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 $[17]. 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 enough to prevent a public relations disaster, due to complex EU Data Protective Regulations[2] or the US HIPAA act[12]. 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[9]. 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 rule 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 case 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

To create such a privacy aware system adaptor, a couple of non-trivial problems need to be solved. The major ones will be outlined shortly, categorized after the MAPE-K feedback loop:

- **Monitoring**
  Acquiring and transforming geo-location information onto an architecture description language

- **Analysing**
  Privacy compliance analysis on a software architecture basis

- **Planning**
  Computation of constraint and privacy compliant redeployments

- **Executing**
  Technology independent, dynamic adaptation routine computation, execution and evaluation

1.3. Goals and Research Questions

This thesis' goal is to contribute a piece of software, that ensures continues privacy compliance, modelled after the MAPE-K feedback loop. Wrapped into this pipeline are several interesting research questions:

- **Monitor**: How can information, required for privacy violation detection, be monitored? How can we transform this information onto our architecture model?

- **Analyse**: How to analyse the model for privacy violation detection? And how good does this analysis scale?

- **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? How much human interaction is necessary?

Individual goals for the MAPE loop? Feels redundant, doesn't it?

1.4. Outline

The remainder of this thesis is structured as following: The thesis continues by introducing the foundations (chapter 2) and the related work (chapter 3) of this thesis. The main part starts with the privacy concept (chapter 4), leading into the system overview (chapter 5), followed by the big conceptual work packages: Palladio modification (chapter 6), iObserve extension (chapter 7), privacy analysis (chapter 8), PerOpteryx extension (chapter 9) and the system adaptation (chapter 10). The thesis closes with the evaluation (chapter 11) and finally the conclusion (chapter 12).

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.[7]

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.[14]

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.[15]

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.
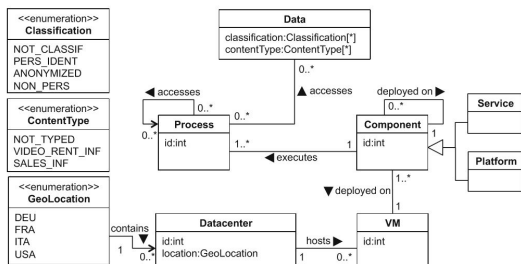
Figure 3.1.: R-PRIS meta-model
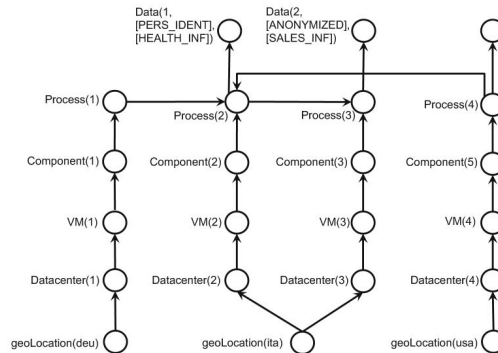
Figure 3.2.: R-PRIS runtime model

Based on their runtime model, Schmieders et al. identified four relevant migration-cases and extracted six required informations to detect a policy violation[15]:

#	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 [15]

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, as well as automated rights management on small, mobile devices [4].

Data-flow Analysis is a hot research topic due to the omnipresence of cloud services and mobile devices with rich data sources. Applications like *JOANA* [16], *TaintDroid* [5], *Privacy Oracle* [10] or *automated privacy instrumentation* [18] are only some of many applications and approaches around data-tracking, data-flow analysis and leak detection. However, nearly all of these approaches are using actual code or information rich models.

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

3.4. **Privacy Analysis**

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."[15]

As mentioned in Section 3.2, R-PRIS is searching for potential access violations in the application model, by using a st-connectivity analysis.[15][14] 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.[11]

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. [8][1]

4. Privacy Concept

Many say: Data is the new oil and the most valuable resource there 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

The EU General Data Protective Regulations clearly states, that data of EU citizens have to be saved and processed inside EU countries [2]. The only exception is a small set of countries with equal data protective laws. As a consequence, we need a simple data-flow analysis (Section 3.3) to know the data distribution in our software system. Frankly speaking, we must know which data is where in our system. This task got especially important, since distributed cloud system are a reality and data saved on "on premise" servers are becoming increasingly rare.

As mentioned in Section 3.3, the automated data-flow analysis on architecture level is still in its fledgling stages and therefore not suited for practice. As a compromise we decided on manual data tagging. To ease the data tagging and analysis process, we decided to use the common well defined categories [15]:

- **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.

We use three categories, because 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. [14][15]

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:

- **Rule #1**: Type 0 components must be deployed in a "save" geo-location.

- **Rule #2**: Type 2 components can be deployed anywhere.

- **Rule #3**: Type 1 components can be deployed anywhere.

