Automated Cloud-to-Cloud migration of distributed software systems for privacy compliance

Master's Thesis of

B.Sc. Philipp Weimann

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

Reviewer:	Prof. Dr. Ralf H. Reussner
Second reviewer:	Jun.-Prof. Dr.-Ing. Anne Koziolek
Advisor:	Dr. rer. nat. Robert Heinrich
Second advisor:	Dipl.-Inform. Kiana Rostami

01. March 2017 – 31. September 2017

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

PLACE, DATE

Please replace with actual values

. . . .

(B.Sc. Philipp Weimann)

Abstract

English abstract.

Zusammenfassung

Deutsche Zusammenfassung

Contents

List of Figures

List of Tables

1. Introduction

1.1. Motivation

Over the last years, cloud computing has become more and more popular. This is a result of its business advantages, the continuing usage simplification and the abundance of own data centres. Netflix, for example, closed all its owned data centre in 2015 and moved completely to Amazons AWS.[3] As a result of this trend, the expected revenue in 2016 is about 200 Billion $.[10] The high degree of elasticity, automation, self-service, flexibility in payment and, as a result, lower cost are only some of the many advantageous points of cloud computing.[1]

However, many – especially European - companies fear dependencies, loss of data control, industrial espionage or privacy law violations. Precautionary measures like encryption or data splitting - among data centres - is not all there is to prevent a PR disaster, due to complex EU Data Protective Regulations or the US HIPAA act. To tell the whole truth, the complexity, the hidden services usages and the therefore resulting unawareness of many EU citizens (and law enforcement institutes) makes it very unlikely for current law violators to face any consequences. Nevertheless, citizens start to be more aware and the law enforcements point of attention tends to change quickly, like the Max Schrems' "Facebook Process" showed.[6] Further, in 2018 the "reform of EU data protection rules" will come into effect, which states severe punishments for privacy violations.[2] As a result, in the future entrepreneurs, companies and institutions need to be more aware of privacy compliance to prevent major monetary and reputation losses.

The EU General Data Protection Regulations sets the legal boundaries for European companies. It defines multiple regulations about data handling, data trading, personal advertising and more. One sets the boundaries for personal data processing and saving. It states for example, that the processing of personal data is only allowed in data centres inside the EU or certain certified countries with equivalent privacy laws. As a result, software systems require a pre-deployment law compliance checking, considering especially the hosts geo-locations. The problem comes to a head with the ease of migration of whole cloud services during runtime with next to no downtime. With this in mind a potential privacy violation could occur even after the initial deployment was law compliant. This requires a non-stop observation of the applications geo-location and automatic, law compliant redeployment onto other cloud providers.

1.2. Problems

...

1.3. Goals

- **Monitor**: How can privacy violations be monitored and what information are required for an efficient privacy policy violation detection?
- **Analyse**: How to analyse the model for efficient privacy violation detection?
- **Plan**: How can the runtime model and analysis results be used to reacquire policy compliance?
- **Execute**: How can the plan automatically be executed and policy compliance established?

1.4. Outline

…

2. Foundations

In this chapter we introduce applications and principles, this thesis is based on. This introduction aims for a general understanding. Some aspects my be discussed in more detail in the corresponding section of this thesis.

2.1. MAPE-K loop

MAPE-K or MAPE was first introduced by IBM for automatic computing and later discussed in the context of self-adaptive systems. A MAPE system is usually a stand alone application, which is specially build for optimizing and adapting a monitored system. MAPE-K is an acronym, consisting of the first letters of the loops stages: Monitor, Analyse, Plan, Execute and Knowledgebase. These stages are sequentially ordered in a pipeline structure, each one has a well defined task:

- **Monitor**: Collects, aggregates, filters and correlates information about a monitored system.
- **Analyse**: Performs (complex) data analysis and reasoning on the monitored data. The analysis is often supported by data from the knowledgebase. If changes are required, a change request is passed to the plan function.
- **Plan**: Determines what kind of changes are required and develops a transformation which adapts the monitored system towards the desired state.
- **Execute**: Executes the transformation calculated during the planning phase.
- **Knowledgebase**: Additional or advanced information that are shared among all stages.

The monitored system runs independently from the MAPE application. However, the desired monitoring information are usually explicitly provided via specially designed APIs, intefaces or probes.

2.2. Palladio Component Model

The Palladio Component Model (short PCM) is an Architecture Description Language (ADL) for component based software, originally designed to enable software architects to run pre-implementation performance analysis. The Palladio Simulator reports on "performance bottlenecks, scalability issues, reliability threats, and allows for a subsequent optimisation." The PCM is composed of several sub-models which depend on another. Each model represents a certain aspect of a component based software:

- **Repository Model**: Defines Components with required and provided interfaces. Interfaces include function signatures.
- **System Model**: Defines the complete software system, by connecting components defined in the repository model.
- **Usage Model**: Defines process workload, based on the systems interfaces.
- **Resource Environment Model**: Defines available host environments with its provided performance.
- **Allocation Model**: Defines the deployment of system components onto the provided hosts.

The separation of concern enables the system architect to manage the complexity of even bigger software systems and still gain meaningful results from the performance simulation.

Since its initial release the Palladio Component Model was adapted and used in several research fields alongside the performance prediction like automated Dataflow Analysis and Application Monitoring. Due to its explicit representation of the software architecture and flexible component-host-mapping it is perfectly suited to model distributed cloud systems. Although, PCM was not designed to be used as a runtime model, it has proven to be suited for this task due to its versatile model elements.

