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

With the steadily increasing number of (distributed) cloud systems and more strict data protection regulations, an increasing interest in privacy law compliant cloud applications arises. The major factors to privacy compliance is the distribution of personal data among the geo-location. We developed, implemented and tested a privacy analyser for this factor. Further, ee extend iObserve after the MAPE loop for automated privacy violation detection, alternative deployment generation and an according cloud system adaptation. This way we can provide continuous privacy compliance on a software architecture level, without code analysis. However, we require the closed world assumption for privacy compliance. PerOpteryx is used for the generation of privacy compliant alternative system deployment. Based on this alternative we compute a series of adaptation steps to re-establish privacy compliance. On error occurrence, we make use of the operator-in-the-loop approach of iObserve to help with system evolution. iObserve and PerOpteryx use the Palladio Component model as Architecture Description Language. In this thesis, we are describing our concepts, point out implementation details and evaluate the iObserve extension. The accuracy evaluation shows our system works as intended and the scalability evaluation reveals the good performance characteristics.

Zusammenfassung

Deutsche Zusammenfassung

Contents

List of Figures

List of Tables

1. Introduction

During the introduction we will *motivate* (Section 1.1) for the topic at hand and introduce the *problems* (Section 1.2) arising from it. Further, we will introduce the *goal and research questions* (Section 1.3) handled in this thesis. Finally, we will give a short *outline* (Section 1.4) about the remainder chapters.

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 simplification of its usage and the abundance of own data centres. Netflix, for example, closed all its owned data centre in 2015 and moved completely to Amazons AWS[6]. This trend results in the expected revenue of about 200 Billion $ in 2016[24]. 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.[3]

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[4] or the US HIPAA act[18]. To tell the whole truth, the complexity, the hidden usage of services 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 enforcement point of attention tends to change quickly, like the Max Schrems' "Facebook Process" showed [13]. Further, in 2018 the "reform of EU data protection rules" will come into force, which states severe punishments for privacy violations[4]. As a result, 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 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 though 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 [5]:

- **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 and execution

1.3. Goals and Research Questions

This thesis' goal is to contribute and outline a piece of software, that ensures continues privacy compliance, modelled after the MAPE-K feedback loop. Connected to this goal, a number of important research questions arise:

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

2. Foundations

In this chapter we will introduce applications and principles on which this thesis is based on. This introduction aims for a general understanding. We would like to point out that some topics may be discussed in more detail in the corresponding section.

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 [5]:

- **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 [5].

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].

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 instrumenting the system with probes during pre-compilation. Each probe has designated purpose and gathers data accordingly, for example hardware utilization, stack trace or host geo-location. Kieker uses event-based probes, as well as periodic-based (heart-beat) probes [20].

2.4. iObserve

iObserve is a system optimization tool after the MAPE feedback loop. The two primary goals are an automated system adaptation and an *operator-in-the-loop* system evolution (see Figure 2.1).

For the initial MAPE step, monitoring, iObserve uses Kieker. The gathered information are transformed onto a Palladio Component Model. Due to the live characteristic this PCM model is called a runtime model. Further, iObserve is designed to support distributed cloud systems. Key features of the transformation are the processing stack trace information for usage model updates, so more precise performance simulations are possible. Further the processing of deployment and un-deployment events, as well as hardware utilization measurements.

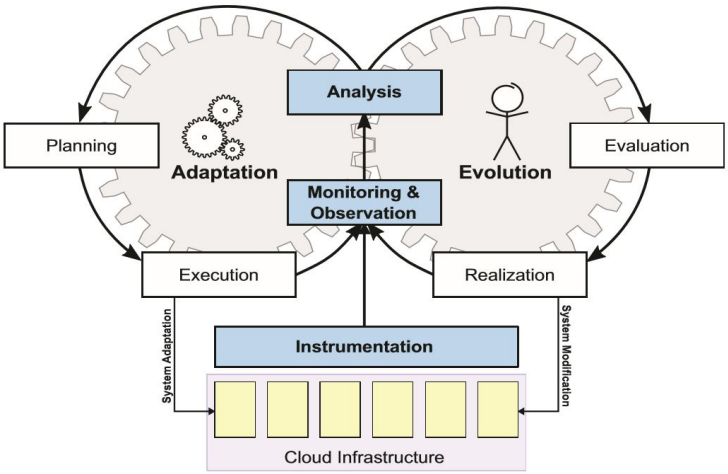

Figure 2.1.: iObserve cloud application life cycle [9]

Currently, iObserve goes as far as updating the model. iObserve uses the TeeTime framework [27], a pipeline-filter-framework with signal based stage invocation. More details on iObserve can be found in [9][10].

2.5. PerOpertyx

"PerOpteryx is an optimization framework to improve component-based software architecture"[16]. The 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. This approach is usually described as *Design Space Exploration* (DSE). PerOpteryx uses multiple dimensions for its DSE, like alternating component multiplicities, runtime parameters or changing component allocations. The pareto optimal models are calculated through multiple iterations of a series of tasks. Initially a variance of candidates is created through an evolutionary algorithm and random generation. In the next step, the candidates are analysed for the desired quality marks along the different dimensions. The iteration concludes with the elimination of poorly performing candidates. PerOpteryx is designed to optimize Palladio Component Models, is based on the *Rich Client Platfrom* and uses the *Opt4J Framework* [17].

3. Privacy Concept

Many say: Data is the new oil and the most valuable resource there is. This shows how important the control of our personal data is. 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 one can't act for the individual, we can provide tools and rules for institutions to help with legal compliance.

3.1. General Concept

The EU General Data Protective Regulations clearly states, that data of EU citizens have to be saved and processed within EU countries [4]. Only a view countries with equal data protective laws are excepted from this constraint. As a consequence, one needs a simple data-flow analysis (Section 11.3) to know the data distribution in our software system. To put it straight, one needs to know, what kind of data are available on which server. This task got especially important, since distributed cloud systems are reality and data saved on "on premise" servers are becoming increasingly rare.

As mentioned in Section 11.3, the automated data-flow analysis on architecture level is still in its early 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 [22]:

- **Type 0: Personal Information**: Data relates directly or indirectly to personal information. This is independent from encryption or pseudonymization. (e.g. call detail record)
- **Type 1: Personally Identifiable Information**: Data does not contain personal information. However, by combining, fusing or analysing data sets, the personal data could be reconstructed for complete or partial personal information. (e.g. browser history without user)
- **Type 2: Anonymous Data**: Data does not contain any personal information. Even by extensive data analysis no direct or indirect personal information can be extracted. (e.g. shop inventory data)

These three categories are used, since many data does not contain any direct or indirect link onto private data, however still contain indicators onto private data. This means,

they neither qualify for the type 0 category (Personal), nor for the type 2 category (Anonymous). For example an online shop wants to analyse which products usually get ordered together. The orders got anonymized by removing the customer and shipping address. Nevertheless, the time-stamp is required to get a timed evaluation factor. These 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 [21][22].

Summarizing, a manual, categorized annotation approach to identify the system components privacy level is used. Based on this privacy level the check, whether a systems deployment is privacy compliant, can be performed.

3.2. Deployment Constraints

How can one 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.

Rule 1 & 2 does not 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.

The ideas is, when a depersonalised component (d) has a single edge to a personal component (p), and many edges to other depersonalised components (D), and the components (D) do not have any connection a personal categorized component, then they share the data passed via the connection between d and p, which can not be personal. Otherwise d would be categorized as personal. 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.

We are aware, that these rules are over-approximations towards legal compliance. However, we are using a software architecture model as the only source of information. Therefore, detailed information about actual privacy symbiotic data streams are not available on this high level ob abstraction. Nevertheless, we argue, that these rules already help to identify illegal deployments and establish privacy compliant (re-)deployments.

Figure 3.1, Figure 3.2 and Figure 3.3 illustrate the different base scenarios applying to Rule 4. In the remainder of this section, we will elaborate their privacy compliance state by applying Section 3.2, rule 4.

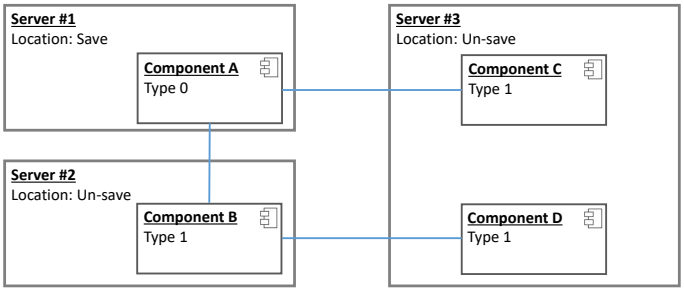

Figure 3.1.: Privacy violating deployment

The deployment shown in Figure 3.1 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 combination, the data could be privacy relevant. So, this deployment is considered illegal.

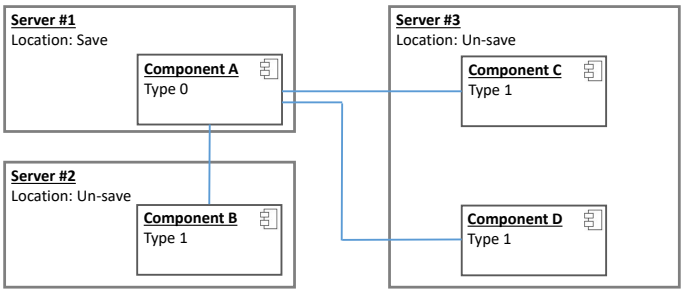

Figure 3.2.: Privacy violating deployment

Figure 3.2 also shows an illegal considered deployment. Component C and D share the same data source, which is marked as type 0. As previously shown, the combination of data on Component C and D could lead to privacy relevant informations on *Server #3*.

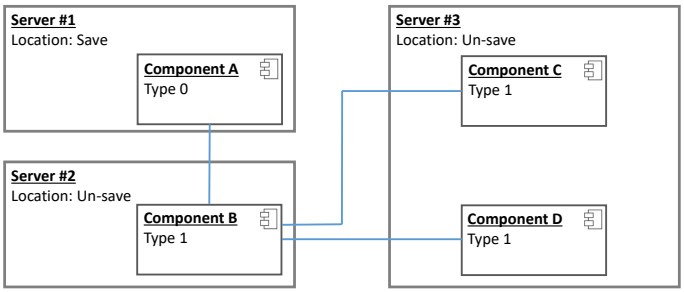

Figure 3.3.: Privacy compliant deployment

(Figure 3.3) shows a privacy compliant deployment. Due to, Component C and D sharing the same data source (Component B), which already only contains type 1 data. As

a result, Component C and D can only contain data, already obtained by Component B. This means, even through data combination and extensive data analysis, no privacy concerning information can be extracted.

3.3. 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, that a component shouldn't be categorized by hand.

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

As a result, the decision was made 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 with the *Closed World Assumption* (CWA) [19]. 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 that exist. Naturally, this assumption is wrong, since every component may be 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. Considering the limited information we have available, the system architecture model, the statement we are providing can be considered outstanding, even while using the CWA.

3.4. 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. As a result, 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 3.3, 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.

4. Overview

In this chapter we will give an overview on the approach developed during this thesis and the according research. The system is massively extending iObserve, while keeping its original purpose. For the fundamentals on iObserve see Section 2.4. All extensions are made for accomplishing the goals, defined in Section 1.3. The extended iObserve is referred to 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).