- **Rule #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.
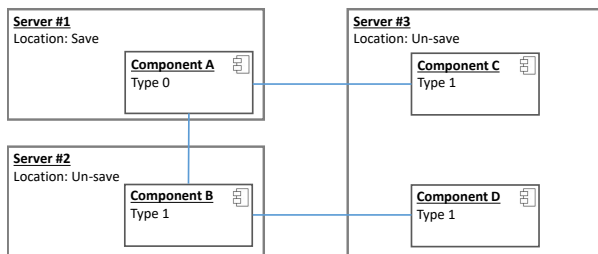
Figure 4.1.: Privacy violating deployment
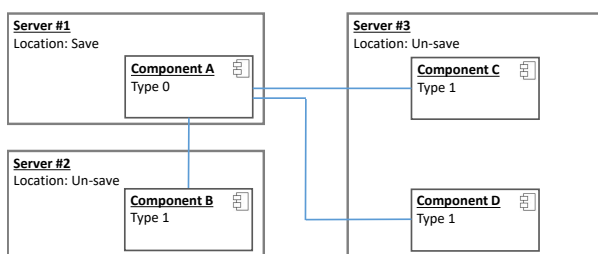
Figure 4.2.: Privacy violating deployment

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, Figure 4.2 and Figure 4.3 illustrate the different base scenarios applying to Rule 4.

The deployment shown in Figure 4.1 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. As a result, *Server #3* hosts two type 1 components, which don't contain privacy relevant data by themselves. However, after combining, could be privacy relevant. So, this deployment is illegal.

Figure 4.2 also shows a illegal deployment. Component C and D share the same data source, which is marked as type 0. The same as before, the combination of data on Component C and D could lead to privacy relevant informations on *Server #3*.

The final example (Figure 4.3) is a legal deployment. This is because, Component C and D share the same data source (Component B), which already only contains type 1 data. This means, Component C and D can just contain data which are already available on Component B. This means, even through data combination and extensive data analysis, no privacy concerning information can be extracted.

Figure 4.3.: Privacy compliant deployment

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.

We need to point out, that this categorization is only valid under the *Closed World Assumption* (CWA) [13]. Simplified, the CWA states, that if a system doesn't contain any information about a given statement, this statement is automatically wrong. Applied onto the privacy concept this means: The model contains all the information and information sources there are. Naturally, this assumption is wrong, since every component is connected to the internet and can access a nearly infinite amount of data. However, we need the CWA, in order to be able to make any statement about the systems privacy

compliance. This is due to the high abstraction level - the system architecture - we have an information source.

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).
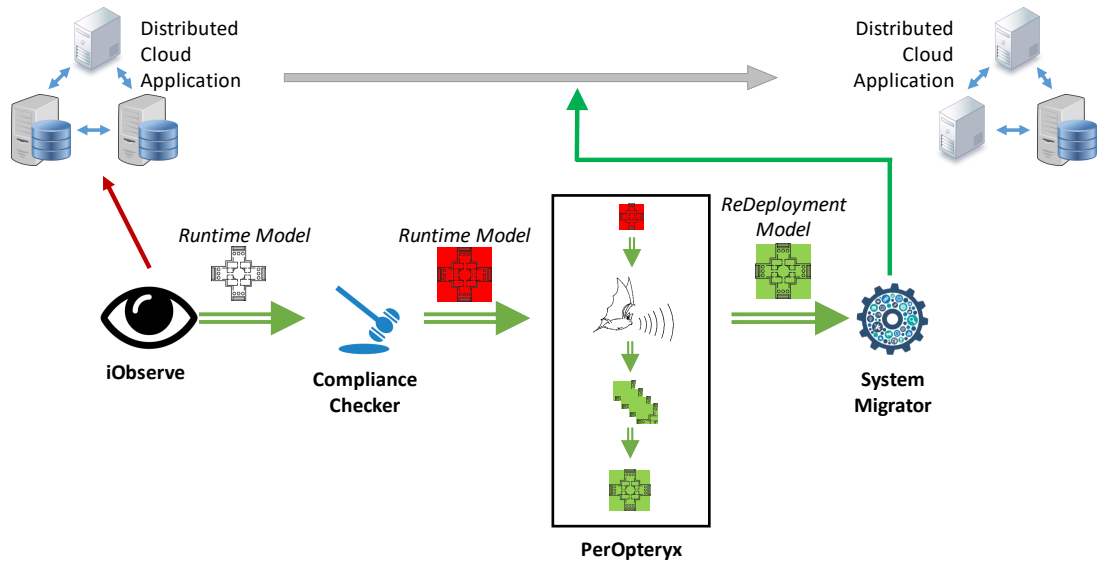
Figure 5.1.: iObserve Privacy pipeline

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 [20]).