2.3. Kieker

Kieker is a software system monitoring application with the goal of retrieving runtime information for performance evaluation, (self-)adaptation control and many other tasks. Kieker gains these information from the designated software by weaving probes into the systems source code during pre-compilation. Each probe has designated purpose and gathers data accordingly, for example hardware utilization, stack trace or host geolocation. Kieker uses event-based probes, as well as periodic-based probes.

2.4. iObserve

iObserve is based on Kieker and therefore also a software monitoring application. However, iObserve uses the monitored information to update the systems Palladio model during execution, making it a runtime model. Further, iObserve and the extended Kieker version are deigned to support distributed cloud systems. Key features are the transformation form the gathered information onto the model update. Using the stack trace information the PCM usage model can be updated and more precise performance simulations created.

Currently, iObserve goes as far as updating the model, representing the first stage "Monitoring" of the MAPE-K loop. iObserve uses the Teetime framework, a pipeline-filter-framework with signal based state invocation.[4]

2.5. PerOpertyx

"PerOpteryx is an optimization framework to improve component-based software architecture". Optimization uses model-based quality prediction techniques. Starting from an input model, the framework generates multiple pareto optimal alternative deployments, based on given simulation and alternation algorithms. PerOpteryx make architecture adjustments via multiple approaches like alternating components multiplicity, runtime parameters or changing component allocations. The Pareto optimality models are calculated through multiple iterations via a series of stages. Initially a variance of candidates is created though evolutionary algorithm and random generation/mutation. In the next step, the candidates get analysed for the desired quality marks and ranked accordingly. The iteration concludes with the elimination of poorly performing candidates. The framework terminates with a cost rating of all Pareto optimum candidates.

3. Related Work

3.1. Application Monitoring

The monitoring of software systems is common task in many research fields. Automated data-flow analysis, software profiling and hardware utilization are only a small selection of groups, using this term. In the following, we use application monitoring in the sense of online extraction of runtime data form a (distributed cloud) application for architecture optimization.

R-PRIS (Section 3.2) and Kieker are application monitoring frameworks. While iObserve uses Kieker to extract the desired information, R-PRIS is independent from other programs. Neither of them uses a meaningful architecture description language (ADL) to process and store the gathered information. iObserve however gathers the transmitted data, processes them and stores them into a PCM model, enabling all sorts of PCM-based applications to use the gathered information.

gather more details & Refs

3.2. Privacy Analysis

R-PRIS is a monitoring and analysing tool for distributed cloud systems. Like iObserve, R-PRIS updates a runtime model by monitoring the cloud systems. During the analysis phase the model is checked for (potential) privacy violations.

R-PRIS combines push-based heartbeat monitoring with event processing, and graph grammars for efficiently updating those models.[8]

Add more details?

R-PRIS uses a formal specification for geo-location policies. These consists of data classification S, data content types T and geo-locations L. Every specified policy $p = (S, T, L)$ is forbidden. During privacy analysis R-PRIS checks whether a privacy protected information can be accessed from an non-privacy compliant location. This can be transformed into an st-connectivity problem, a standard problem in graph theory and analysis. Based on the runtime model (Figure 3.2) - with its meta-model (Figure 3.1) - R-PRIS performs a reachability check.[9]

In terms of software, R-PRIS searches for communication paths in the distributed system, which can potentially transmit personal data to a non-privacy compliant geo-location. In order to detect these communication paths a policy p must be specified, representing exactly this case, which however doesn't necessarily communicate private data. As a result, a lot of policies have to be specified, which as a result prohibits many potentially harmless communication paths.

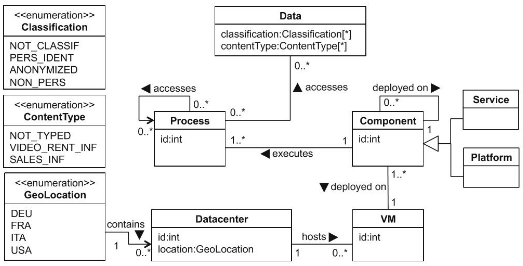

Figure 3.1.: R-PRIS meta-model

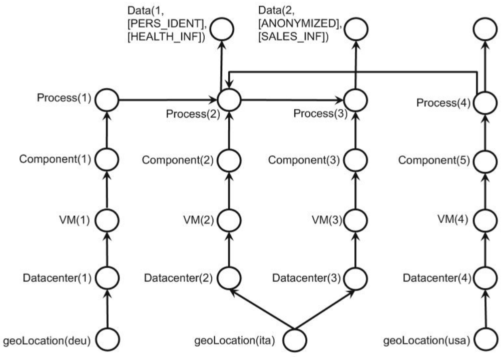

Figure 3.2.: R-PRIS runtime model

Based on their runtime model, Schmieders et al. identified four relevant migration-cases and extracted six required informations to detect a policy violation[9]:

#	**Required information to carry out runtime check**
R1	Interactions of two components
R2	Access of components to locally stored files
R3	Meta-information of stored or processed data
R4	Information on component deployments on physical resources
R5	Geo-location information of physical resources
R6	Explicit or implicit information on transitive data transfers

Table 3.1.: R-PRIS information for runtime privacy checks [9]

3.3. Data-flow Analysis & Rights Management

(Access) Rights Management, like the Bell-LaPadula Model or Role-based access control, are fundamentals in our modern information society. These systems restrict or allow access on certain entities with the intention of information protection. The fundamentals are well researched, so research currently is focused on resource and time efficient rights management in large scale systems like companies.

Data-flow Analysis is a hot research topic due to the omnipresence of cloud services. Research is based around the core task of investigating, predicting and tracing the path of data inside a program. Research can be classified among multiple dimensions. The two most imported are static analysis vs. runtime analysis. And the level of abstraction, which ranges from code line based to component based grain.