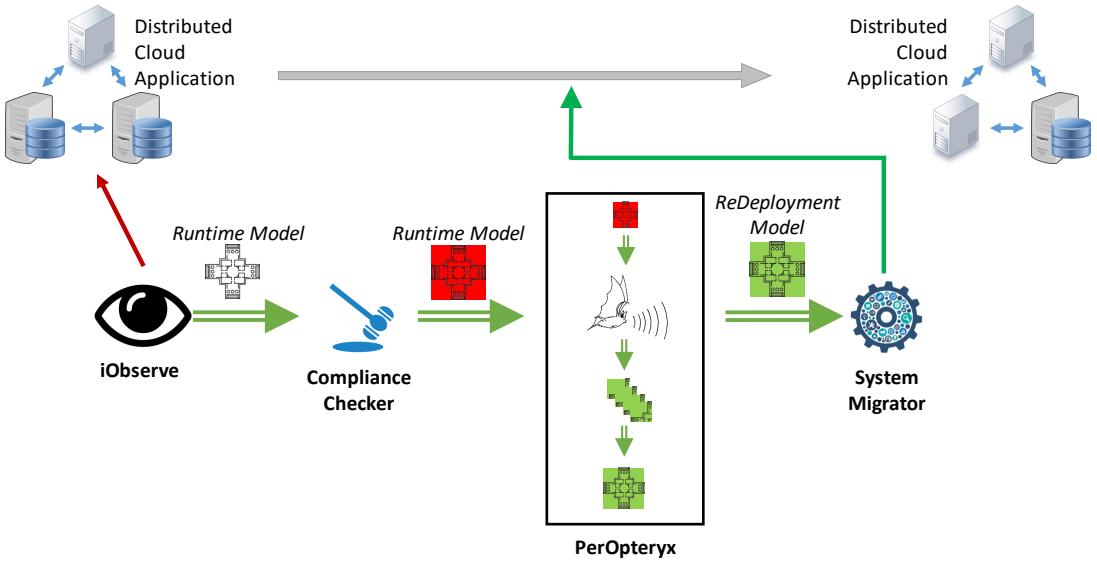

Figure 4.1.: iObserve Privacy pipeline

The initial step, monitoring, is provided by the original iObserve. However, it doesn't support any information on the components' geo-location. This extension is made directly in the original iObserve and Kieker. Upon detected geo-location or deployment changes, the runtime model gets updated and the next filter stage is invoked. We have determined the required data for a privacy analysis and provide a suited transformation of these information onto the PCM Privacy runtime 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 3. 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 and

optimization framework. However, PerOpteryx doesn't support privacy or deployment constraints and therefore needs an extension. Furthermore, 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 architectural re-deployment model to the architectural runtime model. Based on comparison, a migration adaptation sequence is calculated. The adaptation sequence is technology independent and consists of individual adaptation actions. Finally, the adaptation sequence is executed. The execution is based on scripts, that represent a technology dependent implementation of the individual adaptation actions. After the 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?

5. PCM Extension

As mentioned in Section 2.2, the Palladio Component Model was designed for early architectural performance analysis. On this basis, the PCM was modified many times to fulfil many adjacent tasks. Instead of 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.

5.1. General

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 to be as minimal invasive as possible and to keep the adaptation effort for existing Palladio models to a minimum.

Our extension is based on deriving the PCM meta-model entities. This way the new models stay compatible to other PCM applications like the Performance Simulators or PerOpteryx. Further, other extension possibilities like the *steriotype* extension needs to be updated with every PCM meta-model update. This is not necessary, when the model is derived. Only changes in reference structure require the derived model to apply minor alterations. For details on referencing and modularizing in meta-models see [25].

The concept was described in Section 3.4.

5.2. 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 3.4, we need to save a servers geo-location. The Resource Container was extended and the attribute *Geolocation* added. The derived 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 [30]). By using an integer encoded international standard we stay wildly compatible with other applications. Further, potential error sources like spelling differences, shifting borders and name changes are avoided.

The *Assembly Connector Privacy* is derived from the Assembly Connector and saves the data privacy categorization. The attribute is designed as an EEnum Ecore type, with the values Personal, Depersonalized, Anonymized. This way, extending the potential categorization values requires minimal effort and is less error then a dynamic categorization via

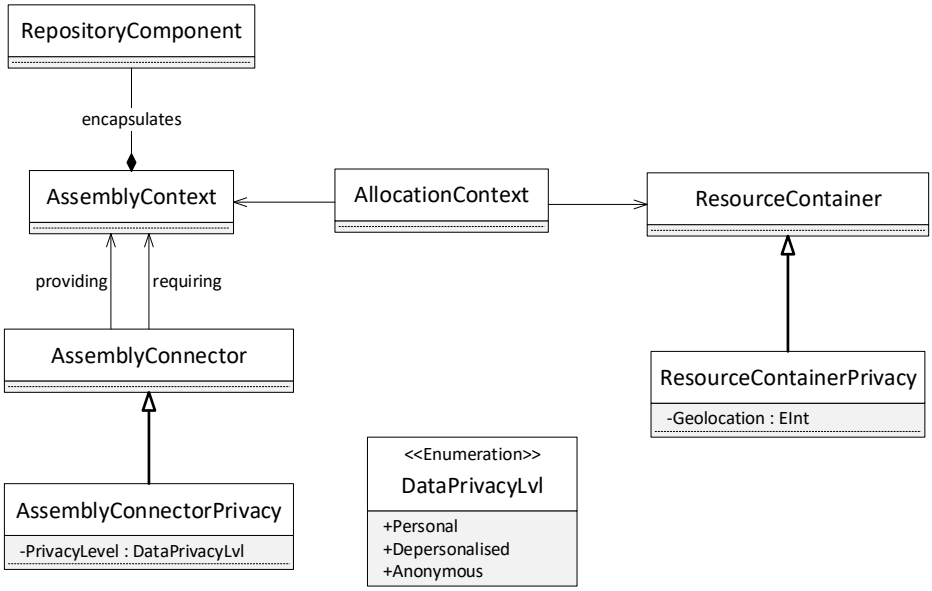

Figure 5.1.: PCM Privacy meta-model

a float or an integer value. The value *Personal* is set as default, to provide a over-estimation towards the legally compliant categorization, when the categorization if a connector was not performed. Figure 5.1 shows a simplified PCM meta-model with the three added privacy elements.

6. 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 [10].

6.1. Kieker

Kieker was also briefly introduced in Section 2.3. Kieker had already specified the geo-location record for transporting the geo-location information form the observed system to Kieker. However, a probe was still missing. As a result, we created a heart-beat/periodic probe (*ServerGeoLocationSampler*). This probe uses a *ICountryInvestigator* to determine the actual geo-location and creates a *ServerGeoLocation* record (see Figure 6.1).

As an alternative, we could have created an event-based geo-location probe. Possible event could have been the component deployment or un-deployment or the server acquiring or release. This however, would mean, there would not be a geo-location update between these events and potential privacy violation could stay undetected indefinitely or for a long period. Even though a heart-beat probe often takes longer to send an initial record, eventual changes will definitely be detected due to the regular updates.

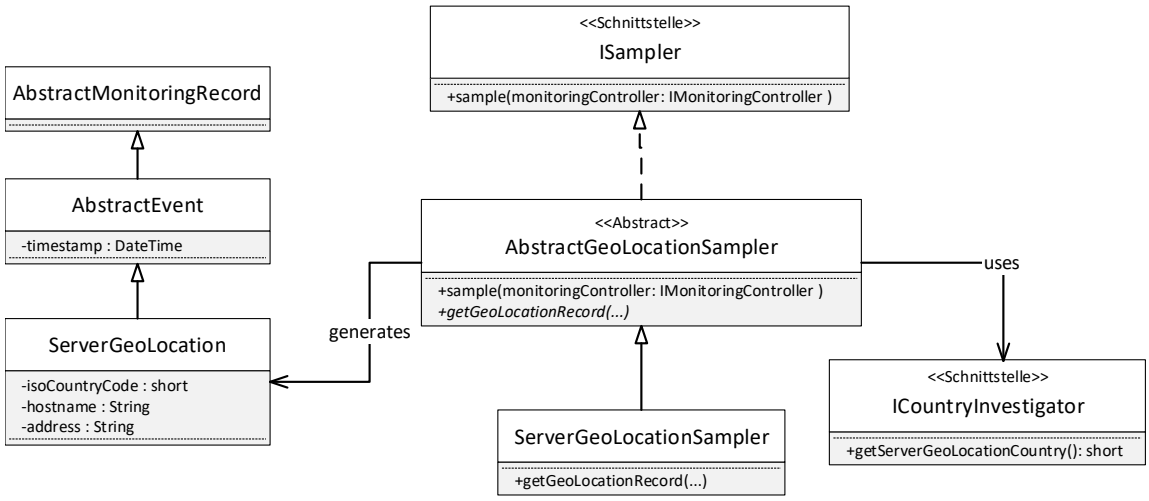

Figure 6.1.: Server Geo-Location Record and Sampler

Kieker gather the information from the observed system probes and redirects them to iObserve. More information on Kieker can be found at [20].

6.2. iObserve Privacy

iObserve uses the TeeTime framework. Its allows easy pipeline building by connecting matching input and output ports during runtime [27]. 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 6.1) 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.

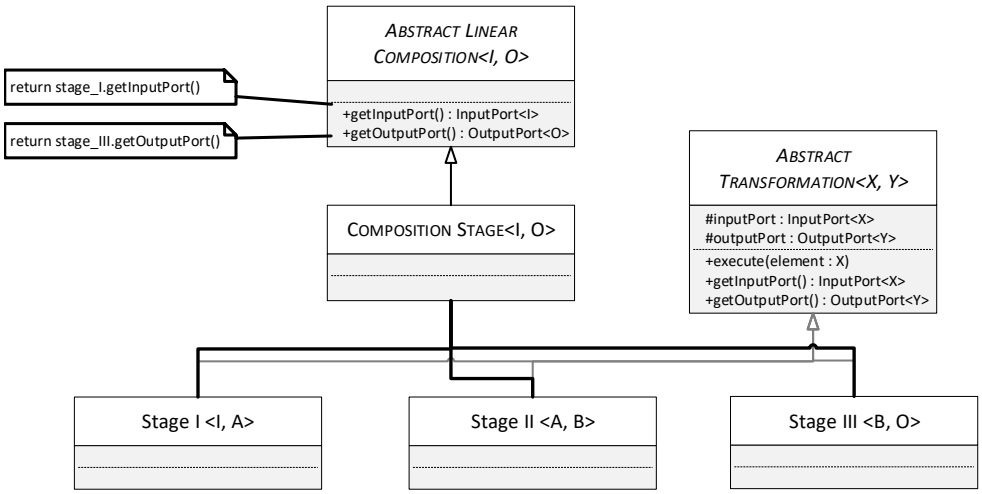

Figure 6.2.: iObserve Privacy Filter

iObserve has a well defined structure due to the TeeTime framework. We decided to keep this structure and extend it. Figure 4.1 shows the conceptual filter pipeline structure for our planned extension. One conceptual stage usually consists of several sub tasks. The system adaptation, for example, consists of an adaptation calculation, an adaptation calculation and an execution. To embrace re-usability and lose coupling, we decided to keep on using the TeeTime framework structure and compose one conceptual stage out of several sub-stages. Figure 6.2 shows the general structure of a conceptual stage. The *Composistion Stage* functions as a wrapper for the conceptual filter, while the *Stages* I to III represent the sub-tasks.

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*, so potential misunderstandings are avoided.

7. 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 deployment related privacy violations. The privacy concept, described in chapter 3, states that privacy analysis consist of two major tasks. First, the correct privacy categorization for each software components needs to be determined. Second, the deployment evaluation, based on the deployment rules, defined in Section 3.2, needs to be performed.

The remaining chapter is divided into a theoretical part (Section 7.1), a component categorization part (Section 7.2), a deployment evaluation part (Section 7.3) and the implementation part (Section 7.4).

7.1. Analysis Theory

The exclusive source of information for the privacy analysis is the systems PCM Privacy runtime model. For an efficient analysis one first need to identify the minimal required information and substitute them, depending on the available sources.

7.1.1. Required information

In the context of Privacy Analysis there is a minimum of two pieces of 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 7.1.: Minimal information for privacy analysis

Usually *M1* and *M2* is 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 a data privacy categorization of the communicated information between component interfaces. This enables a component classification, based on the most critical communication with another component. As a result, a components privacy level can be determined 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 these information. However, it is spread over multiple models. As a result we require only four pieces of information (Table 7.2), compared to R-PRISs six pieces of information (Table 11.1).

#	**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 7.2.: iObserves information for runtime privacy checks

7.1.2. Data-flow direction

Usually component interfaces are categorized into providing and requiring interfaces. Interface connections are made between a pair of required and provided interfaces of the same type. This suggests 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 component (usually) only has provided interfaces. These interfaces allows the user to store and retrieve data from the database and therefore contains getter and setter methods. This means, there is no data-flow direction for the whole interfaces, since the data needs to "flow" into both directions. Note, that we are explicitly speaking of an interface as a whole, since individual methods can have a data-flow direction.