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 Extension

iObserve was briefly introduced in Section 2.4. iObserve uses Kieker (Section 2.3) to gain real-time information about an observed (distributed) software system. iObserve transforms these information onto a Palladio Component Model. This model is referenced as *runtime PCM* or *runtime model*, since it reflects the actual observed software system during runtime. [7]

7.1. Kieker

describe ServerGeoLocation Probe and Listener
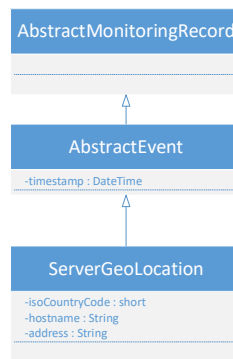
Figure 7.1.: Server Geo-Location Record

7.2. iObserve Privacy

iObserve uses the teetime framework. Its allows easy pipeline building by connecting matching input and output ports during runtime [19]. This mechanism is used by iObserve to invoke different transformations. Based on the received *Monitoring Record*, the according output port gets invoked and the matching transformation will be executed.

For the *ServerGeoLocation Record* (see Figure 7.1) another output port and transformation was added. The transformation uses the records host and address field to find the record sending Resource Container. The according Geo-Location attribute of the *Resource Container Privacy* is compared to the incoming geo-location and updated if needed.

iObserve has a well defined structure due to the teetime framework. We decided to keep this structure and extend it. Figure 5.1 shows the conceptual filter pipeline structure for our planned extension. One conceptual stage usually consists if several sub tasks. To

embrace re-usability and lose coupling, we decided to keep on using the teetime framework structure and compose a conceptual stage of several sub-stages. Figure 7.2 shows the general structure. The *Composistion Stage* functions as a wrapper for the conceptual filter, while the *Stages* I to III represent the sub-tasks.
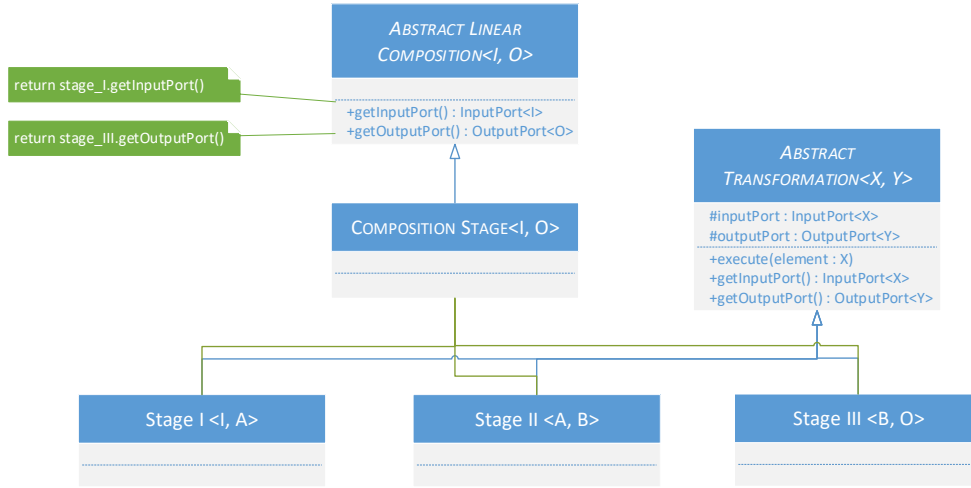
Figure 7.2.: iObserve Privacy Filter

This structure does not only allow easy restructuring of the conceptual pipeline and encapsulates the sub-tasks, it also allows for simple reuse and a clear separation of concerns. For the teetime framework, the filter stages looks like a series of linear connected stages, while the developer gets easy to handle packages. It is worth mentioning, that Stages linked to each other require matching data. The actual task executed, must be placed inside the *execute* method.

The extended iObserve contains the following conceptual filter stages: *Monitoring*, *Snapshot* (creates a copy of the current runtime model), *Privacy Analysis*, *Model Generation*, *System Adaptation* and *Evaluation*. In the remainder of this thesis the extended iObserve is referenced as *iObserve Privacy*.

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. The privacy concept, described in chapter 4, states that 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 minimal required information and substitute them, depending on the available sources.

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. Data-flow direction

Usually component interfaces are categorized into providing and required interfaces. Connections are made between a pair of required and provided interface of the same type. This suggest a certain data- and control-flow direction. This is a wrong assumption. While there are cases, when the control-flow can be derived from this structure, the data-flow is completely independent from this categorization.

For example a database interface usually only has provided interfaces. These interfaces allows the user to store and retrieve data from the database. Such a database interface normally contains getter and setter methods, which proves there is no data-flow direction for the whole interfaces. Note, that we are explicitly speaking of a whole component interface. Individual methods can have a data-flow direction.

Information passed through an interface are available on the providing and requiring component. This means, if an component connector got categorized as *Personal*, that information of this type are available in both components. These components need to get categorized accordingly.