For our purposes we need automated data-flow analysis on method signature level, to determine the violation of privacy regulations. This research is still in its fledgling stages and therefore not suited for applications with our designated level of complexity.

Add Refs

3.4. (Privacy) Policy Check

Most research in this field focuses on prevention of policy violation. “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.”[9]

As mentioned in Section 3.2, R-PRIS is searching for potential access violations in the application model, by using a st-connectivity analysis.[9][8] This approach is overestimating the privacy aspects by not including which kind of data are actually communicated between components and geo-locations. This makes it impractical for many business applications due to likelihood of allowing only save-considered components deployment.

3.5. Automated Model Optimization & Modification

The research field of model analysis based performance optimizer can be roughly divided into two sections. First, the rule-based approaches, which apply a predefined rule, based on the detected problem, onto the system model. Second, metaheuristic-based approaches, which use a generic framework and evolutionary algorithms for multiple arbitrary quality criteria.[7]

PerOpteryx (Section 2.5) is a metaheuristic-based approach. However, PerOpteryx does not consider a hosts geo-location during its optimization process. This can be changed by adding an allocation constraint, preventing privacy violating deployments.

3.6. Automated Cloud Migration

Since the start of cloud computing there has been plenty of research on how to migrate regular on premise applications and software into the cloud. Either software is cloud-enabled in the most automatic fashion possible or the software is cloud-native, meaning specially developed or redesigned, by developers, for running inside the cloud. While there has been good progress semi-automatically cloud-enabled software, the field of migrating cloud applications form one cloud provider to another is just beginning. One of the main issues is resulting in provider individual Cloud-APIs. Current, state of the art is the "Docker" or container-technology, which wraps the application like a VM and is suitable for many cloud provider. Nevertheless, many cloud provider offer special purpose solutions, where a docker solution is not viable. The technology side of cloud to cloud migration will be left out in this thesis. [5][1]

4. Privacy Concept

Many say: Data is the new oil and the most valuable resource there is. This shows us how important it is that we keep in control of our personal data. In order to achieve this, many players have to fulfill their obligations. On the one hand, the personal awareness of every user himself to only communicate the required and necessary information. On the other hand, the data handling institutes duty to guarantee legal compliance to laws like the EU's general data protective regulations. While we can't act for the individual, we can provide tools and rules for institutions to help with legal compliance.

4.1. General Concept

To achieve privacy compliance, we need to get knowledge about which software component gets in contact with what kind of data. This is usually called a data-flow analysis. This task got especially important, since distributed cloud system are a reality and data saved on "on premise" servers are becoming increasingly rare.

A growing and continuous research field is exploring and developing approaches for automatic data-flow analysis. However, these are still very limited and are therefore not yet suited for practice. (See Section 3.3) Due to these limitations, we need manual annotation of components. Because of the diversity of data types and purposes, we decided to categorize these into three well defined categories:

- **Type 0: Personal Information**: Data stand in direct or indirect relation to personal information. This is independent from encryption or pseudonymization.
- **Type 1: Personally Identifiable Information**: Data do not contain personal information. However, by combining, fusing or analysing data sets the personal data could be reconstructed for complete or partial personal information.
- **Type 2: Anonymous Data**: Data don't contain any personal information. Even by extensive data analysing no direct or indirect personal data can be extracted.

Why are we using three categories instead of just two? In many cases data don't contain any direct or indirect link onto private data, however still contain indicators onto private data. For example an online shop wants to analyse, which products usually get ordered together. The orders get anonymized by removing the customer and the shipping address. Nevertheless, the time-stamp is required to get a timed evaluation factor. This data are not personal. However, combining and evaluating these with user-login-times, also non-personal data, privacy relevant data can be extracted. This also disqualifies them for Type 2, completely anonymous data.

To summarize, we use a manual, categorized annotation approach to identify a system components privacy level. Based on this privacy level we can check, whether a system deployment is privacy compliant.

4.2. Deployment Constraints

How can we guarantee legal compliant distribution? As mentioned in Section 1.1, personal data of EU citizens are only allowed to be processed, transferred or saved inside EU countries [...]. We argue that the following constraints, combined with correct manual annotation, are sufficient to accomplish this:

1. Deploy Type 0 components only inside save locations.
2. Deploy Type 2 components anywhere.
3. Type 1 components can be deployed anywhere.
4. Only deploy Type 1 components together in an un-save geo-location, if they receive their information (transitively) from the same Type 1 component.

Rule 1 & 2 don't need any further explanation. Rule 3 states that Type 1 components can be deployed anywhere. This is due to the fact that Type 1 data should not contain any personal information.

Rule 4 however limits this deployment. This constraint is necessary, because the combination of multiple Type 1 data streams could lead to privacy relevant information. If the data streams, however, have a common type 1 component as data source, the deployment can be considered privacy compliant. Note that data streams with type 2 components can be ignored, since - by definition - they don't get in contact with any privacy relevant data. Figure 4.1 illustrates the different base scenarios applying to Rule 4:

Figure 4.1.: Deployment Examples

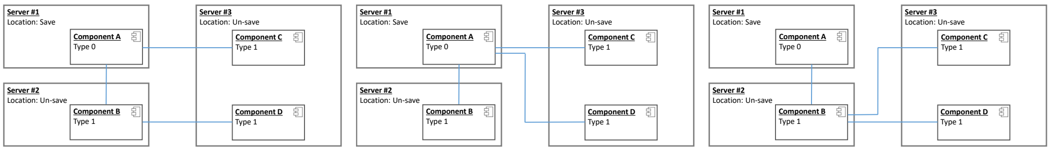

Figure 4.2.: A

Figure 4.3.: B

Figure 4.4.: C

The deployment shown in Figure 4.2 is illegal. Server#3 contains components with data streams from two different components, where one is a type 1 and one is a type 0 component. Even though they have a communication path, Component A could send different information to Component B and C.