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

7.1.3. Joining data streams

In Section 3.2 we elaborated on the danger of two *Personally Identifiable Information* (Type 1) data streams, from two sources, joining on a single server. In such a case, the combination of these data streams could lead to personal, privacy relevant data. (Compare Section 3.2, Rule 4). This concept applies on the described deployment level and 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* (short: JDS). In the following section, a couple of categorization and deployment examples will help clarifying this scenario.

7.2. Component categorization

The component categorization requires two tasks. The initial categorizing of every component and the analysis for with *join data streams*.

The initial data privacy level of a component is equal the components most critical communication level (see Subsection 7.1.2). This task is performed during the model graph construction. The search for joining data streams is more complex and requires a formalization, as well as an extended explanation. We will continue with the formal description, followed by an textual explanation and close with some examples.

Definitions

- The Graph $G := (N, E)$
- The Nodes N consist of personal Nodes $p \in N_p$, depersonalised Nodes $d \in N_d$ and anonymized Nodes $a \in N_a \wedge N_p \cup N_d \cup N_a = N \wedge N_p \cap N_d \cap N_a = \{\emptyset\}$
- The edges E consist of personal Edges $e_p \in E_p$, depersonalised Edges $e_d \in E_d$ and anonymized Edges $e_a \in E_a \wedge E_p \cup E_d \cup E_a = E \wedge E_p \cap E_d \cap E_a = \{\emptyset\}$
- Path $P = \langle (p_1, p_2, \ldots, p_n), (e_1, e_2, \ldots, e_{n-1}) \rangle$ with $p \in N$ and $e \in E$

Formalization

$$\text{Every component is correctly categorized}$$
$$\Leftrightarrow$$
$$\nexists \text{ Path P } \text{ with let } i, j \in [0, n-1] \wedge i \neq j : e_i \neq e_j \wedge e_i \in E_d$$
$$\wedge (p_2, p_3, \ldots, p_{n-1}) \in N_d \wedge \{p_1, p_n\} \in N_p \wedge n \geq 3$$

The formalization states, that every component in the graph G is correctly categorized if no path from one personal component to a personal component exist. However, the path must traverse at least three components, while every edge is only used once and only depersonalised edges are used. Further, all internal nodes must have a depersonalised categorization.

When such a case is found, the depersonalised components of the path must be categorized as *Personal*. As a result, the case is eliminated and categorization is corrected according to the *joining data stream*. The following examples will illustrate the categorization and point out certain special cases.

Figure 7.1 shows the components data privacy level after the initial categorization phase. Figure 7.2 shows the result of categorization analysis. The comparison of these two states show, that component D and E get "upcasted" and gain a *Personal* - more critical - data privacy categorization. This is due to the fact, that component D and E have a connection onto two personal data sources (Component A and B). Applying the formalization, a path from component A to B via component E and D, using only depersonalised edges, can be found. This means, as mentioned above, a joining data stream exists and component D and E needs to be categorized as *Personal*.

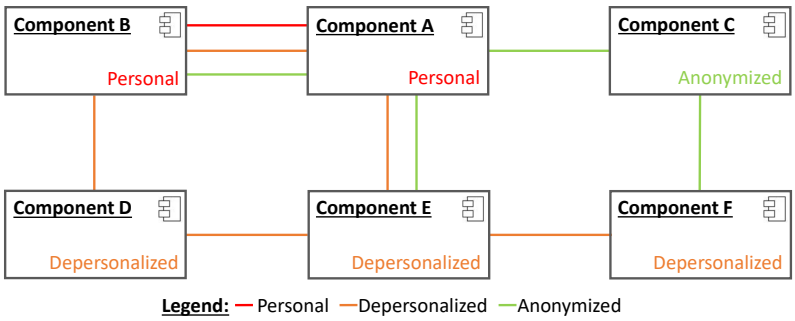

Figure 7.1.: Initial component categorization

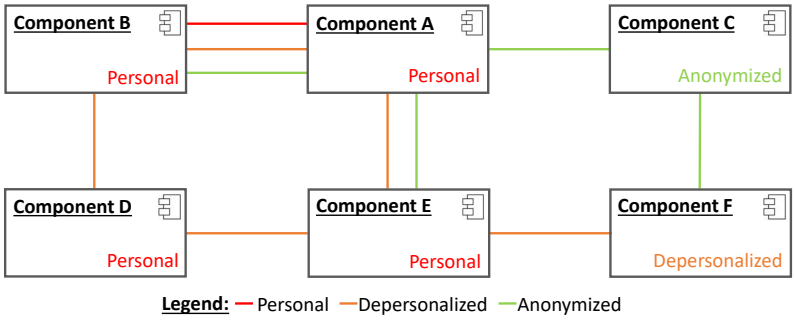

Figure 7.2.: Post categorization analysis - basic example

Two special cases are shown in Figure 7.3. So is component D categorized as *Personal* 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. The formalization states, that a personal component needs to be reached, while using every edge only once. This conditions are fulfilled.

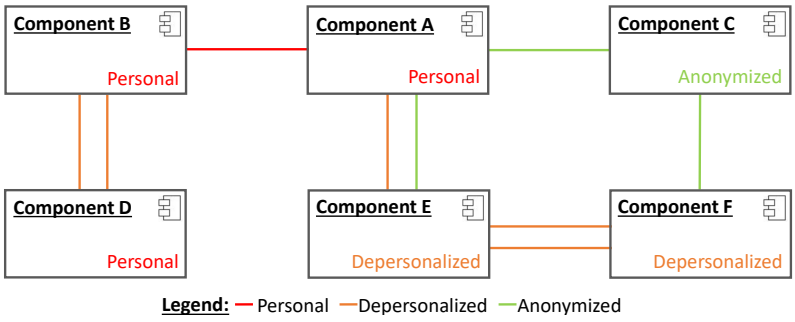

Figure 7.3.: Post categorization analysis - advanced example

Component F doesn't get an *Personal* categorization 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 they don't contain any privacy related information. Applying the formalization, no path from A to a personal component can be found, using only depersonalised edges and each edge only once. So, the categorization is correct.

7.3. Deployment analysis

The deployment analysis' goal is to find out whether the current deployment is privacy compliant. A deployment is considered privacy compliant if no deployment violation is found. The rules for a privacy compliant deployment were described in Section 3.2. In the following we will formalize the deployment analysis, describe the formalization textually and finally give an example.

Definitions

- The Graph $G := (N, E, S)$
- The Nodes N consist of personal Nodes $p \in N_p$, depersonalised Nodes $d \in N_d$ and anonymized Nodes $a \in N_a \wedge N_p \cup N_d \cup N_a = N \wedge N_p \cap N_d \cap N_a = \{\emptyset\}$
- The edges E consist of personal Edges $e_p \in E_p$, depersonalised Edges $e_d \in E_d$ and anonymized Edges $e_a \in E_a \wedge E_p \cup E_d \cup E_a = E \wedge E_p \cap E_d \cap E_a = \{\emptyset\}$
- The servers S consist of save Servers $s_s \in S_s$ and un-save Servers $s_u \in S_u \wedge S_s \cup S_u = S \wedge S_u \cap S_s \cap N_a = \{\emptyset\}$
- Let N_{si} are the nodes deployed on server s_i
- Path $P = \langle (p_1, p_2, \ldots, p_n), (e_1, e_2, \ldots, e_{n-1}) \rangle$ with $p \in N$ and $e \in E$

Formalization

$$\text{The deployment is illegal}$$
$$\Leftrightarrow$$
$$\exists n \in N_{si} : n \in N_p \wedge n \text{ deployed on } s_i \wedge s_i \in S_u$$
$$\vee$$
$$\exists s_x \in S_u \nexists \text{ Path with } N_{sx} \subset P \wedge p_i \in N_d \wedge e_j \in E_d$$
$$\vee$$
$$\exists \text{ Paths } \{P_1, P_2, \ldots, P_n\} \text{ from } N_p \text{ to } N_{si} \text{ with } i \in [0, n], j, k \in [0, |P_i|] :$$
$$e_j \in E_d \wedge e_{pj} \cap e_{pk} = \{\emptyset\} \wedge n \geq 2$$

The formalization states three independent conditions for an illegal deployment. First, the deployment of an as personal categorized component on a server located in an un-save geo-location. Second, the depersonalised components on a un-save server are not connected. The potential connection must consist of depersonalised edges but can be transitive via depersonalised components and edges. The third condition states, that if there is more then one path, from the set of personal components to the depersonalised

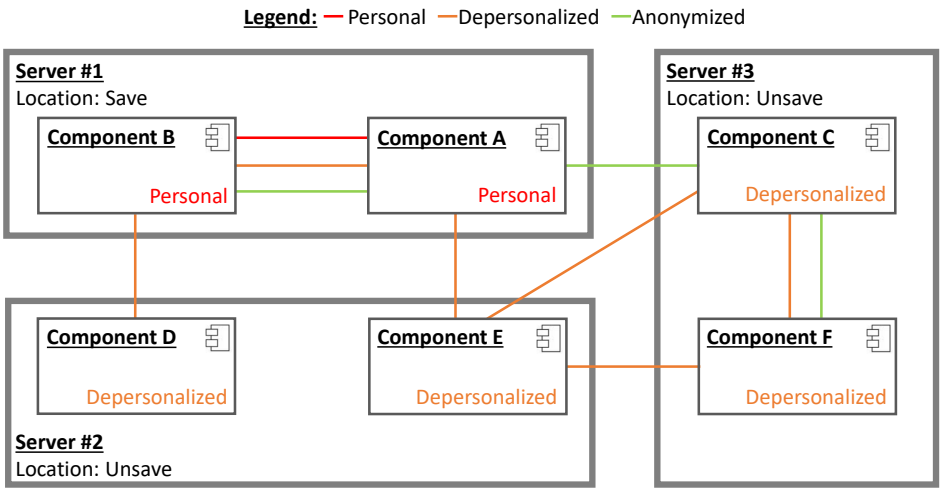

Figure 7.4.: Deployment analysis example

components hosted by an un-save server, then the deployment is illegal. However, must consist of individual, non overlapping and depersonalised edges.

Figure 7.4 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. Applying the formalization, the first condition is broken, since server #3 does not host a personal component, the second condition is also false, since the components C and D are transitive and direct connected via depersonalised edges. The third condition does also not apply, since only one individual path via depersonalised edges from the personal components exists.

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. The second and the third condition of the formal specification are true. So, are component E and D not transitively connected via depersonalised nodes. Further, two individual paths from the set of personal components to the components hosted by server #2 exist.

7.4. Privacy Analysis implementation

The PCM meta-model defines multiple models, each providing knowledge about a certain aspect of the target system (see Figure 7.5). 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

7.4.1. 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 7.6). 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:

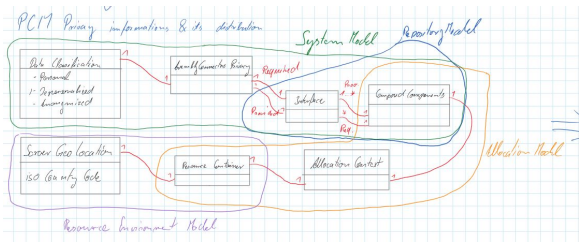

Figure 7.5.: PCM Privacy information spread

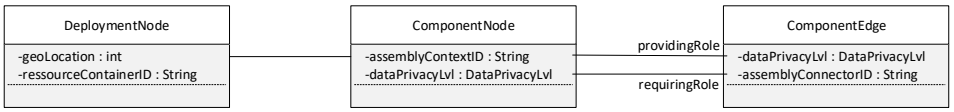

Figure 7.6.: Graphs meta-model for Privacy Analysis

7.4.2. Component categorization implementation

The second phase of the privacy analysis algorithm finalizes the component categorization. As described in Section 7.2, 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.

7.4.3. 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

```
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
```

8. PerOpteryx Extension

PerOpteryx, briefly introduced in Section 2.5, is a model optimization framework. It is designed to calculate performance and cost optimised PCM models. For this purpose PerOpetryx uses an evolutionary algorithm to generate new PCM candidates. Every candidate needs to be evaluated in order to decide if the decisions made during its constructions lead to a good results. Each evaluator can produce multiple results per analysis runs. Each result belongs to a certain QML dimension. A dimension can have *Objectives* and/or *Constraints*, which help the evolutionary algorithm to find the Pareto-optimal candidates. Every evaluator is encapsulated in an *Eclipse Plug-in* [16]. Since we need another evaluator, we need to create a new plug-in.