8.1.3. 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 (see Subsection 8.1.2). Example Figure 8.1 shows the components data privacy level after the initial categorization phase.
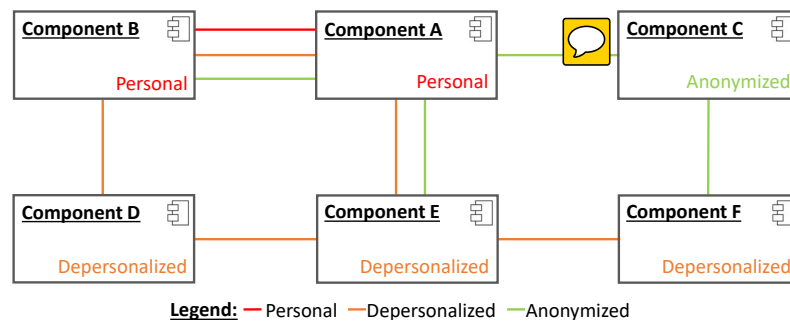
Figure 8.1.: Initial component categorization

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).
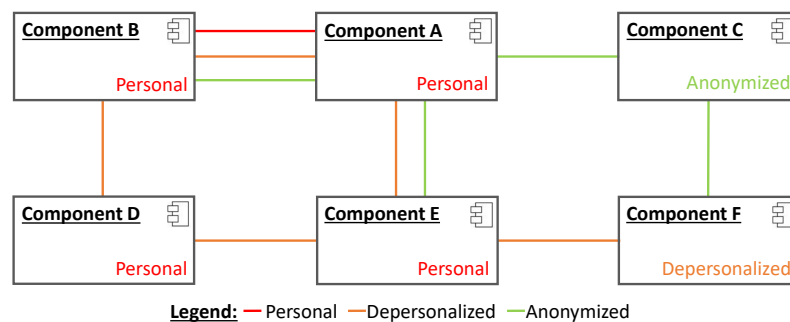
Figure 8.2.: Post categorization analysis - basic example

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.

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.
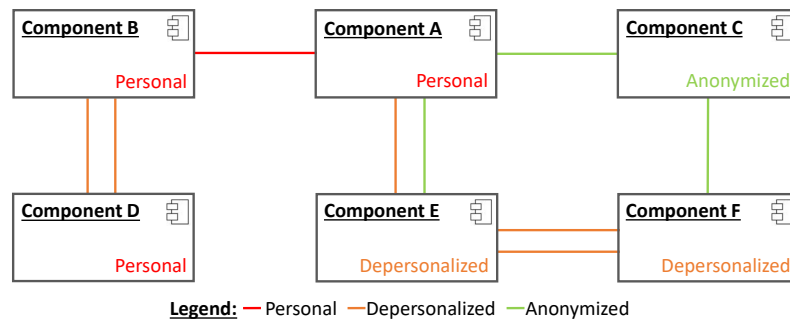
Figure 8.3.: Post categorization analysis - advanced example

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.
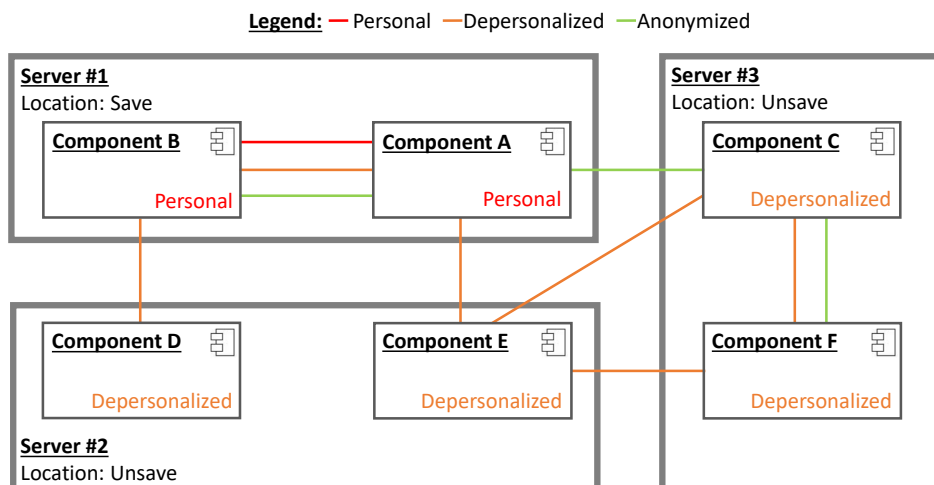
Figure 8.4.: Deployment analysis example

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

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 un-save 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