Deployment B (Figure 4.3) is also illegal. Component C and D share the same data-flow origin, this however is marked as type 0. The same as befor, the combination of data on Component C and D could lead to privacy relevant informations.

The final example (Figure 4.4) is a legal deployment. Component C and D share the same source component, which already only contains type 1 data. This means, Component C and D can only contain, even through data combination and heavy data analysis, only non-privacy concerning information.

We are aware, that these are over-approximations towards legal compliance. Nevertheless, we argue, that these simple, high-level rules already help to identify illegal deployments and establish privacy compliant (re-)deployments.

4.3. Component categorization

Components can be very complex due to multiple communication partners and dozens of interfaces. In such cases it can be considered impossible, to keep track of every single information flow on every component. This shows us, that a component - as a whole - shouldn't be categorized by hand.

But a single data stream between two component is easy to understand and easy to analyse. In an component based software architecture, data exchange happens via component interfaces. And during system composition the software architect must be aware of what data he passes through an interfaces.

As a result, we decided to categorize each interface communication during system composition. The components privacy categorization is then derived by evaluating the components communication.

4.4. Information storage

We are using the Palladio Component Model (Section 2.2), which is just one of several Architecture Description Languages. Most ADLs share a comparable structure, which can differ, due to the designated purpose. We will describe the storage exemplarily on the PCM.

The runtime model is supposed to reflect every relevant information, concerning the models purpose. So, we need to store the components privacy level and the servers geo-location in the PCM model.

The geo-location belongs to a server, which is part of the resource environment model in PCM. The resource environment contains resource containers, which represents a server or virtual machine. So the resource container is the perfect place for storing the severs geo-location.

As mentioned in Section 4.3, we need to categorize the communication between two connected component interfaces. The PCM system model uses the Assembly Connector to connect two component interfaces. This is the optimal model element to store the data privacy level for the inter interface communication.

5. Overview

In this chapter we will give an overview on the system developed during this thesis and the according research. The system is massively extending iObserve, while keeping its original purpose, see Section 2.4 for the fundamentals. All extensions are made for accomplishing the goals, defined in Section 1.3. The extended iObserve is mostly referenced as *iObserve Privacy* during this thesis. iObserve Privacys architecture is a filter pipeline, where each filter represents one stage of the MAPE feedback loop (Section 2.1).

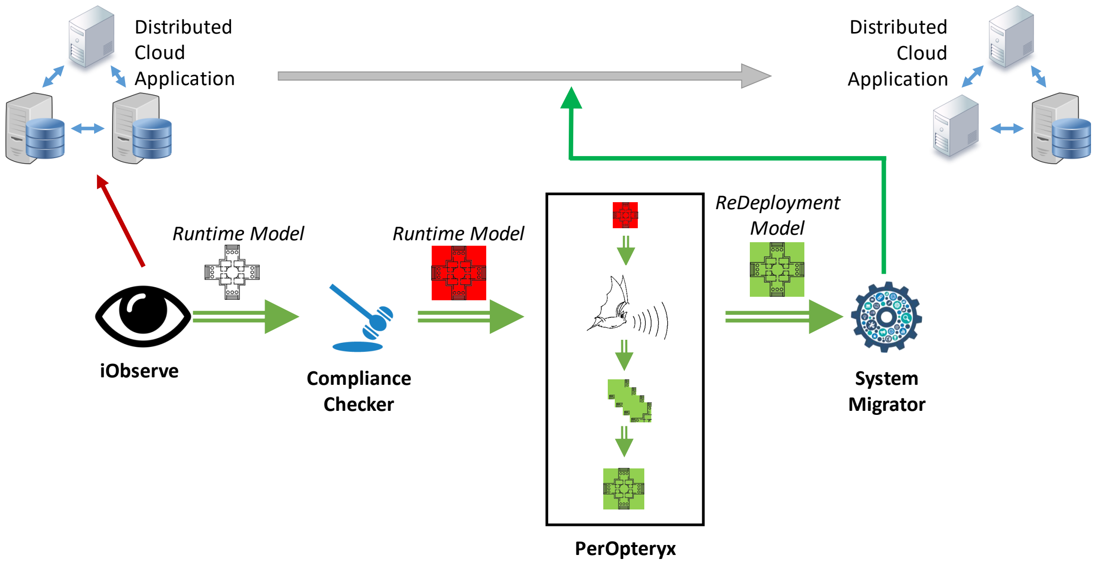

Figure 5.1.: iObserve Privacy pipeline

The initial step, monitoring, is provided by the original iObserve. However, it doesn't provide any information on the components geo-location. This extension are made directly in the original iObserve and Kieker. Upon detected geo-location changes, the runtime model gets updated and the next filter stage gets invoked. We determined the required data and proved a fitting transformation onto the PCM Privacy model.

The compliance checker, mostly referenced as *Privacy Analysis*, represents the analysis phase in the MAPE loop. It analyses the current runtime model for privacy violations. The fundamental principles were discussed in chapter 4. When a privacy violation is detected, the MAPE Planning phase gets activated.

The planning stages task is to find a privacy compliant redeployment model. For this job PerOpteryx is used. PerOpteryx (Section 2.5) is a complex model generation framework. PerOpteryx doesn't support deployment constraints and needs therefore extension.

Further, an output model needs to be selected as the final redeployment candidate, which gets transmitted to the final pipeline filter stage.

Add research accomplishments

The execute phase of the MAPE loop compares the redeployment model to the runtime model. Based on this, a migration plan gets calculated and finally executed. The execution exits of several parametrized function and script calls. After the stages execution the observed software system needs to be in privacy compliant state.