8.1. Plug-in Design

We want to provide a *Privacy Analysis* evaluator for PerOpteryx, while using our previously developed *Privacy Anylsis* (chapter 7). Since both systems are based on the Palladio Component Model, the privacy analysis itself can be used as described in chapter 7. However, PerOpteryx does not know a "Privacy Dimension", which is required, since every analysis result needs an according dimension. A new dimension is required, since using a pre-existent dimension - like the "cost dimension" - would undermine the evolutionary algorithms optimizations effort. The privacy analysis has a single result, which is a *Constraint*. It states, that no privacy violation is permitted.

As mentioned before, we need to create a new *QML Dimension*, the *Privacy Dimension*, which is referenced in a *QML Contract Type*. The contract type references all the evaluation dimensions. The *QML Declaration* references the contract type and specifies the actual objectives and constraints for the dimensions. Further, it specifies a *QML Profile*, which *Usage Model* is used for the evaluation.

8.2. PerOpteryx Modification

PerOpteryx' prior structure considers every generated candidate to be valid, only with different runtime results. This is incorrect, when the privacy dimension is included. Only privacy compliant models are valid options. As a result, we can abort the evaluation of the current PCM model, if the privacy constraint is broken. However, this means we need to execute the privacy evaluation first, check if the constraint is broken, break the evaluation if the model is invalid and fill all other objectives and constraints with according values.

As mentioned above, every evaluation is encapsulated in an *Eclipse Plug-in*. So, every evaluation is represented as *Proxy Analysis* in PerOpteryx, who has no information on

what evaluation he is actually currently executing. To save evaluation time and increase our search space, we need to execute the Privacy Analysis first, while not breaking the generic evaluation characteristic. As a result, the *IAnalysis* interface for every evaluation was extended with a *Evaluation Complexity* query, returning an Enum representing the analysis runtime duration. PerOpteryx was modified in a way, that analysis returning the value *VERY_SHORT* get executed first. The Privacy Analysis evaluator returns this value.

Further, PerOpteryx was modified, to output the *most cost efficient* candidate once all evaluation iterations are completely executed. The cost criteria was chosen over performance due to PerOpteryx's tendency to spread the allocation over many server to optimize the performance. The cost optimal model tends to group components on servers, reducing the servers required and therefore saving money, while still having a decent performance.

9. 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 divided into the calculation of an adaptation plan (*Adaptation Planning*, Section 9.1) and the execution of this adaptation plan (*Adaptation Execution*, Section 9.2). It closes with a look onto the implementation (Section 9.3).

9.1. 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 (see chapter 8). This file contains all choices made during generation of the redeployment model. These informations are a viable source for the action computation. However, this source create a strong dependency on PerOpteryx. This means, when PerOpteryx would be exchanged for another model optimization tool, the complete adaptation planning needs to be re-thought and developed.

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 architectural runtime model and the architectural redeployment model. 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. The comparison itself is based around identifier and content equality. The *Adaptation Caluclation* compares the model graphs (see Subsection 7.4.1) whether a component was added, removed or modified, as well as a server was acquired or released. Derived from these differences, *Adaptation Actions* are created.

9.1.1. 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.

Add Ref

Further, we designed a set of basic actions which allows us to transform any PCM runtime instance into any PCM re-deployment instance. These are derived from the runtime changes specified in **XXX** and the potential variation *PerOpteryx* calculates during its optimization process. 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. The Allocate action represents a new or first deployment of a system component, the Deallocate action represents the exact opposite, the deleting or un-deployment of a component. And Migration action moves a component from a server o another. 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.

We decided to model these actions into an EMF model and reference the *PCM meta-model*. This allows us to directly reference the affected resource container or assembly contexts without any technology implications. Writing this code by hand would cause more effort on a PCM meta-model update, however, the generated EMF models perform poorly during debugging. Nevertheless, EMF meta-models are easily extendable and modifiable and therefore perfect for the task at hand. These actions were designed and developed in cooperation with [28].

9.1.2. Action Ordering

For each action a set of pre-execution-conditions can be determined. Using this sets, a universal order can be derived. However, we need two assumptions for this order to be valid:

- Each component is affected by an action only once.
- The *Change Repository Component Action* does not affect a component.
- A server never gets acquired a terminated in one sequence.

The assumptions state, that neither assembly contexts, nor resource container are affected by transitive actions. These assumptions are true, since the *Adaptation Calucu-lation* (Section 9.1) is designed to calculate a direct transition into an entities final (re-deployment) state. As a result, the order inside a set of actions from the same type does not matter. Further, the order within all *Assembly Context Actions* does not matter. The pre-execution-condition per action are:

Action	**Pre-Execution-Condition**
Allocate	execute after *Acquire*
Deallocate	execute before *Terminate*
Migrate	execute after *Acquire* & before *Terminate*
Change Repository Component	execute before *Migrate*
Acquire	–
Terminate	–
Replicate	–

Table 9.1.: Pre-Execution-Conditions for adaptation actions

Note, that the change repository component condition is not a "hard" condition, since the encapsulated component could be exchanged even after the migration is performed. However, the appended action data reference the hosting resource container and assembly context *before* the component is migrated. Without this condition, the action execution would have to check, whether the component was migrated.

Based on the pre-execution-conditions of Table 9.1 the following order was calculated:

1	Acquire
2	Change Repository Component
3	Deallocate & Allocate & Migrate
4	Terminate & Replicate

Table 9.2.: Universal action execution order

This order is a universal execution order for the actions calculated by the *Adaptation Calculation*. When the actions are executed in this order no dependency conflicts will occur. Note, execution errors are not considered, since the are independent form planning.

9.2. Adaptation Execution

The adaptation executions task is to execute the adaptation sequence calculated by *adaptation planning*. Since the actions are technology independent, we require a technological dependent script/function that represents the action. A scripts can be considered the implementation of an action. These scripts are given to iObserve Privacy via input parameter.

The technological implications are not considered by this thesis and are therefore not further discussed. However, the separation of technology independent and dependent part of the execution is very futile to the whole iObserve principle.

We recommended to execute the scripts asynchronous. This enables iObserve privacy to track the changes in real-time. This allows for post-adaptation evaluation. This means, iObserve (privacy) can check, whether the *runtime model* is now semantically equivalent to the *re-deployment model*. Further details on the execution of the adaptation actions can be found in [28].

9.3. 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 7.6). The Assembly Context Actions get calculated independently from the Resource Container Actions, the principle however is the same. See Algorithm 3 for the pseudo code.

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.