1: List of Components *components*
2: Set of Edges *usedEdges*
3:
4: **procedure** STARTCATEGORIZATION(List<Components> *components*)
5:     *personalComponents ← componentswithPrivacyLvl == PERSONAL*
6:     **for all** *personalComponent ← Components* **do**
7:         CLEAR(*usedEdges*)
8:         TRAVERSECOMPONENT(*personalComponent*)
9:     **end for**
10: **end procedure**
11:
12: **function** TRAVERSECOMPONENT(Component *component*)
13:     *dePersonalEdges ← component*.GETEDGES with *PrivacyLvl == DEPERSONAL*
14:     **for all** *edge ← dePersonalEdges* **do**
15:
16:         **if** *usedEdges*.CONTAINS(*edge*) **then**
17:             CONTINUE
18:         **else**
19:             *usedEdges*.ADD(*edge*)
20:             *edgeParnter ← edge*.GETEDGEPARTNER(*component*)
21:
22:             **if** *edgeParnter.PrivacyLvl == PERSONAL* **then**
23:                 **return** *edgeParnter*
24:             **else**
25:                 *secondSource ←* TRAVERSECOMPONENT(*edgeParnter*)
26:                 **if** *secondSource ≠ PERSONAL* **then**
27:                     *component.PrivacyLvl ← PERSONAL*
28:                     *components*.ADD(*component*)
29:                     **return** *secondSource*
30:                 **end if**
31:             **end if**
32:         **end if**
33:     **end for**
34:     **return** Null
35: **end function**

Algorithm 2 Deployment analysis algorithm

1: Set of Components *compToReach*
2: Set of Edges *usedEdges*
3: Edge *dataSourceEdge*
4:
5: **procedure** EXTENSIVEANALYSIS(Server *server*)
6:     *compToReach* ← *server*.GETCOMPONENTS with *PrivacyLvl == DEPERSONAL*
7:     *dataSourceEdge* ← *Null*
8:     *startComp* ← *compToReach*.GETANY
9:     clear *usedEdges*
10:     *singlePersonalDataSource* ←TRAVERSECOMPONENT(startComp)
11:     **return** *singlePersonalDataSource* AND *compToReach*.ISEMPTY
12: **end procedure**
13:
14: **function** TRAVERSECOMPONENT(Component *component*)
15:     *compToReach*.REMOVE(*component*)
16:     *dePersonalEdges* ← *component*.GETEDGES with *PrivacyLvl == DEPERSONAL*
17:
18:     **for all** *edge* ← *dePersonalEdges* **do**
19:         **if** *usedEdges*.CONTAINS(*edge*) **then**
20:             CONTINUE
21:         **else**
22:             *usedEdges*.ADD(*edge*)
23:             *edgeParnter* ← *edge*.GETEDGEPARTNER(*component*)
24:
25:             **if** *edgeParnter.PrivacyLvl == PERSONAL* **then**
26:                 **if** *dataSourceEdge == Null* **then**
27:                     *dataSourceEdge* ← *edge*
28:                 **else**
29:                     **return** False
30:                 **end if**
31:             **else**
32:                 *singleDataSource* ← TRAVERSECOMPONENT(*edgeParnter*)
33:                 **if** *singleDataSource* **then**
34:                     **return** False
35:                 **end if**
36:             **end if**
37:         **end if**
38:     **end for**
39:     **return** True
40: **end function**

---

9. PerOpteryx Extension

10. System Adaptation

In the system adaptation stage of the iObserve Privacy pipeline the current software system is modified to match the re-deployment PCM. This filer stage represents part of the planning and the complete execution phase of the MAPE loop (Section 2.1).

The remainder of this chapter is structured into adaptation planning and the actual adaptation execution. It closes with a look onto the implementation.

10.1. Adaptation Planning

The planning phases job is to calculate what actions are required to bring the observed software system into the state defined by the redeployment model. While the task is pretty clear, the available source of informations are quite uncertain.

There are multiple potential sources of information that can be used to calculate adaptation steps. For example the *Design Decisions* file used and modified by PerOpteryx (chapter 9). This file contains all choices made during generation of the redeployment model. These informations are a viable source for action computation, however create a strong dependency on PerOpteryx. This means, if PerOpteryx would be exchanged for another model optimization tool, the complete adaptation planning needs to be rewritten.

Another information source for the adaptation planning could be the close observation of the candidate calculation/generation. This could be achieved by logging decisions made by the evolutionary algorithm. When considering, that the starting point of an evolutionary algorithms is usually a given input, the modification steps could be traced and remodelled for the system adaptation. However, evolutionary algorithms usually don't take the shortest path onto their end result and also random mutations are an valid generation factor. This means, the results needs to be analysed and optimized, while also injecting observations probes. While being a good potential information source, the effort for post-generation analysis and the resulting dependencies make it a bad choice.

We decided to make a direct comparison of the runtime PCM and the redeployment PCM. This builds up no further dependencies and the shortest adaptation path can be found. However, the information preprocessing and the comparison algorithm can be more complex than the other options.