Add research accomplishments

@Robert: Add actual implemented pipeline UML with details, like SnapshotBuilder?

6. PCM Extension

6.1. General

As mentioned in Foundational Work the Palladio Component Model (Section 2.2) was designed for early architectural performance analysis. On this basis, the PCM was modified many times to fulfil many adjacent tasks. Contrary to a modification, we decided to extend the existing PCM meta-model. This enables us to keep compatible with the existing Palladio Models and other Palladio applications.

The standard Palladio meta-model is insufficient for privacy compliance analysis. To save the required information, an extension is the best practice approach. The extension was designed be as minimal invasive as possible, to keep the adoption effort for existing Palladio models to a minimum.

The concept was described in Section 4.4.

6.2. Implementation

The Palladio meta-model was modelled with the Eclipse Modeling Framework [EMF]. So, our extension, namely *PCM Privacy*, is also modelled with EMF and references the original Palladio meta-model. The required classes were extended in corresponding sub-packages.

As described in Section 4.4, we need to save a servers geo-location. The Resource Container was extended and the attribute *Geolocation* added. The extended element is named *Resource Container Privacy*. The geo-location itself is saved as an EInt Ecore type, a standard integer, encoded in the ISO country code (ISO 3166-1 [11]).

The *Assembly Connector Privacy* is the extension of the Assembly Connector and saves the data privacy categorization. The attribute is designed as an EEnum Ecore type, with the values Personal, Depersonalized, Anonymized. The value Personal is set as default.

Add PCM Privacy meta-model diagram.

7. iObserve Modification

8. Privacy Analysis

The Privacy Analysis represents the analysis stage in the MAPE-K feedback loop (Section 2.1). The goal is to check whether the runtime model contains any privacy violations. From the privacy concept, described in chapter 4, the privacy analysis consist of two major task. First, the correct privacy categorization of the software components. Second, the deployment evaluation, based on the deployment rules, defined in Section 4.2.

The remaining chapter is divided into a theoretical part (Section 8.1), a component categorization part (Section 8.2), a deployment evaluation part (Section 8.3) and the implementation part (Section 8.4).

8.1. Analysis Theory

The exclusive source of information for the privacy analysis is the systems PCM Privacy runtime model. For an efficient analysis we first need to identify the required information and re-factor them into a suited structure.

8.1.1. Required information

In the context of Privacy Analysis there is a minimum of two information which are required for privacy analysis:

#	**Required information for privacy analysis**
M1	Information on components privacy level
M2	Information on components geo-location

Table 8.1.: Minimal information for privacy analysis

Usually these information are not directly available. As a consequence other ones must substitute these, while containing the same information. A suitable substitution, differs based on the sources and their contained information.

The used PCM Privacy meta-model provides us with a categorization of the communicated information between component interfaces. This enables us to determine a component's privacy level based on the most critical communication with another component. As a result we can determine a components privacy level without knowing its exact purpose, analysing any inner processes or knowing the exact data-flow. So $M1$ gets substituted by information about inter interface communication and its privacy categorization.

In order to get an information equivalent of $M2$, a components host must be determined, as well as that hosts geo-location. The PCM Privacy model provides us with these information. However, it is spread over multiple models. As a result we require only four pieces of information (Table 8.2), compared to R-PRISs six pieces (Table 3.1).

#	**Required information to carry out runtime check**
I1	Interactions of two components per interface
I2	Information on component deployments on physical resources
I3	Geo-location information of physical resources
I4	Data Privacy categorization per interface communication

Table 8.2.: iObserves information for runtime privacy checks

8.1.2. Joining data streams

In Section 4.2 we elaborated on the danger of two *Personally Identifiable Information* (Type 1) data streams, from two sources, joining on a single server, could lead to personal, privacy relevant data. (Compare Section 4.2, Rule 4). This concept applies not only on the described deployment level, but also on the component categorization process.

While on deployment level, the information streams are not actively merged, this is a realistic possibility on the component categorization level. So the argument of applying an overestimation is not valid and this scenario must be taken seriously. In the following this special case will be refereed to as *joining data streams*.

8.2. Component categorization

The component categorization requires two tasks. The initial categorizing of every component and the "upcasting" of components with join data streams.

The initial data privacy level of a component is equal the components most critical communication level. Example Figure 8.1 shows the components data privacy level after the initial categorization phase.

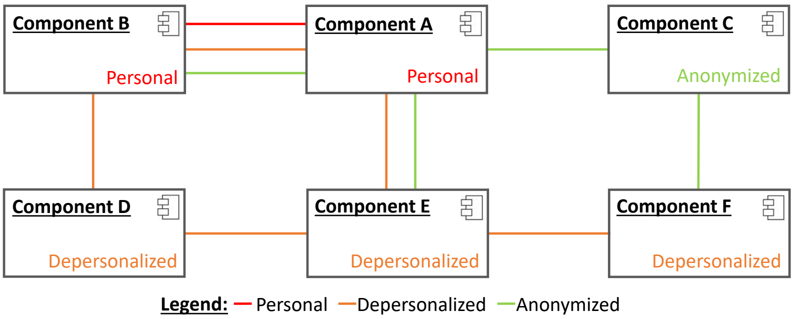

Figure 8.1.: Initial component categorization

The second phase analyses the system for joining data streams. Figure 8.2 continues the initial example), and shows the result of the second phase. In comparison to the initial categorization, component D and E get "upcasted". This is due to the fact, that component D and E have a connection onto two personal data sources (Component A and B).