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
```

10. Evaluation

This chapter is structured as follows: Initially the concept is elaborated (Section 10.1), followed by evaluation scenarios (Section 10.2) and the evaluation of the single tasks: monitoring (Section 10.4), privacy analysis (Section 10.4), model generation (Section 10.5) and adaptation planning (Section 10.7). Finally we are analysing the threats of validity (Section 10.8). Note, all evaluation models, test data and results can be found at [29].

10.1. 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 was 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 starting 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 is of no evaluated 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.

We will use the *Jaccard Coefficient* to evaluate model changes during the accuracy evaluation. Prior to the execution a target model is created, representing the desired post-execution state. The runtime model is compared to the target model, differences are calculated, as well as the jaccard coefficient. The coefficient is defined as the *intersection set* of runtime and target model *divided* through the *union set* of these models:

$$JC(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

If the models are completely equal, the result is *1.0* [1]. We are comparing the system model, the resource environment model and the allocation model. The repository and the usage model model is not modified by iObserve Privacy and therefore do not need any comparison. All models elements are matched by element content. The system and

resource environment model are also compared by element ID. All models modified by iObserve Privacy are order independent, so an order match, like the *Spearman Coefficient*, is not required.

The test device is a *Surface Pro 4* using *Windows 10* as Operating System. The important hardware specifications are a *i7-6650U* processor with *2.2 up to 3.4 GHz, 4 MB cache* and *2 cores with hyper-threading*. The system uses 16 GB of RAM and a 265 GB Solid State Drive. For more details see [31]. The used *Java* version is *1.8.0_121* form *Oracle*.

10.2. 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 out all tasks, while each task gets evaluated individually. Nevertheless, details of a scenario may need clarification during the evaluation of this task.

The following scenarios are derived from the *runtime changes* mentioned in [9]. This runtime changes are possible modification to a distributed software system. However, not all mentioned scenarios are of interest in the privacy analysis context, these will be discussed in Subsection 10.2.5. Scenarios 1 and 2 represent the observed system runtime changes, which trigger the iObserve privacy pipeline. These are designed to show the successful execution of a pipeline run with different triggers. Scenario 3 and 4 cover the *operator-in-the-loop* scenarios. The operator is required, when an error occurs that iObserve privacy can not handle itself. These scenarios are design specific and therefore not derived from the runtime changes described in [9].

10.2.1. Scenario 1: Default

This scenario describes the "default" setting, it is used to evaluate the *geo-location transformation*, the *privacy analysis*, a successful execution of the *re-deployment generation* and the *adaptation planning*.

- **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. The analysis shows a privacy violation and as a result an alternative, privacy compliant re-deployment is generated. A system adaptation plan is calculated based on the re-deployment and finally executed.
- **POST**: The software system is in a privacy compliant state.

10.2.2. Scenario 2: System extension

This scenario describes the deployment runtime change. It is used to evaluate the *deployment transformation*, the *privacy analysis*, a successful execution of the *re-deployment generation* and the *adaptation planning*.

- **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 component categorized as depersonalised to the Ukrainian server.
- **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 is computed by PerOpteryx. A system adaptation plan is successfully calculated by the adaptation planning. Finally the adaptation sequence is executed.
- **POST**: The software system is in a privacy compliant state.

10.2.3. Scenario 3: Failing Adaptation

This scenario is used to evaluate the *operator-in-the-loop* during the *execution* of the adaptation sequence. Due to at least one non-automated adaptation action the operator needs to be informed.

- **PRE**: All components of the software system are hosted by multiple server instances of a cloud reseller.
- **EVENT**: The reseller migrates some of his servers to another cloud provider.
- **REACTION**: iObserve Privacy monitors the migration and starts a privacy analysis. The privacy analysis results in a privacy violation. An alternative privacy compliant deployment is computed. The adaptation calculation and planning is successful. The adaptation sequence contains actions that can not be executed automatically. The *operator* is informed about the action that can not be executed by the *adaptation execution*.
- **POST**: iObserve Privacy shows the operator the adaptation sequence with emphasis on the manual tasks.

10.2.4. Scenario 4: Missing Alternative

This scenario is used to evaluate the *operator-in-the-loop* during the *re-deployment generation*. PerOpteryx did not provide a privacy compliant re-deployment model and therefore the operator is notified.

- **PRE**: All components of the software system are hosted by multiple server instances of a cloud resellers.

- **EVENT**: The reseller starts migrating his servers to another cloud provider.
- **REACTION**: iObserve Privacy monitors the migration and starts a privacy analysis. The privacy analysis shows a privacy violation. An alternative privacy compliant deployment can not be computed. iObserver privacy detects that the re-deployment model is missing or not privacy compliant. The *operator* is notified about the situation.
- **POST**: iObserve Privacy notifies the operator about the missing privacy compliant re-deployment model.

10.2.5. Futile Scenario

There are a couple of scenarios which do not apply to iObserve Privacy, due to various reasons [9]. 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 not 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*.

10.3. Evaluation Models

In the previous sections we defined a couple of scenarios for the evaluation. In order to execute these scenarios, we need PCM Privacy models (chapter 5). Scenario and model need to get selected individually, depending on task to evaluate.

10.3.1. CoCOME-Cloud

The *CoCOME Cloud* PCM model is a representation of the CoCOME system as a distributed cloud variant. It is a representation of a supermarket IT infrastructure. It consists of six individual deployed components: *logic.webservice.cashdeskline.cashdeskservice*, *Cloud.Web*, *traidingsystem.inventory*, *traidingsystem.cashdeskline*, *webservice.inventory* and *traidingsystem.external.bank*. The system design is oriented on real distributed software systems with dozens of interfaces and multiple composite components. As a result, CoCOME-Cloud is very complex and not suited for the evaluation of specific aspects like the component categorization or the deployment analysis. However, it is as the only available model fully specified and "PerOpteryx ready". See [11] for detailed information on CoCOME.

10.3.2. Medi System

The *Medi System* is an PCM model, specially developed for the evaluation of this thesis. It is supposed to reflect the web system of a medical insurance. The required and provided interfaces are reduced to the minimal necessity, to limit side effects and to gain

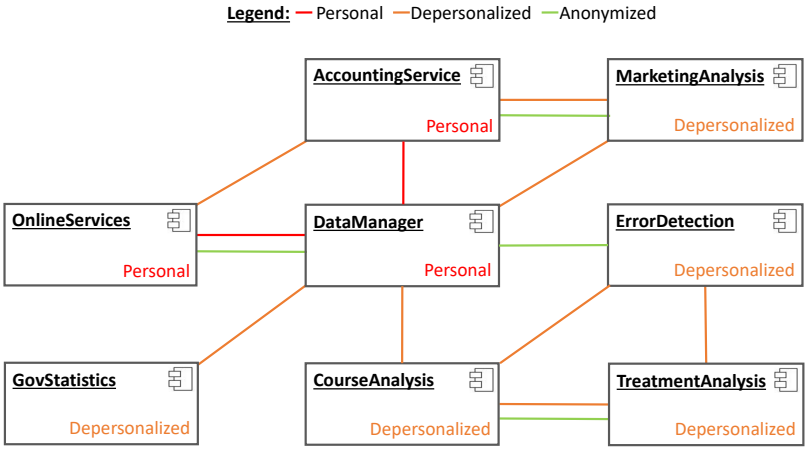

Figure 10.1.: Initial component categorization

meaningful results. Figure 10.1 shows the medi system with all components and interface connections. The deployment will depend on the evaluation scenario.

10.3.3. Generated Models

We developed a model generator and model modificator for the scalability analysis. The generator requires an input repository and creates a valid PCM Privacy model with the given amount of assembly contexts and resource container. The contained component in the assembly context is randomly selected, as well as the resource container it is allocated on. All required interfaces are correctly connected, primarily to provided interfaces without an existing connection. The privacy categorization of an Assembly Connector Privacy is randomly chosen with a distribution of 15% Personal, 35% Depersonalised and 50% Anonymized.

The model modificator adapts the system randomly, based on action counts specified. The modification supports server acquisition and termination and assembly context allocation, deallocation and migration. Further, it supports the exchange of the contained repository component for a component with the same interfaces. Note, the generated models are only suited for the scalability analysis.

10.4. Transformation

The Transformation evaluation consists of an accuracy and a scalability evaluation. The main purpose is to test the transformation of the sent information onto the architectural runtime model. Further, the iObserve privacy pipeline has to be triggered upon changes. *Scenario #1* (Subsection 10.2.1) and *Scenario #2* (Subsection 10.2.2) describe the two possible triggers: the *Deployment Event*, when a component is deployed on a server, and the *GeoLocation Event*, when the geo-location of a server changes.

10.4.1. Transformation: Accuracy Evaluation

For the accuracy evaluation we are using the CoCOME-Cloud model (Subsection 10.3.1), since it is completely specified and reflects a real-world system the most appropriate way available. We need to show, that the TDeployment, TUndeployment and TGeoLocation Transformations apply the sent data correctly onto the PCM model. Potential errors must be handled and processed. For this purpose we will use two executions. First, we will execute a logically valid input set of events, to show the correctness of the transformation. In a second execution we will show, that logically wrong inputs are processed correctly. Both runs start with an empty allocation model.

Table 10.1 shows the initial input event sequence. Initially all components get deployed, followed by a geo-location update on all servers, followed by a re-deployment of the *cloud.web* component from *Server1-EU* to *Server5-EU* and finally a geo-location update on *Server4-EU* to Ukraine.

Action	Values
Deployment	tradingsystem.external.Bank on Server6-EU
Deployment	tradingsystem.cashdeskline on Server4-EU
Deployment	cloud.web on Server1-EU
Deployment	webservice.inventory on Server1-EU
Deployment	tradingsystem.inventory on Server2-EU
Deployment	logic.webservice.cashdeskline.cashdeskservice on Server3-EU
GeoLocation	Server1-EU on 276 (GER)
GeoLocation	Server2-EU on 276 (GER)
GeoLocation	Server3-EU on 250 (FRA)
GeoLocation	Server4-EU on 250 (FRA)
GeoLocation	Server5-EU on 826 (GBR)
GeoLocation	Server6-EU on 826 (GBR)
UnDeployment	cloud.web from Server1-EU
Deployment	cloud.web on Server5-EU
GeoLocation	Server4-EU on 804 (UKR)

Table 10.1.: The correct execution set

We expect a run without any errors, an allocation model, which represents the described deployment and a design decisions model, with the according degree of freedoms.

The results are unbiased. The system reports no errors and the models represent the system exactly as intended. The *Jaccard Coefficient* is *1.0*.

To test the error behaviour, we need to input logically false events. To gain a valid system state, we are starting with the valid order (Table 10.1) and append illegal orders. Table 10.2 shows the exact execution sequence. Illegal events are marked with a *. We expect these orders to give a warning and to be ignored. The system must continue running. The test includes the following cases: Deployment of an already deployed component, deployment or undeployed on a non-existing server, geo-location record from a non-existing server, un-deployment of a non-existing deployment.