10.1.1. Adaptation Actions

Palladio models are independent from programming languages, technologies and other specifics. We decided to enforce this characteristic by defining technology independent actions. They contain the required information, without knowing anything about the used technologies.

Further, we designed a set of basic actions which allows us to transform any PCM into another. The actions can be grouped into two major disjunct subgroups: the *Assembly Context Actions* and the *Resource Container Actions*.

- **Assembly Context Actions**
    - Allocate Action
    - Deallocate Action
    - Migrate Action
    - Change Repository Component Action
- **Resource Container Actions**
    - Acquire Action
    - Replicate Action
    - Terminate Action

Assembly Context Actions reflect all model changes around a software component. Allocate, Deallocate and Migration Actions are self explaining. The Change Repository Component Action addresses the possibility to exchange a software component with an equivalent one. This can due to better fitting performance characteristics, while required and provided interfaces stay the same. As a result, the structure of the system model stays unchanged, but an encapsulated component gets exchanged for another one.

Resource Container Actions reflect changes around a virtual or physical server. Acquire and Terminate Actions don't need any explanations. A Replicate Action clones a server instance with its containing components.

10.1.2. Action Ordering

After all required actions are determined, an order must be established. We decided for the most simple but effective approach: "acquire, migrate, terminate." Meaning, first acquire the new servers, change the component allocations and finally terminate the unnecessary servers. This way no component get moved onto non-existent servers.

More details and thought on ordering!

10.2. Adaptation Execution

The adaptation execution is as straight forward as it gets. The actions get executed - in order determined by adaptation planning (Section 10.1) - by calling equivalent scripts, which wrap the technological side of the action. The technological implications are not considered by this thesis and are therefore not further discussed.

Ref to Pöppke?

More details, when scripts & implementation are rdy?

10.3. Implementation

The implementation is split into three parts: the action calculation, the action ordering and the action execution. The calculation is based on the same graph as used during the *Privacy Analysis* (Figure 8.6). The Assembly Context Actions get calculated independently from the Resource Container Actions, the principle however is the same:

Algorithm 3 Action Calculation algorithm

 1: Dictionary *components*
 2: List of Action *actions*
 3:
 4: **procedure** INIT(List<Components> *runtimeComponents*)
 5:     **for all** *runComponent* ← *runtimeComponents* **do**
 6:         *components*.PUT(*runComponent.AssemblyContextID, runComponent*)
 7:     **end for**
 8: **end procedure**
 9:
10:
11: **procedure** CALCULATEACTIONS(List<Components> *reDeplComponents*)
12:
13:     **for all** *reDeplComp* ← *reDeplComponents* **do**
14:         *runComp* ← GET(*reDeplComp.AssemblyContextID*)
15:         **if** *runComp* == Null **then**
16:             *actions*.ADD(new *AllocateAction*(…))
17:         **else**
18:             **if** *runComp.ComponentID* != *reDeplComp.ComponentID* **then**
19:                 *actions*.ADD(new *ChangeRepoAction*(…))
20:             **end if**
21:             **if** *runComp.ResContainerID* != *reDeplComp.ResContainerID* **then**
22:                 *actions*.ADD(new *MigrateAction*(…))
23:             **end if**
24:         **end if**
25:         *components*.REMOVE(*reDeplComp.AssemblyContextID*)
26:     **end for**
27:
28:     **for all** *runComp* ← *components* **do**
29:         *actions*.ADD(new *DeallocateAction*(…))
30:     **end for**
31: **end procedure**

Initially the algorithm adds all assembly components to a dictionary. In the main procedure, the algorithm iterates over all redeployment assembly components and tries to find a matching one in the runtime model. If no match as found, it is a new assembly component and needs to be allocated. If an equivalent was found, different comparison are made, to check whether adjustments have to made. Keep in mind, that the migration

of a assembly component and the exchange of encapsulated repository component do not exclude each other. At the end of every iteration, the found runtime components get removed from the dictionary. At the end of the algorithm all remaining runtime assembly components are no longer required and can be deallocated. We need to point out, that the comparing operators are simplified for the purpose of a pseudo-code.

The calculation of the Resource Container Actions is implemented similar. Initially all servers get added to a dictionary, all redeployment servers get compared against those and the actions calculated accordingly.

The whole calculation is build around the stability of IDs on Palladio model elements. If the redeployment model creates a completely new system model, while changing only minor details, the calculation will deallocate the old system and allocate a totally new one. PerOpteryx modifies the system model, keeping the assembly context IDs - as intended - stable and therefore produces only minimal actions.

Check back with Replicate Action
Describe Ordering
Describe Execution

11. Evaluation

This chapter is structured as follows: Initially the concept is elaborated (Section 11.1), followed by evaluation scenarios (Section 11.2) and the evaluation of the single tasks: monitoring (Section 11.3), privacy analysis (Section 11.3), model generation (Section 11.4), adaptation planning (Section 11.6) and finally adaptation execution (Section 11.7).