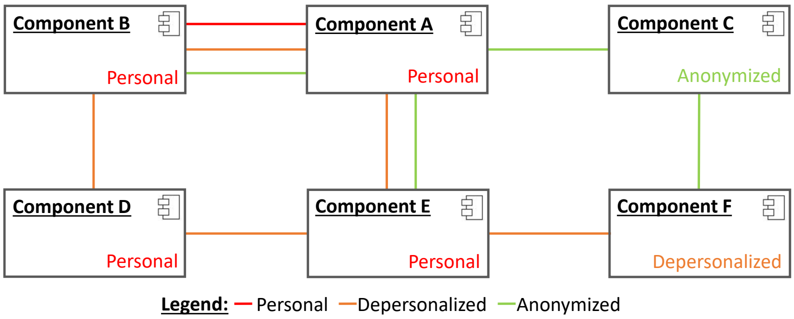

Figure 8.2.: Post categorization analysis - basic example

Two special cases are shown in Figure 8.3. So does component D get a personal data marking even though it has only one other component as data source. However, it has two individual connections to component B, which could contain a joining data stream, since B has a personal categorization.

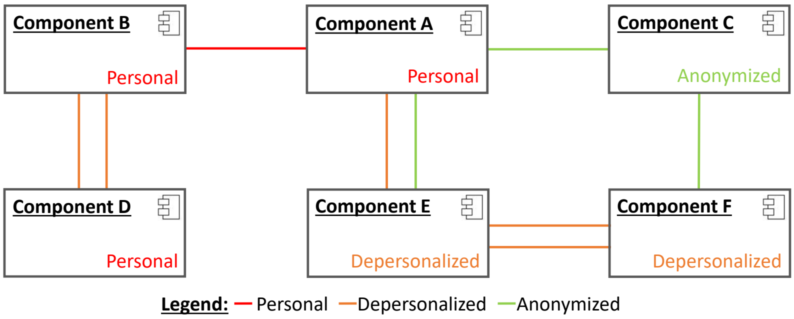

Figure 8.3.: Post categorization analysis - advanced example

Component F doesn't get an "upcast" since it can only contain privacy relevant data that are already present on component E. And component E has a depersonalized categorization. All anonymized categorized connections and components are ignored, since the don't contain any privacy related information.

8.3. Deployment analysis

The deployment analysis' goal is to find out whether the current deployment is privacy compliant. The rules for a privacy compliant deployment were described in Section 4.2.

Figure 8.4 shows an illegal deployment. The deployment of Component A and B is obviously valid, due to its deployment on a "save" geo-location. Component C and F are also legally deployed, since both components share a single communication edge onto privacy relevant information and the "joining data streams" situation does not apply. Server#2, however, hosts an illegal deployment. Component D and E have different single data sources edges and can therefore save, process or transmit data, which can combine to privacy relevant data.

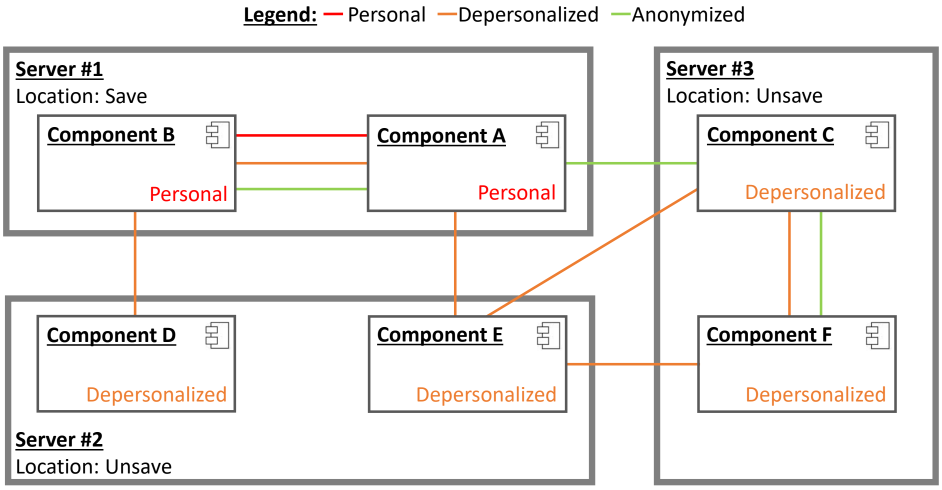

Figure 8.4.: Deployment analysis example

8.4. Privacy Analysis implementation

The PCM meta-model defines multiple models, each providing knowledge about a certain aspect of the target system (see Figure 8.5). This is not suited for an efficient privacy analysis and therefore requires an information preprocessing. So the implementation is spread over three steps:

1. build efficient data structure
2. categorize components
3. analyse deployment

8.4.1. Information preprocessing

In the first algorithm phase, all informations $I1$ to $I4$ are extracted from the different models. $I1$ and $I4$ is part of the System models Assembly Connector Privacy. Where $I1$ consist of the Providing Assembly Context and the Requiring Assembly Context. $I4$ is the Data Privacy Level. $I3$ is a field in the Resource Container, which represents a server in the Resource Environment model. The Allocation model contains $I2$ in Allocation Contexts, which provide a mapping of an Assembly Contexts on a Resource Containers.

After extracting all required information, the basic data privacy level for every component/Assembly Context is calculated by applying the most critical privacy level from the corresponding Assembly Connectors.