Action	Values
Table 10.1 commands	
Deployment	cloud.web on Server1-EU*
UnDeployment	cloud.web from Server5-EU
UnDeployment	cloud.web from Server5-EU*
Deployment	cloud.web on Server7-EU*
Deployment	IllegalComonent on Server1-EU*
UnDeployment	IllegalComonent from Server1-EU*
UnDeployment	tradingsystem.inventory from Server3-EU*
GeoLocation	Server7-EU on 826 (GBR)*

Table 10.2.: The error execution set

The error run shows ends up to be exactly as intended. All faulty commands got ignored and the *Jaccard Coefficient* is *1.0*. Both Jaccard coefficients show, that the (un-)deployment and geo-location transformation works as anticipated. As a consequence we argue, that the *monitoring* research question (Section 1.3) was successfully answered.

10.4.2. Transformation: Scalability Evaluation

For the scalability analysis we are using the Medi-System model with generated input. The Medi-Model is chosen due to its less generic characteristic then the Gen-Model and therefore more realistic result. Further, the Medi-Model as complex as the CoCOME model in the field to test, while easier to understand and less error prone during input generation.

The inputs are logically and syntactically valid. 30% of the inputs are deployments and un-deployments, distributed relative to the current allocation status. The other 70% of inputs are geo-location events, randomly distributed over all available servers. This ratio is an over-approximation towards the more complex and computation intensive allocation and de-allocation events. The expected real-world occurrence of a deployment events to the geo-location event is about 1 to 10000. This way, we expect the deployment and un-deployment event to have a more significant impact on the runtime behaviour. Every measurement was repeated ten times to eliminate potential measurement errors. The log outputs remain active, the snapshot creation is deactivated, so no further pipeline filters get activated. We use input sizes from 10 to one million events on a logarithmic scale.

The results (Figure 10.2) show a linear runtime behaviour, for the maximum input size of 1 million events the execution takes about 820 seconds. The according standard deviation of 22 makes it a stable and fast result.

10.5. Privacy Analysis

The *Privacy Analysis* was discussed in chapter 3 (Privacy Concept) and chapter 7 (Privacy Analysis). As described there, the privacy analysis consists of two sequential parts: *Component Classification* and *Deployment Analysis*. According to this tasks, the accuracy evaluation is also split. The accuracy evaluation uses the Medi-System model (Sub-

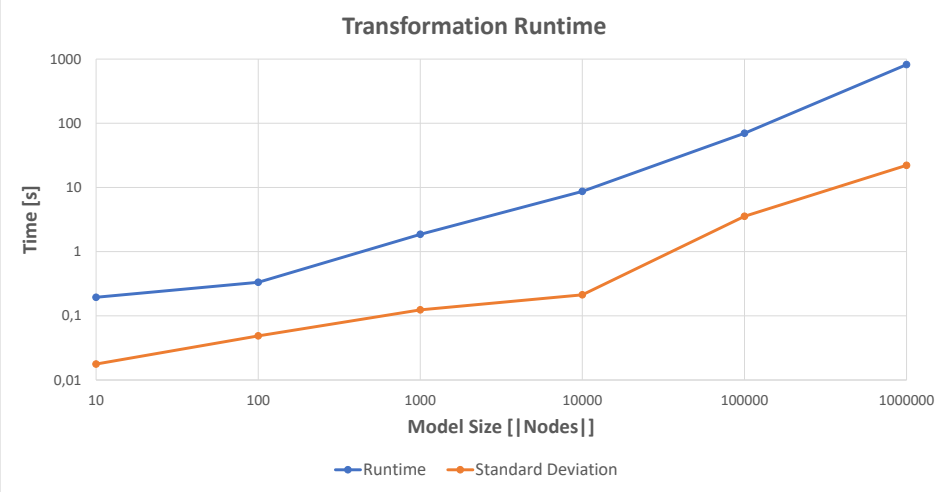

Figure 10.2.: Transformation runtime & Standard Deviation

section 10.3.2), due to its moderate complexity level, where effects like the *Joining Data Stream* occur, but the results are still traceable.

10.5.1. Privacy Analysis: Accuracy Evaluation

We will show, that the component classification categorizes components correctly by finding *joining data streams* in inter component communication. Further, we will show, that the deployment analysis finds *joining data streams* on the deployment level. As a result, we demonstrate the correctness of our privacy analysis, as specified in the goal section (Section 1.3).

The scenarios #1 (Subsection 10.2.1) and #2 (Subsection 10.2.1) aim to trigger a privacy analysis and describe different privacy violations. We showed in Section 10.4, that both trigger, the deployment event and the geo-location event are correctly processed and the information are successfully transformed to the PCM model. Both triggers lead to the same privacy analysis and are therefore equivalent.

The initial system state is as shown in Figure 10.3. The system is privacy compliant and only the *GovStatistics* component is not allocated on a server. iObserve will trigger the pipeline by receiving a GeoLocation event, which migrates the *Server2* to Belarus. We expect the initial component categorization to be equal to the most personal interface level the component has. After the *Categorization Analysis* the *MarketingAnalysis* component should be classified as personal, due to its two personal communication partners. The privacy level categorization of the components *ErrorDetection*, *TreatmentAnalysis* and *CourseAnalysis* must remain unchanged, since they share a single depersonalised interface as data source. The deployment must remain legal.

After the execution we calculated a *Jaccard Coefficient* of *1.0*. This shows us that the pipeline trigger was correctly processed. Figure 10.4 shows the initial component categorization, Figure 10.5 shows the categorization analysis result. This states show, that our expectations on the component categorization were met. The deployment analysis

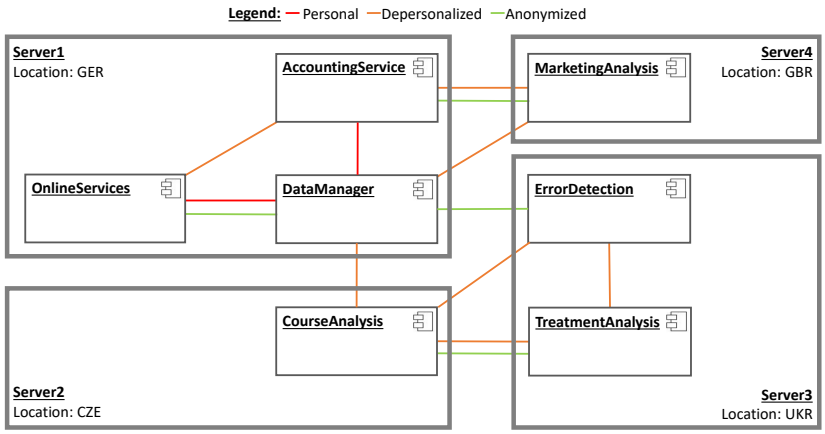

Figure 10.3.: Initial system state

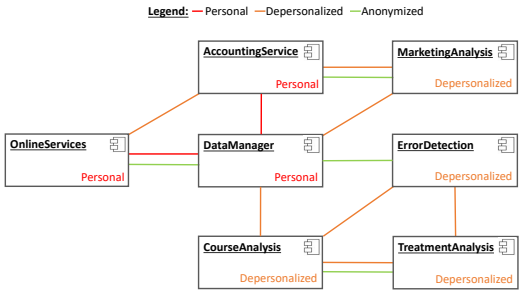

Figure 10.4.: Initial categorization

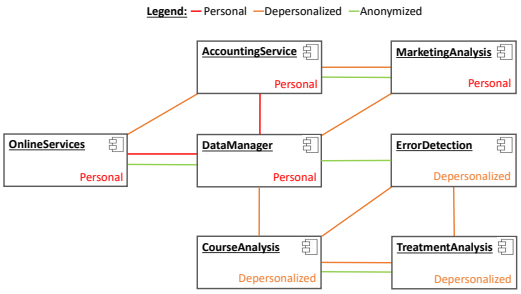

Figure 10.5.: Categorization analysis result

reports a legal deployment, which is also what we anticipated. So, the results in general are true to our expectations.

To trigger the pipeline another time, we will deploy the GovStatistics component onto Server2. The component must be tagged depersonalised and the deployment analysis must report a *joining data stream* on Server2.

The execution of the second pipeline trigger, reports a privacy violation. The cause is a joining data stream on Server2. This is what we provoked and expected. The *Jaccard Coefficient* for the trigger processing is *1.0*, again.

We have shown, that the component classification algorithm and the deployment analysis works as intended, providing a privacy analysis on a architectural level (see Section 1.3). We didn't show every possible privacy violation, however showed the most difficult analysis work: The detection of joining data streams on component categorization and deployment analysis level. And the correct identification of a set of components, located on a server, sharing the same single depersonalised component as a data source. We will use further exemplary privacy violations in the other accuracy evaluations.

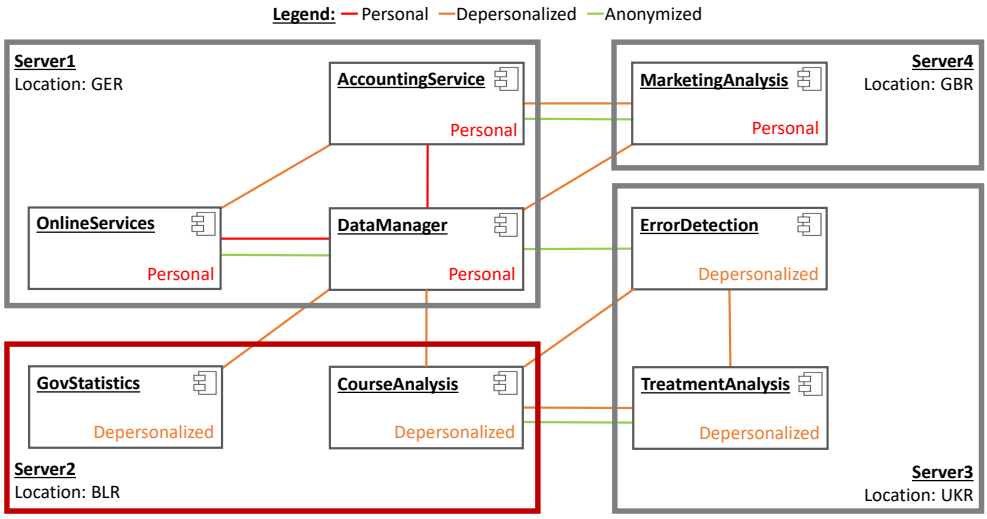

Figure 10.6.: Deployment analysis result

10.5.2. Privacy Analysis: Scalability Evaluation

For the scalability evaluation of the privacy analysis, we will use the generated model (Subsection 10.3.3), since models of the intended scale are not constructable by hand.

The time measurement starts before the graph is constructed from the model. Time rounds are taken to measure the component classification and the deployment analysis. Complex outputs are deactivated to minimise random influences.

We will measure a graph size from 10 to one million nodes on a logarithmic scale, with a distribution of 40% Resource Container and 60% Assembly Contexts. The classification of the Assembly Connectors is distributed among 15% Personal, 35% Depersonalised and 50% Anonymous. We expect this configuration to reflect a real world setting, with multiple components on a single server and only a few unused servers and an slightly increasing effort to analyse the models. The *legal* geo-locations consist of 40 random countries. Each model measurement is repeated ten times to minimize variation effects.

Figure 10.7 shows the accumulated runtime in the order the tasks are executed. Initially, the graph for the privacy analysis gets created, followed by the component classification and finally the deployment analysis. The graph shows a nearly linear increasing runtime. The major time is consumed by the graph construction, while the component classification and deployment analysis only show minor size effects. The whole runtime for one million nodes is still significantly below ten seconds. The limiting resource for further scalability tests is the graph generation. However, graphs with the size of 1000 and more nodes are already very unlikely.

The standard deviation Figure 10.8 is increasing - in general - linear with the model size. The deployment analysis however, shows some irregular behaviour. This could be due to the randomly selected save countries. A standard deviation of three times the actual mean runtime shows significantly runtime computation differences. Nevertheless, a maximum deployment analysis time of ten seconds on a one million nodes graph is still

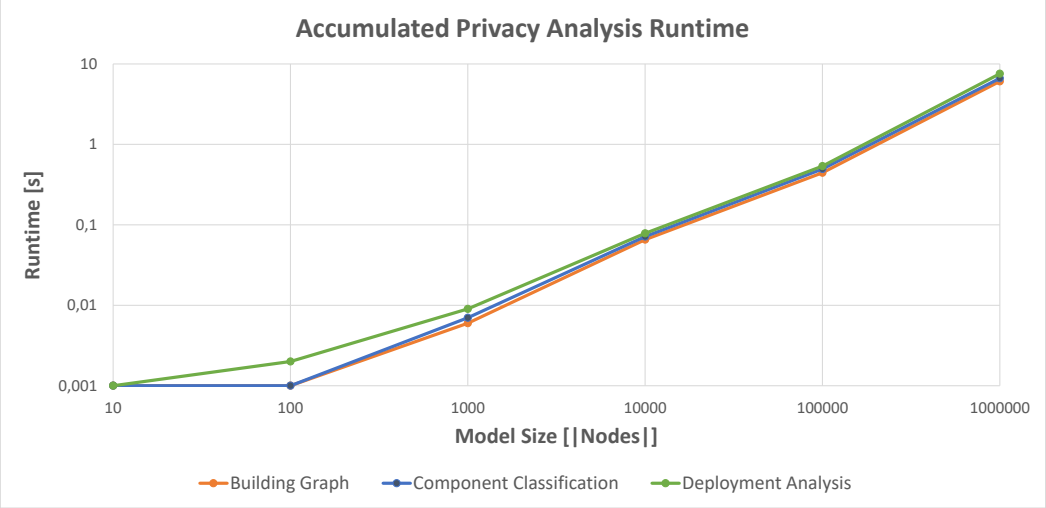

Figure 10.7.: Privacy Analysis runtime

considered very fast. And an overall privacy analysis runtime of maximal 15 seconds is also very quick.

10.6. Model Generation

The privacy compliant model generation is only evaluated under the *Accuracy* aspect. The *Scalability* evaluation was already performed by the creators of PerOpteryx (see [15]) and our Privacy Analysis was extensively evaluated in Section 10.5. We will focus on the in Section 1.2 specified problem of generating a privacy compliant redeployment model.

For this evaluation we will use the CoCOME model (Subsection 10.3.1) together with the scenarios #2 (Subsection 10.2.2) and #4 (Subsection 10.2.4). Scenario #2 describes a complete iObserve Privacy execution, including the successful execution of PerOpteryx, the model generation. Scenario #4 describes the case, in which PerOpteryx fails to generate a privacy compliant candidate. We will start with Scenario #2.

We deploy the CoCOME components onto one EU server each and adjust the individual server costs to reflect EU and Non-EU status. We will trigger the pipeline by moving the *Server4-EU*, which hosts the as personal categorized component *webservice.inventory*, to Ukraine.

After the execution of our model generation framework, *PerOpteryx*, we expect the re-deployment model to be privacy compliant. This is the major concern and primary focus. However, we further anticipate a deployment with fewer server in use, as well as a migration of multiple *depersonalised* components onto Non-EU servers.

PerOpteryx is configured to execute *four iterations* with *20 candidates* per iteration. This means, a total of 80 candidates will be created and evaluated.

Table 10.3 shows the component, their data privacy level classification, the initial deployment and the generated re-deployment. An re-deployment model privacy analysis

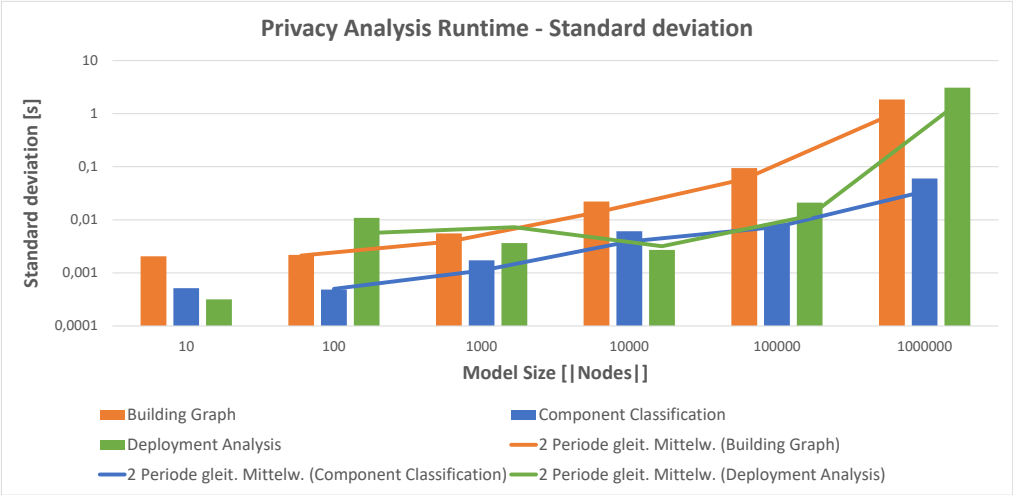

Figure 10.8.: Privacy Analysis runtime standard deviation

Component	Categorization	Deployment	Redeployment
cloud.web	Depersonalized	Server1-EU	Server6-EU
tradingsystem.inventory	Personal	Server2-EU	Server1-EU
cashdeskline.cashdeskservice	Depersonalized	Server3-EU	Server2-NonEU
webservice.inventory	Personal	Server4-EU	Server6-EU
tradingsystem.cashdeskline	Depersonalized	Server5-EU	Server4-EU
tradingsystem.external.Bank	Depersonalized	Server6-EU	Server3-NonEU

Table 10.3.: Component categorization, runtime deployment and re-deployment

shows, that the model is privacy compliant. The major goal (see Section 1.3) was accomplished. Further, depersonalised components were moved to more cheap Non-EU severs and two components were deployed onto the same server. Both deployment goals indicate, that the *most cost efficient* model was chosen - as intended. However, no component stays at this original server, which produces (unnecessary) migration costs. The execution time of PerOpteryx was about 30 seconds and elven of 80 candidates were privacy compliant.