11.1. Evaluation Design

iObserve Privacy is a complex approach with many depending tasks. Evaluating the program as a whole is next to impossible due to the multiplexing dependencies. The evaluation factors would not be manageable and inconclusive results would make the evaluation itself pointless. So we decided to evaluate every task independently. The order and structure got inspired by the iObserve pipeline.

The task evaluation is generally split into an *Accuracy* evaluation and a *Scalability* evaluation. The accuracy evaluation aims for the correct functionality, testing whether the actual results are conform to expected results. For the evaluation we are creating a set of *Evaluation Scenarios*, which reflect real world situations by defining a staring point and an expected endpoint. If the systems result differs from the endpoint, the reasons must be found and analysed.

The scalability evaluation aims for the systems runtime characteristic, based on an increasing work load. The actual accuracy result of the task doesn't interest during this analysis. The primary measurement is the task response time, dependent on the assembly context count and resource container count. Both axis are logarithmic scaled, so the response behaviour is clearly visible. The individual models are randomly generated, based on a repository model input.

11.2. Evaluation Scenarios

The scenarios are structured in *PRE*, *EVENT*, *REACTION* and *POST*. Pre describes the distributed software system before the event takes place. The event is a trigger for a certain process or task chain, usually referenced as reaction. Post defines the state of the software system after the reaction.

The scenarios describe the behaviour of iObserve Privacy through all tasks, while each task gets evaluated individually. Nevertheless, details of an scenario may need clarification during the evaluation of this task.

The scenarios are derived from [6]. This paper describes potential runtime changes to a distributed software system. However, not all mentioned scenarios are of interest in

the privacy analysis context. Scenarios 1 and 2 represent the observed system runtime changes. However, major iOberve privacy failing scenarios are not covered. This is due to the iObserve Privacy design specific nature of error states. Scenario 3 and 4 cover these error.

11.2.1. Scenario 1: Default

This scenario describes the "default" setting. A migration gets monitored, analysed, an alternative deployment calculated and finally migrated. The process works completely automated. This means no operator interaction is required and no task error is generated. As trigger for privacy analysis the server geo-location migration is used.

- **PRE**: All components of the software system are deployed on Amazons EC2 service on the *EU Frankfurt* location. The system is privacy compliant.

- **EVENT**: Amazons EU Frankfurt data centre has a critical failure. As a result Amazon starts migrating local virtual machines towards the US Ohio and EU Ireland locations.

- **REACTION**: iObserve Privacy monitors the migration and starts a privacy analysis. If the analysis doesn't show a privacy violation no further action is taken. If the analysis shows a privacy violation, an alternative, privacy compliant deployment gets computed, a system adaptation plan gets calculated and finally executed.

- **POST**: The software system is in a privacy compliant state.

11.2.2. Scenario 2: System extension

This scenario describes the deployment runtime change. The deployment of a new software component triggers a privacy analysis, which detects "joining data streams" (see Section 8.1). This triggers the generation of an alternative deployment and the system adaptation. The pipeline works like in Scenario 1 without any operator interaction.

- **PRE**: All personal categorized components of the software system are deployed on Amazons EC2 service on the *EU Frankfurt* location. All other components are hosted by an Ukrainian provider. The system is privacy compliant.

- **EVENT**: The system operator adds another as depersonalised categorized component to the Ukrainian host.

- **REACTION**: iObserve Privacy monitors the migration and starts a privacy analysis. The privacy analysis shows a privacy violation due to joining data streams. An alternative, privacy compliant deployment gets computed, a system adaptation plan gets calculated and finally executed.

- **POST**: The software system is in a privacy compliant state.

11.2.3. Scenario 3: Failing Adaptation

This scenario describes the operator-in-the-loop use case, when an adaptation sequence can't be executed automatically. The migration of a software component results in a privacy violating state. After the generation of a privacy compliant alternative and the calculation of the adaptation sequence, the operator is required. One or more adaptation actions can't be executed automatically. This is usually due to missing source control, like the *Change Repository Component Action* or the *Allocation Action*.

- **PRE**: All components of the software system are hosted multiple server instances by cloud reseller.
- **EVENT**: The reseller starts migrating his server to another cloud providers.
- **REACTION**: iObserve Privacy monitors the migration and starts a privacy analysis. The privacy analysis shows a privacy violation. An alternative, privacy compliant deployment gets computed. The adaptation calculation shows certain steps can't be executed automatically. After ordering the adaptation steps the operator gets informed for manual execution.
- **POST**: iObserve Privacy shows the operator the adaptation sequence with emphasis on the manual tasks.

11.2.4. Scenario 4: Missing Alternative

This scenario describes the use case, where no privacy compliant, alternative deployment could be calculated. The migration of a software components results in a privacy violating state. The calculation of privacy compliant alternatives deployment fails. The operator needs to be informed about the current situation.