As last step of the preprocessing, the data are reassembled by constructing a sufficient graph (Figure 8.6). The graph is a simple, more direct representation of host-component-allocation structure from the PCM model. The graph contains two types of nodes: the DeploymentNode, a host representation, and the ComponentNode, a component representation. The data streams/interfaces are represented by the ComponentEdge:

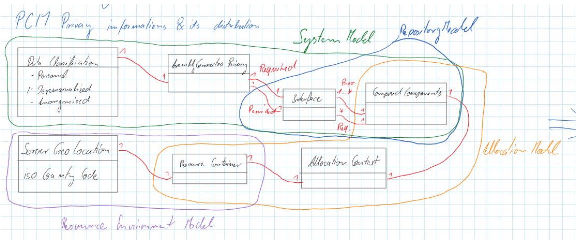

Figure 8.5.: PCM Privacy information spread

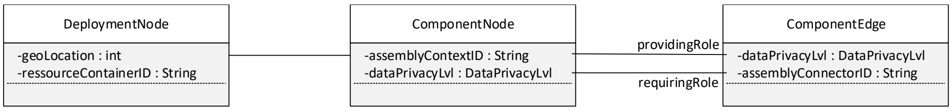
DeploymentNode
-geoLocation : int
-ressourceContainerID : String
ComponentNode
-assemblyContextID : String
-dataPrivacyLvl : DataPrivacyLvl
providingRole
requiringRole
ComponentEdge
-dataPrivacyLvl : DataPrivacyLvl
-assemblyConnectorID : String

Figure 8.6.: Graphs meta-model for Privacy Analysis

8.4.2. Component categorization implementation

The second phase of the privacy analysis algorithm finalizes the component categorization. As described in Section 8.2, joining data streams need to be found and fitting components' data privacy level corrected.

The algorithm (Algorithm 1) searches for depersonalised-marked connections from one personal categorized component to another. It uses a *deep search first* approach, while never using an edge twice. Note, that once a joining data stream is found, the involved components are appended to the list of personal components.

8.4.3. Deployment analysis implementation

The final privacy analysis phase is the deployment evaluation. The base analysis is very simple, since it simply checks whether every as personal categorized component is deployed on an as save considered geo-location. The a geo-location is considered as save, when it is contained in the save-geo-location list. When a server is located in an unsave geo-location and contains more then one depersonalised component, an extensive analysis for joining data streams has to be made. This extensive analysis is described in Algorithm 2.

The Algorithm works similar to Algorithm 1. Initially, it extracts all as depersonalised categorized components on the server. These components have to form a transitive hull and share a single depersonalised communication link to a single personal component. The algorithm uses a *deep search first* approach to traverse through the components. If a second link to a personal component is found or not every depersonalised component on the server is reached, the deployment is illegal.

Note, that anonymized categorized components and edges can be ignored during analysis. Also, a server won't contain any personal marked component since such a deployment would be automatically illegal due to the servers un-save geo-location.

Algorithm 1 Component categorization algorithm

```
List of Components components
Set of Edges usedEdges

procedure StartCategorization(List<Components> components)
    $personalComponents \leftarrow componentswithPrivacyLvl == PERSONAL$
    for all $personalComponent \leftarrow Components$ do
        clear( usedEdges )
        TraverseComponent( personalComponent )
    end for
end procedure

function TraverseComponent(Component component)
    $dePersonalEdges \leftarrow component$.GetEdges with $PrivacyLvl == DEPERSONAL$
    for all $edge \leftarrow dePersonalEdges$ do

        if usedEdges.Contains( edge ) then
            Continue
        else
            usedEdges.Add( edge )
            $edgeParnter \leftarrow edge$.GetEdgePartner( component )

            if $edgeParnter.PrivacyLvl == PERSONAL$ then
                return edgeParnter
            else
                $secondSource \leftarrow$ TraverseComponent( edgeParnter )
                if $secondSource \neq PERSONAL$ then
                    $component.PrivacyLvl \leftarrow PERSONAL$
                    components.Add( component )
                    return secondSource
                end if
            end if
        end if
    end for
    return Null
end function
```

Algorithm 2 Deployment analysis algorithm

```
Set of Components compToReach
Set of Edges usedEdges
Edge dataSourceEdge

procedure ExtensiveAnalysis(Server server)
    compToReach ← server.GetComponents with PrivacyLvl == DEPERSONAL
    dataSourceEdge ← Null
    startComp ← compToReach.GetAny
    clear usedEdges
    singlePersonalDataSource ←TraverseComponent(startComp)
    return singlePersonalDataSource AND compToReach.IsEmpty
end procedure

function TraverseComponent(Component component)
    compToReach.Remove( component )
    dePersonalEdges ← component.GetEdges with PrivacyLvl == DEPERSONAL

    for all edge ← dePersonalEdges do
        if usedEdges.Contains( edge ) then
            Continue
        else
            usedEdges.Add( edge )
            edgeParnter ← edge.GetEdgePartner( component )

            if edgeParnter.PrivacyLvl == PERSONAL then
                if dataSourceEdge == Null then
                    dataSourceEdge ← edge
                else
                    return False
                end if
            else
                singleDataSource ← TraverseComponent( edgeParnter )
                if singleDataSource then
                    return False
                end if
            end if
        end if
    end for
    return True
end function
```

9. Evaluation

...

9.1. First Section

...

9.2. Second Section

...

9.3. Third Section

...

10. Conclusion

10.1. Limitations

A cloud provider, like any other person or company, hast own objectives. In the most cases a profit maximization can be assumed as the primary goal. This can be interpreted in multiple ways, from law incompliant behaviour, SLA violations to premium prices for premium services. Nevertheless, in general the assumption stands, that Cloud Providers want to stay SLA and law compliant to avoid lawsuits and reputation loss. Based on this, we assume our providers to be law compliant.

Furthermore, we state that our providers don't have any own objectives. This means we don't have to fear any activates from the provider except providing his cloud services. As a result, we can deploy multiple Type 1 Data (Section 4.1) onto one data-centre, but on different (virtual) server, without considering geo-location constraints.