In scenario #4, no privacy compliant re-deployment model can be found, PerOpteryx crashes or any kind of error occurs. However, the end result stays unchanged, no valid re-deployment model is available after the PerOpteryx execution. We will provoke this situation by deploying all components onto a single server and moving this server to a Non-EU geo-location. We expect iObserve Privacy to report, that no privacy compliant PCM model was found. Basically, we expect, that iObserve invokes the *operator-in-the-loop* for manual error treatment.

The execution shows an error, which states, that the given re-deployment model is not privacy compliant. If a listener would be registered, the listener would have been notified. The execution lives up to our expectations.

10.7. Adaptation Planning

The Adaptation Planing, described in chapter 9, aims for calculating an sequence of adaptation actions. The execution of this adaptation order must result in a runtime model, that is equivalent to the redeployment model. This is one of the research goals stated in Section 1.3. We will evaluate this task towards its accuracy and scalability aspects.

10.7.1. Adaptation: Accuracy Evaluation

We will continue with scenario #2 from generation evaluation (Section 10.6) to show the continuous flow and re-establishment of the privacy compliance. To show, that the given adaptation order leads to a privacy compliant system, we will simulate the adaptation. This will be achieved by translating the adaptation order to an equivalent iObserve privacy input. After iObserve privacy computed the input, the *Jaccard Coefficient* - between the computed re-deployment model from PerOpteryx and the iObserve runtime model - must be *1.0*.

We expect the adaptation sequence to initially *acquire* the newly used servers, followed by a series of *migration* actions, that move the components to their new servers. Finally there should be three *terminate* actions to release the servers, that are no longer needed.

Action	Values			
Acquire	Server3-NonEU			
Acquire	Server2-NonEU			
Migrate	webservice.inventory	Server4-EU	->	Server6-EU
Migrate	tradingsystem.cashdeskline	Server5-EU	->	Server4-EU
Migrate	tradingsystem.external.Bank	Server6-EU	->	Server3-NonEU
Migrate	cashdeskline.cashdeskservice	Server3-EU	->	Server2-NonEU
Migrate	cloud.web	Server1-EU	->	Server6-EU
Migrate	tradingsystem.inventory	Server2-EU	->	Server1-EU
Terminate	Server3-EU			
Terminate	Server2-EU			
Terminate	Server5-EU			

Table 10.4.: The ordered adaptation sequence

Table 10.4 shows the output adaptation sequence for scenario #2. The result is unbiased. The in Table 9.2 stated universal order is met. For the transformation onto iObersve input events, we can ignore the acquire actions, since the resource containers already exist in the resource environment model. Every migrate action needs to be translated into two events: a TUndeployment event and a TDeployment event. The terminate can also be ignored, since there is no real server to release/terminate. We expect the system to be in a

privacy compliant state after the execution of the iOberve input events. Table 10.5 shows the resulting iObserve input.

Action	Values
UnDeployment	webservice.inventory from Server4-EU
Deployment	webservice.inventory on Server6-EU
UnDeployment	tradingsystem.cashdeskline from Server5-EU
Deployment	tradingsystem.cashdeskline on Server4-EU
UnDeployment	tradingsystem.external.Bank from Server6-EU
Deployment	tradingsystem.external.Bank on Server3-NonEU
UnDeployment	cashdeskline.cashdeskservice from Server3-EU
Deployment	cashdeskline.cashdeskservice on Server2-NonEU
UnDeployment	cloud.web from Server1-EU
Deployment	cloud.web on Server3-EU
UnDeployment	tradingsystem.inventory from Server2-EU
Deployment	tradingsystem.inventory on Server1-EU

Table 10.5.: iObserve input event translated adaptation sequence

The execution of the transformed adaptation sequence (Table 10.5) results in a privacy compliant PCM model and the *Jaccard Coefficient* of the post-execution runtime model and the calculated redeployment model is *1.0*. Frankly speaking, the result is unbiased and solves the in Section 1.2 specified problem of *automated privacy compliance re-establishment*.

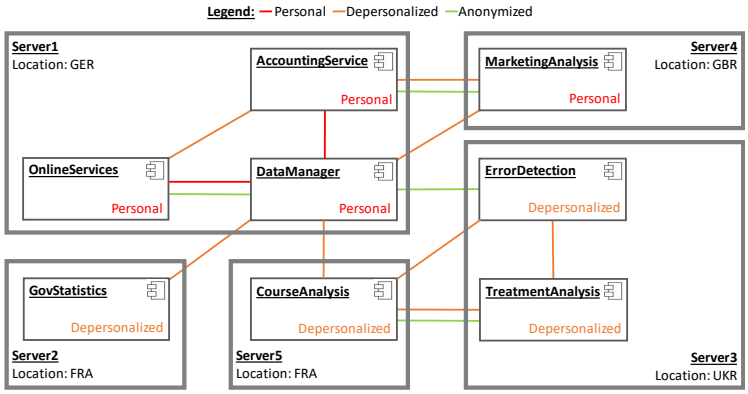

Figure 10.9.: Runtime model

We will continue to evaluate the *System Adaptation* with scenario #3 to show the limitations and so far unseen capabilities of the adaptation calculation. For this scenario the Medi-Sys will be used.

PerOpteryx only adapts the system given according to the *design decision* model, it will only produce acquire, migrate and terminate actions - at least in our case. These are (usually) automatically executable. However, there are more potential actions specified, which can not be executed automatically. We will modify the Medi-Sys model to provoke one of

each action. Figure 10.9 shows the runtime model, Figure 10.10 shows the re-deployment model. The anticipated difference between these models is shown in Table 10.6. We expect each of the seven actions to be detected, except the *replicate* action, since this is not (yet) supported by the adaptation calculation. It will be subsided by an *aquire Server2-2* and an *allocate GovStatistics on Server2-2*.

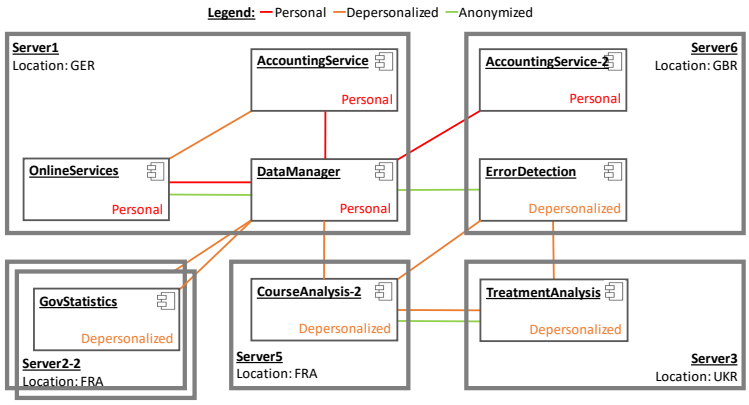

Figure 10.10.: Redeployment model

Action	Values
Acquire	Server-6
Exchange Component	CourseAnalysis to CourseAnalysis-2
Deallocate	MarketingAnalysis from Server-4
Allocate	AccountingService-2 on Server-6
Migrate	ErrorDetection from Server-3 to Server-6
Replicate	Server-2 (with GovStatistics)
Terminate	Server-4

Table 10.6.: Expected adaptation sequence for Scenario #4

The execution of the adaptation calculation shows the expected result (see Table 10.7). The replicate action is split into the expected *acquire* and *allocate* actions (marked with an *).

what is not automatically executable?

10.7.2. Adaptation: Scalability Evaluation

For the scalability analysis we are initially generating logically valid PCM Privacy models. These models are then randomly modified with respect to the logically validity of the model. This means e.g., that components deployed on a terminated server must be migrated to another server. And finally, we run the *Adaptation Calculation* and *Adaptation Planning* stages. We will measure the runtimes of the graph creation, the adaptation calculation and the adaptation planning to gain a detailed runtime behaviour for each task.

Action	Values
Acquire	Server-6
Acquire	Server2-2 (*)
Exchange Component	CourseAnalysis to CourseAnalysis-2
Allocate	AccountingService-2 on Server-6 (*)
Allocate	GovStatistics-2 on Server2-2
Deallocate	MarketingAnalysis from Server-4
Migrate	ErrorDetection from Server-3 to Server-6
Terminate	Server-4

Table 10.7.: Expected adaptation sequence for Scenario #4

The initial model is constructed in a way, that a model of size 100 consists of 60 assembly contexts and 40 resource containers. The modification is linear distributed over the acquire, terminate, allocate, deallocate, migrate and exchange component related modification. However, not every modification in the model leads to an adaptation action. The adding of another resource container is only recognized as an acquire action, if an assembly context is allocated on that new resource container. The termination of a resource container, that hosts one or more components, leads also to a migration action. With this in mind, we measure the time of the adaptation calculation and adaptation planning based on the amount of modifications in the model. The model itself is three times as big as the modifications provoked. Each measurement is repeated ten times, the input sizes are 10, 100, 1000 and 10000. An evaluation run with 100000 provoked changes was aborted after twelve hours of computation.

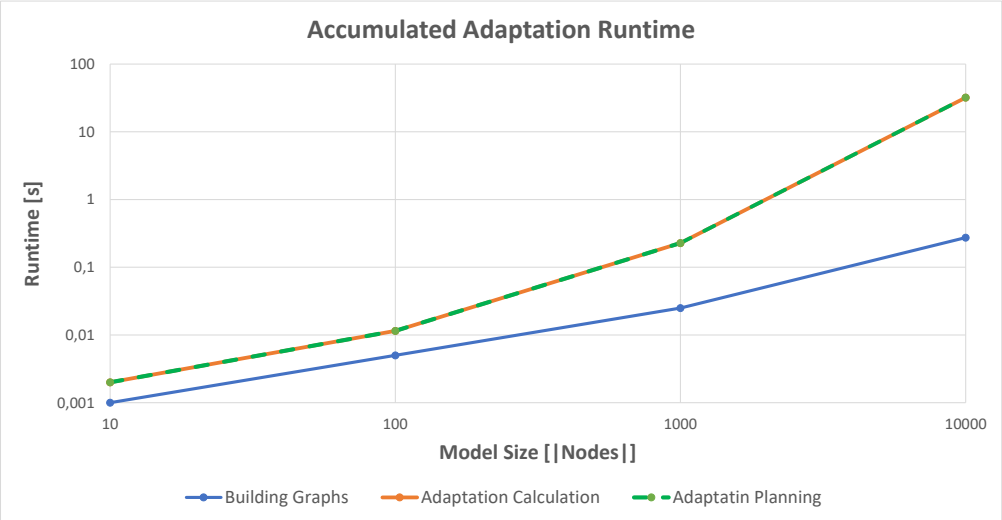

Figure 10.11.: Adaptation runtime

Figure 10.11 shows the runtime results. The graph creation shows a linear runtime behaviour, like in Subsection 10.5.2 already noticed. The adaptation calculation consumes

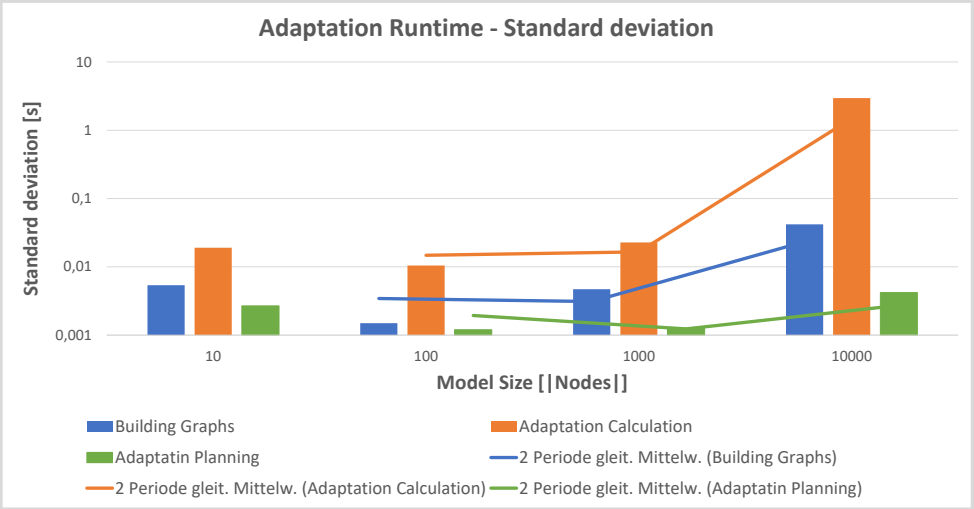

Figure 10.12.: Adaptation runtime SD

an increasingly amount of time. This can be explained with the increasing model size, since the actual action calculation is optimized for $O(1)$ operations, while the action creation operates with PCM model elements directly. The EMF framework in general shows poor performance characteristics, which can also be seen at the failed 100000 changes evaluation. The adaptation planning needs next to no time.

The standard deviation (Figure 10.12) shows mostly constant deviations. However, the adaptation calculation increases significantly on the last evaluation run. We assume this is due to JVM memory management and EMF effects. However, a standard deviation of roughly 3 seconds to a runtime of 30 seconds is within an acceptable ratio, considering that two models and graphs of 30000 nodes have to be managed.

The adaptation runtime in general shows satisfying results. The calculation can be considered very fast with an acceptable variance.

10.8. Threats to validity

Like in any scientific publications, the evaluation has threats to its validity. We will scope the most obvious points and highlight the crucial aspects.

10.8.1. Internal Validity

Our decision to evaluate every major iObserve Privacy task independently is a key stone to the internal validity. It allowed us to provoke every possible reaction and effect without potential side effects. As a result, we could perfectly track cause and result of certain effects.

Scalability tests often require an even more isolated and specialised testing. In our case the individual task had to be called from outside their ordinary call scheme, the teetime framework. This was necessary so enable tests of the performed size. This blurs the results in the context of the whole system, however, sharpens the individual tasks runtime behaviour.

10.8.2. External Validity

One drawback towards external validity are the missing live tests. Since there is (currently) no software system available to generate live events, we created inputs by hand. The input is logically and syntactically valid, however, potential corner cases could have been missed, despite extensive testing. Further, due to the missing live system, all test aiming for a real world authenticity were using the CoCOME PCM model. The CoCOME system is primarily designed to provide scientific comparable, near real world application. Nevertheless, the iObserve Privacy functionality has been extensively evaluated with real world equivalent inputs.

10.8.3. Construction Validity

While the individual task evaluation was the internal validities strength, it is a weakness of the construction validity. However, during the accuracy evaluations, the whole pipeline was (usually) invoked, evaluation scenarios were continued and an internal logically validity was established. So, the evaluation is well interleaved and the likelihood of construction errors should be, despite the individual task evaluation, minimized.

Nevertheless, the target evaluation models are constructed by hand, which tends to be error prone. However, to produce the same error in the evaluation target model like iObserve Privacy does is very unlikely.

10.8.4. Conclusion Validity

The accuracy evaluation is considered very conclusive with sufficient tests, scenarios and models. Most of the scalability evaluations are also quite conclusive. However, the bad EMF performance limits the test sizes. This is especially true for the adaptation planning scalability evaluation, with the maximum input of 10000 changes. This makes the result not as meaning as possible, but still viable.

11. Related Work

In this chapter we are presenting a couple of related publications, this work can be referenced to. The sections are oriented on the task and problems mentioned in Section 1.3.

11.1. 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 11.2) 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 kind of PCM-based applications to use the gathered information.

gather more details & Refs

11.2. 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.