- **PRE**: All components of the software system are hosted multiple server instances by cloud reseller.
- **EVENT**: The reseller starts migrating his server to another cloud providers.
- **REACTION**: iObserve Privacy monitors the migration and starts a privacy analysis. The privacy analysis shows a privacy violation. An alternative, privacy compliant deployment can't be computed.
- **POST**: …

> Specify end state!

11.2.5. Futile Scenario

There are a couple of scenarios which don't apply to iObserve Privacy, due to various reasons [6]. We will elaborate those scenarios shortly.

Performance or workload characteristics are not tackled, since performance and privacy analysis combined wouldn't be manageable in the scope of this thesis.

The un-deployment or de-replication are two scenarios which reduce the complexity of the privacy analysis. A privacy violation can't be triggered by eliminating a component and/or a server from the system.

The replication of a server, with all its components, will trigger a deployment event. This means, this scenario is already covered by *Scenario 2.*

11.3. Monitoring & Transformation

…

11.4. Privacy Analysis

…

11.5. Model Generation

…

11.6. Adaptation Planning

…

11.7. Adaptation Execution

…

12. Conclusion

12.1. 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] Jochen Dinger and Hannes Hartenstein. *Netzwerk- und IT-Sicherheitsmanagement: Eine Einführung.* Karlsruhe: Univ.-Verl. Karlsruhe, 2008. ISBN: 3866442092.

[5] William Enck et al. "TaintDroid". In: *Communications of the ACM* 57.3 (2014), pp. 99–106. ISSN: 00010782. DOI: 10.1145/2494522.

[6] Robert Heinrich. "Architectural Run-time Models for Performance and Privacy Analysis in Dynamic Cloud Applications?" In: *ACM SIGMETRICS Performance Evaluation Review* 43.4 (2016), pp. 13–22. ISSN: 01635999. DOI: 10.1145/2897356.2897359.

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

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

[9] 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).

[10] Jaeyeon Jung et al. "Privacy oracle". In: *Proceedings of the 15th ACM Conference on Computer and Communications Security.* Ed. by Peng Ning, Paul Syverson, and Somesh Jha. New York, NY: ACM, 2008, p. 279. ISBN: 9781595938107. DOI: 10.1145/1455770.1455806.

[11] 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.

[12] Office for Civil Rights. *Summary of the HIPAA Security Rule*. Washington, D.C., 2013-07-26. URL: `https://www.hhs.gov/hipaa/for-professionals/security/laws-regulations/index.html` (visited on 05/29/2017).

[13] Juan F. Ph.D. Sequeda. *Introduction to: Open World Assumption vs Closed World Assumption - DATAVERSITY*. Ed. by Datavercity. 2012. URL: `http://www.dataversity.net/introduction-to-open-world-assumption-vs-closed-world-assumption/` (visited on 06/13/2017).

[14] 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`.

[15] 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`.

[16] Gregor Snelting et al. "Checking probabilistic noninterference using JOANA". In: *it - Information Technology* 56.6 (2014). ISSN: 1611-2776. DOI: `10.1515/itit-2014-1051`.

[17] 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).

[18] G. Edward Suh et al. "Secure program execution via dynamic information flow tracking". In: *ACM SIGOPS Operating Systems Review* 38.5 (2004), p. 85. ISSN: 01635980. DOI: `10.1145/1037949.1024404`.

[19] teetime. *TeeTime Framework*. 2017-05-16. URL: `http://teetime-framework.github.io/` (visited on 05/26/2017).

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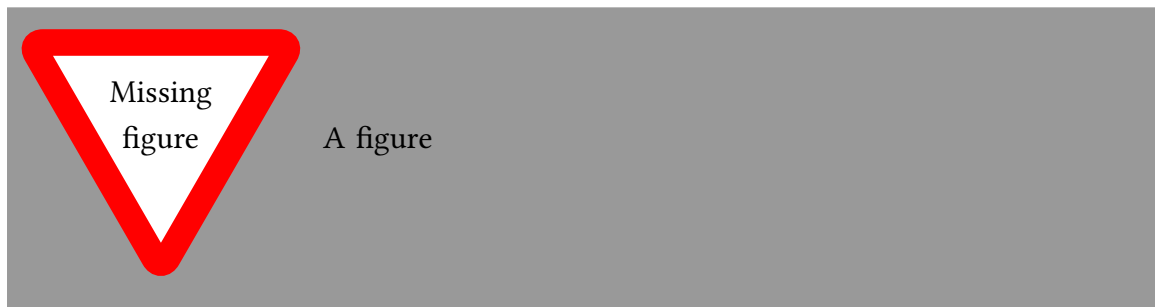
A. Appendix

A.1. First Appendix Section

Figure A.1.: A figure

…