If these assumptions wouldn't be made, major implications would follow. First, immense security measures must be taken to encrypting any data transmitted, saved or computed by cloud services. The system would have to consider running on an "evil" machine with its own objectives. This is a hot research topic around the globe and not part of the goals tackled by this thesis. Secondly, even if the provider is law and SLA compliant, it is not possible, with the resources at hand, to consider all privacy laws for each country individually. We argue, that these assumptions are a fair trade-off between the reality of a reputable cloud provider and still gaining meaningful results.

Add cloud provider reveals geo-location via API

Modern software systems and distributed cloud applications in particular, are designed after separation of concern principle. Systems designed after this principle encapsulate cohesive functionality in a component. If a system ignores this basic design principle, it is possible that our approach does not detect a privacy violation. Since a component gets its data privacy level from the Assembly Connector, a component that saves personal data, but does not communicate them via an Assembly Connector can receive a false privacy level. An example is a component that receives personal information via an user interface and saves or processes them itself. We argue that such a monolithic system stands contrary to the fundamental idea of distributed cloud systems and will be therefore ignored in this further thesis.

Bibliography

[1] Tobias Binz et al. “Migration of enterprise applications to the cloud”. In: *it - Information Technology* 56.3 (2014). ISSN: 1611-2776. DOI: `10.1515/itit-2013-1032`.

[2] COMM/JUST/01. *Protection of personal data.* Brussels, 2011. URL: `http://ec.europa.eu/justice/data-protection/` (visited on 04/24/2017).

[3] David Chernicoff. *Netflix closes data centers and goes to public cloud.* London, 2015. URL: `http://www.datacenterdynamics.com/content-tracks/colo-cloud/netflix-closes-data-centers-and-goes-to-public-cloud/94615.fullarticle` (visited on 04/24/2017).

[4] Robert Heinrich. *iObserve.* Ed. by SDQ-Wiki. Karlsruhe, 2016. URL: `https://sdqweb.ipd.kit.edu/wiki/IObserve` (visited on 10/17/2016).

[5] Baskaran Jambunathan and Dr.Y. Kalpana. *Multi Cloud Deploymentwith Containers.* Ed. by International Journal of Engineering and Technology. (Visited on 11/04/2016).

[6] Julia Bähr. *Schrems’ jahrelanger Kampf gegen Facebook.* Ed. by Frankfurter Allgemeine Zeitung. Frankfurt am Main, 2015-09-23. URL: `http://www.faz.net/aktuell/feuilleton/medien/max-schrems-jahrelanger-kampf-gegen-facebook-13819522.html` (visited on 04/24/2017).

[7] Anne Martens et al. “Automatically Improve Software Models for Performance, Reliability and Cost Using Genetic Algorithms”. In: *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering.* Ed. by Alan Adamson et al. WOSP/SIPEW ’10. ACM, New York, NY, USA, 2010, pp. 105–116. ISBN: 978-1-60558-563-5. DOI: `10.1145/1712605.1712624`. URL: `http://www.inf.pucrs.br/wosp`.

[8] Eric Schmieders, Andreas Metzger, and Klaus Pohl. “Architectural Runtime Models for Privacy Checks of Cloud Applications”. In: *2015 IEEE/ACM 7th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems (PESOS)*, pp. 17–23. DOI: `10.1109/PESOS.2015.11`.

[9] Eric Schmieders, Andreas Metzger, and Klaus Pohl. “Runtime Model-Based Privacy Checks of Big Data Cloud Services”. In: *Service-Oriented Computing: 13th International Conference, ICSOC 2015, Goa, India, November 16-19, 2015, Proceedings.* Ed. by Alistair Barros et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 71–86. ISBN: 978-3-662-48616-0. DOI: `10.1007/978-3-662-48616-0{\textunderscore}5`. URL: `http://dx.doi.org/10.1007/978-3-662-48616-0_5`.

[10] statista.com. *Umsatz mit Cloud Computing** weltweit von 2009 bis 2016 (in Milliarden US-Dollar)*. Ed. by statista.com. 2016. URL: `http://de.statista.com/statistik/daten/studie/195760/umfrage/umsatz-mit-cloud-computing-weltweit-seit-2009/` (visited on 09/05/2016).

[11] Wikipedia, ed. *ISO 3166*. 2017-04-18. URL: `https://de.wikipedia.org/wiki/ISO_3166#ISO_3166-1` (visited on 05/04/2017).

A. Appendix

A.1. First Appendix Section

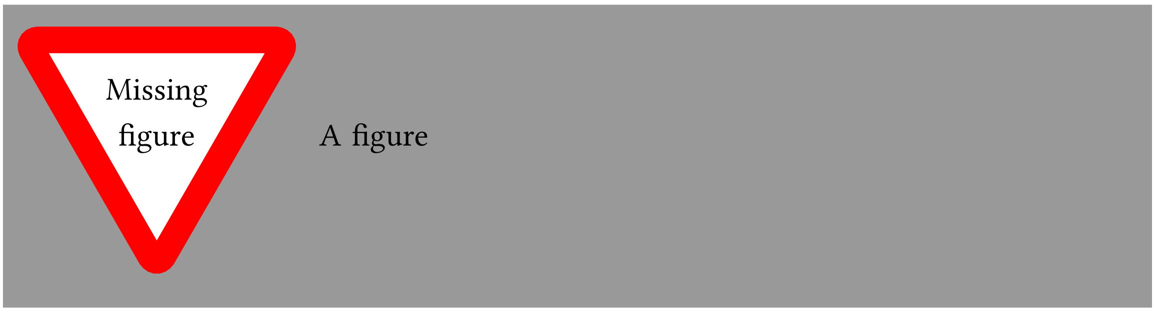

Figure A.1.: A figure

…