Like Kieker, R-PRIS combines push-based heartbeat monitoring with event processing, and graph grammars for efficiently updating those models [21]. 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. This means, a data modelling is required with a *Classification* and the containing *ContentType* (see Figure 11.1). 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 11.2) - with its meta-model (Figure 11.1) - R-PRIS performs a reachability check [22].

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

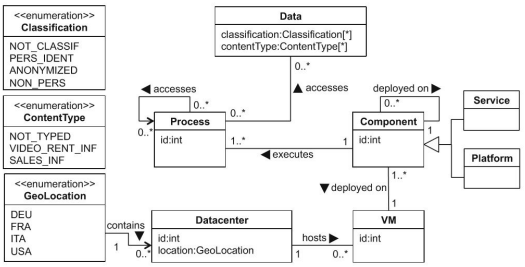

Figure 11.1.: R-PRIS meta-model

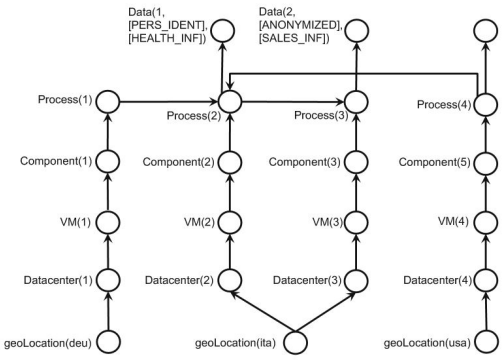

Figure 11.2.: R-PRIS runtime model

exactly this case, which however doesn't necessarily communicate private data. As a result, a lot of policies have to be specified, which prohibit many potentially harmless communication paths.

Based on their runtime model, Schmieders et al. identified four relevant migration-cases and extracted six required informations to detect a policy violation[22]:

#	**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 11.1.: R-PRIS information for runtime privacy checks [22]

Due to the comparable core task of detecting privacy violations, we are comparing the R-PRIS privacy analysis against the iObserve Privacy privacy analysis. More detailed information on R-PRIS can be found in [21][22].

Runtime Model R-PRIS uses the specially developed model shown in Figure 11.1. Even while it models components, VMs and process, it does not capture the systems architecture as the Palladio Component Model does. Further, it is not known, whether tool support or other compatible programs exist, like the PCM has.

Categorization We are utilizing a component communication classification, which leads to a component classification and a deployment analysis. Therefore, we do not know what data are actually processed in a component or on a server. R-PRIS, however, tags the data itself, traces the transitive access and prohibits rule violating access. As a result, R-PRIS, uses the more accurate *data tracking*, which requires more information then and a deep knowledge about the observed system then iObserve privacy. While iObserve privacy uses data categorization by hand, the R-PRIS approach does not explain their modelling and categorization process.

Rule Compliance iObserve Privacy is clearly build to stay EU General Data Protection Regulation and HIPAA compliant. Therefore, a simple file input with *save* geo-locations is sufficient. R-PRIS requires a rule input, which specifies prohibited data access. Compared to iObserve privacy, this is a more powerful and flexible input. Nevertheless, it complicates the usage and scales poorly with the system size, deployment locations and data variety, since every prohibited access needs to be specified for every data type per geo-location and content type.

11.3. 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 [7].

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* [23], *TaintDroid* [8], *Privacy Oracle* [14] or *automated privacy instrumentation* [26] 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 early stages and therefore not suited for applications with our designated level of complexity.

11.4. 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.”[22]

As mentioned in Section 11.2, R-PRIS is searching for potential access violations in the application model, by using a st-connectivity analysis.[22][21] 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 the high likelihood of allowing only deployments on save-considered geo-locations.

11.5. 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.[17]

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.

11.6. Automated Cloud Migration & Adaptation

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. [12][3]

12. Conclusion

In this chapter we will wrap up this thesis with the *limitations* and the *future work* sections.

12.1. Limitations & Assumptions

Like every other scientific work, we can't build a universal, world ready system. We need to accept and sometimes even require limitations to produce meaningful results for certain aspects.

Cloud Provider objectives A cloud provider, like any other person or company, hast own objectives. In the most cases *profit maximization* can be assumed as the primary goal. This can be interpreted in multiple ways, from law in-compliant behaviour over SLA violations to premium prices for extended 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 are law and SLA compliant.

Separation of virtual servers For simplicity reasons, we need to assume that we can deploy multiple Type 1 Data, *depersonalised data*, (Section 3.1) onto one data-centre, but on different (virtual) server, without considering *joining data stream* (Subsection 7.1.3) implications. Basically we assume, every virtual server has its own independent disk and memory storage. If this assumption wouldn't be made, massive per-instance encryption or per data-centre deployment would be the valid solution. However, encryption as a cloud-ready middle wear is a hot research topic around the globe and not considered by this thesis.

Geo-location API To make a statement about the systems current privacy compliance, we need the Resource Containers geo-location. If we don't want to make extensive geo-location determination process, like the *ping round trip measurement*, we need the cloud provider to provide the geo-location via his cloud API.

Closed World Assumption As mentioned in Section 3.3 we need the close world assumption to make any statement about the privacy compliance. The implications of privacy and data protective laws are too complex to make a automated, detailed and well balanced statement on privacy compliance without the CWA.

Privacy Analysis Overestimation For the privacy analysis we forbid *joining data streams* (Subsection 7.1.3). We are aware, that this is a considerably overestimation, especially during the deployment analysis. We are doing so to ensure privacy compliance without taking any chances and prevent deep component inspection and extensive data protective law discussions.

(In-)Correct Component Based Architecture Modern software systems and distributed cloud applications in particular, are designed after the *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 incorrect data privacy level. An example is a component that receives personal information via an user interface (e.g. Graphical User Interface) and saves or processes them itself breaks this principle. We argue that such a monolithic system stands contrary to the fundamental idea of distributed cloud systems and were ignored during this thesis.

12.2. Future Work

During the work on iObserve Privacy a couple of future oriented tasks and development directions came visible to improve iObserve. In the following we are introducing a couple of them.

Thesis merge *B. Sc. Tobias Pöppke* developed in his master thesis, *Design Space Exploration for Adaptation Planning in Cloud-based Applications*, another iObserve modification. His modification aims for the automated support of modern cloud system. The development of our iObserve systems happened under close cooperation and is therefore well aligned for merging.

PerOpteryx integration *PerOpteryx* provides one of the core features of iObserve Privacy, as a model generation framework. However, its huge dependencies, immense complexity and plug-in architecture makes it nearly impossible to directly migrate it into iObserve Privacy. Even small modifications take major effort. The changed mechanic must be well understood to prevent the system from breaking while modifying. A well thought and designed re-engineering is required to keep the core functionality while reducing dependencies and complexity to a minimum. Such a radical re-development effort should not be taken lightly, however would make future extensions way easier.

Live tests Due to a missing distributed test system, iObserve Privacy could not be tested in a real situation. Even though many test were run during the evaluation (see chapter 10) and proven *Kiker* concepts were used, a live test provides further reassurance and validity to the system as a whole.

Bibliography

[1] Andale. *Jaccard Index / Similarity Coefficient*. Ed. by Statistics How To. 2016. URL: http://www.statisticshowto.com/jaccard-index/ (visited on 06/28/2017).

[2] Steffen Becker, Heiko Koziolek, and Ralf Reussner. "The Palladio component model for model-driven performance prediction". In: *Journal of Systems and Software* 82.1 (2009), pp. 3–22. ISSN: 01641212. DOI: 10.1016/j.jss.2008.03.066.

[3] 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.

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

[5] Kashif Dar. *MAPE-K ada MAPE-K adaption control loop*. Ed. by University of Oslo. 2012. URL: http://www.uio.no/studier/emner/matnat/ifi/INF5360/v12/undervisningsmateriale/MAPE-K%20adap%20control%20loop.pdf (visited on 11/06/2016).

[6] 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).

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

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

[9] 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.

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

[11] Robert Heinrich et al. "A Platform for Empirical Research on Information System Evolution". In: *The 27th International Conference on Software Engineering and Knowledge Engineering*. International Conferences on Software Engineering and Knowledge Engineering. KSI Research Inc. and Knowledge Systems Institute Graduate School, 2015, pp. 415–420. DOI: 10.18293/SEKE2015-066.

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

[13] 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).

[14] 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.

[15] Anne Koziolek. *Automated Improvement of Software Architecture Models for Performance and Other Quality Attributes: Zugl.: Karlsruhe, Karlsruher Institut für Technologie (KIT), Diss., 2011*. Vol. 7. The Karlsruhe Series on Software Design and Quality / Ed. by Prof. Dr. Ralf Reussner. Karlsruhe: KIT Scientific Publishing, 2014. ISBN: 9783866449732.

[16] Anne Koziolek. *PerOpteryx - SDQ Wiki*. 2017-06-27. URL: https://sdqweb.ipd.kit.edu/wiki/PerOpteryx (visited on 06/27/2017).

[17] 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.

[18] 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).

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

[20] Prof. Dr. Wilhelm (Willi) Hasselbring. *Kieker Framework | Kieker*. 2017. URL: http://kieker-monitoring.net/framework/ (visited on 06/26/2017).

[21] 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.

[22] 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.

[23] 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.

[24] 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).

[25] Misha Strittmatter et al. "A Modular Reference Structure for Component-based Architecture Description Languages". In: *2nd International Workshop on Model-Driven Engineering for Component-Based Systems (ModComp)*. CEUR, 2015, pp. 36–41. URL: http://ceur-ws.org/Vol-1463/paper6.pdf.

[26] 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.

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

[28] Tobias Pöppke. "Design Space Exploration for Adaptation Planning in Cloud-based Applications". Master thesis. Karlsruhe: Karlsruher Institut für Technology, 2017-06-26. (Visited on 06/26/2017).

[29] Philipp Weimann. *iObserve Privacy Evaluation Data & Results*. 2017-06-25. URL: https://github.com/cocome-community-case-study/cocome-cloud-jee-privacy/tree/master/Evaluation (visited on 06/25/2017).

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

[31] Wikipedia. *Surface Pro 4 - Wikipedia*. Ed. by Wikipedia. 2017-06-13. URL: https://en.wikipedia.org/w/index.php?oldid=783726963 (visited on 06/29/2017).

A. Appendix

A.1. First Appendix Section

Figure A.1.: A figure

...