). http://www.sciencedirect.com/science/article/pii/S1389128613001084

2. Brosig, F., Huber, N., Kounev, S.: Automated extraction of architecture-level performance models of distributed component-based systems. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE) (2011)
3. Chen, H., Chiang, R.H., Storey, V.C.: Business intelligence and analytics: From big data to big impact. MIS Q. **36**(4), 1165–1188 (2012)
4. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag New York Inc., Secaucus (2006)
5. Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G.: Model evolution by run-time parameter adaptation. In: 31st International Conference on Software Engineering (ICSE) (2009)
6. e Ghazia, U., Masood, R., Shibli, M.: Comparative analysis of access control systems on cloud. In: 2012 13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel Distributed Computing (SNPD) (2012)
7. Gondree, M., Peterson, Z.N.: Geolocation of data in the cloud. In: Proceedings of the third ACM Conference on Data and Application Security and Privacy, CODASPY 2013. ACM, New York (2013)
8. Huber, N., Brosig, F., Kounev, S.: Modeling dynamic virtualized resource landscapes. In: Proceedings of the 8th International ACM SIGSOFT Conference on Quality of Software Architectures (2012)
9. Ivanović, D., Carro, M., Hermenegildo, M.: Constraint-based runtime prediction of SLA violations in service orchestrations. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) Service Oriented Computing. LNCS, vol. 7084, pp. 62–76. Springer, Heidelberg (2011)
10. Juels, A., Oprea, A.: New approaches to security and availability for cloud data. Commun. ACM **56**(2), 64–73 (2013)
11. Mao, M., Humphrey, M.: A performance study on the VM startup time in the cloud. In: 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), pp. 423–430
12. Maoz, S.: Using model-based traces as runtime models. Computer **42**(10), 28–36 (2009)
13. Park, S., Chung, S.: Privacy-preserving attribute distribution mechanism for access control in a grid. In: 21st International Conference on Tools with Artificial Intelligence (2009)
14. Rausch, A., Reussner, R., Mirandola, R., Plasil, F. (eds.): The Common Component Modelling Example (CoCoME). LNCS, vol. 5153. Springer, Heidelberg (2011)
15. Schmieders, E., Metzger, A.: Preventing performance violations of service compositions using assumption-based run-time verification. In: Abramowicz, W., Llorente, I.M., Surridge, M., Zisman, A., Vayssière, J. (eds.) ServiceWave 2011. LNCS, vol. 6994, pp. 194–205. Springer, Heidelberg (2011)
16. Schmieders, E., Metzger, A., Pohl, K.: A runtime model approach for data geolocation checks of cloud services. In: Franch, X., Ghose, A.K., Lewis, G.A., Bhiri, S. (eds.) ICSOC 2014. LNCS, vol. 8831, pp. 306–320. Springer, Heidelberg (2014)
17. Schmieders, E., Metzger, A., Pohl, K.: Architectural runtime models for privacy checks of cloud applications. In: Proceedings of the 7th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems, PESOS 2015, ACM, New York (2015)
18. Szvetits, M., Zdun, U.: Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. Softw. Syst. Model., Dec 2013