Uncertainty-aware Confidentiality Analysis Using Architectural Variations

Bachelor's Thesis of

Tizian Bitschi

at the Department of Informatics
KASTEL – Institute of Information Security and Dependability

Reviewer:	Prof. Dr. Ralf H. Reussner
Second reviewer:	Prof. Dr.-Ing. Anne Koziolek
Advisor:	M.Sc. Sebastian Hahner
Second advisor:	M.Sc. Maximilian Walter

June 21, 2022 – October 21, 2022

I declare that I have developed and written the enclosed thesis completely by myself. I have submitted neither parts of nor the complete thesis as an examination elsewhere. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. This also applies to figures, sketches, images and similar depictions, as well as sources from the internet.

Karlsruhe, 2022.10.21

. . . .

(Tizian Bitschi)

Abstract

When planning software architectures, many aspects of the system to be developed are still unclear. The larger the architecture, the more design decisions it contains and the more uncertainty exists about the system. Even if a detailed design of the architecture already exists, there may still be uncertain aspects, such as user inputs Software systems usually contain data that only authorized persons should have access to. In addition, companies have to comply with laws that specify the usage or storage policies of data. Violations of confidentiality requirements can cost companies high fines. Therefore, it is strongly advised to consider confidentiality compliance in the design process, even if the architecture is not yet fully defined.

Previous approaches that analyze confidentiality provide skewed results when uncertainties are attempted to be included, or ignore uncertainty entirely. Currently, to be able to check whether violations of confidentiality policies occur in an architecture, it must be examined where uncertainty occurs and what effect it has on elements in the architecture. Furthermore, a large architecture can become very complex, making it more difficult to find all the places where confidentiality violations can occur. This poses a hurdle for software architects to intercept violations of confidentiality in their entirety and to maintain confidentiality after changes in the design.

In this thesis, we present an approach that combines the results of existing confidentiality analyses with modeling capabilities about uncertainties. For this, we develop an algorithm that receives information about the uncertainty, the architecture and the confidentiality breaches that have occurred as input. This results in an analysis that examines uncertainties regarding their impact on a violation of confidentiality and shows a software architect more quickly where and why violations of confidentiality occur. The algorithm is designed for the possibility of future expansion.

We evaluate our approach by applying it to case studies and checking its usability and feasibility. Under the assumption that we can describe architectural uncertainty as variations that lead to scenarios of the architecture, the results indicate that the approach can filter uncertainty impacts that are not responsible for confidentiality violations.

Zusammenfassung

Bei der Planung von Software-Architekturen sind viele Aspekte des zu entwickelnden Systems noch unklar. Je größer die Architektur ist, desto mehr Entwurfsentscheidungen enthält sie und desto mehr Ungewissheiten bestehen über das System. Selbst wenn bereits ein Feinentwurf der Architektur existiert, kann es immernoch ungewisse Aspekte geben, wie z. B. Nutzereingaben. Softwaresysteme enthalten in der Regel Daten, auf die nur autorisierte Personen Zugriff haben sollen. Darüber hinaus müssen Unternehmen Gesetze einhalten, die die Verwendung oder Speicherung von Daten vorschreiben. Verstöße gegen die Vertraulichkeit von Daten können Unternehmen teuer zu stehen kommen. Daher ist es dringend ratsam, die Einhaltung von Vertraulichkeitsanforderungen bereits im Entwurfsprozess zu berücksichtigen, auch wenn die Architektur noch nicht vollständig definiert ist.

Bisherige Ansätze zur Analyse der Vertraulichkeit liefern verfälschte Ergebnisse, wenn Ungewissheiten einbezogen werden sollen oder ignorieren Ungewissheit vollständig. Um überprüfen zu können, ob in einer Architektur Verstöße gegen Vertraulichkeits-Richtlinien auftreten, muss derzeit untersucht werden, wo die Ungewissheit auftritt und welche Auswirkungen sie auf die Elemente der Architektur hat. Außerdem kann eine große Architektur sehr komplex werden, so dass es schwieriger wird, alle Stellen zu finden, an denen Vertraulichkeitsverletzungen auftreten können. Dies stellt eine Hürde für Softwarearchitekten dar, Vertraulichkeitsverletzungen in ihrer Gesamtheit abzufangen und Vertraulichkeit nach Änderungen im Entwurf zu erhalten.

In dieser Arbeit stellen wir einen Ansatz vor, der die Ergebnisse bestehender Vertraulichkeitsanalysen mit Modellierungsfähigkeiten über Ungewissheiten kombiniert. Hierfür entwickeln wir einen Algorithmus, der als Eingabe Informationen über die Ungewissheit, die Architektur und die aufgetretenen Vertraulichkeitsverletzungen erhält. Das Ergebnis ist eine Analyse, die Ungewissheiten hinsichtlich ihrer Auswirkungen auf eine Vertraulichkeitsverletzung untersucht und einem Softwarearchitekten schneller zeigt, wo und warum Vertraulichkeitsverletzungen auftreten. Der Algorithmus ist auf die Möglichkeit einer zukünftigen Erweiterung ausgelegt.

Wir evaluieren unseren Ansatz, indem wir ihn auf Fallstudien anwenden und die Nutzbarkeit und Durchführbarkeit prüfen. Unter der Annahme, dass wir architektonische Ungewissheit als Variationen beschreiben können, die zu Szenarien der Architektur führen, zeigen die Ergebnisse, dass der Ansatz Ungewissheitseinflüsse herausfiltern kann, die nicht für Vertraulichkeitsverletzungen verantwortlich sind.

Contents

List of Figures

List of Tables

1. Introduction

In a world where more and more information is shared and more devices are connected, maintaining confidentiality becomes increasingly important. If companies do not follow the guidelines of confidentiality (e.g. the GDPR [25]), they suffer great damage. The punishments harm them both financially and in public confidence. In May 2020 alone, violations of the General Data Protection Regulation (GDPR) imposed fines of around €315 million in the UK and around €25 million in Germany [28]. The later companies repair such violations, the more expensive it becomes for them [4]. Therefore, it is strongly advised to include confidentiality early in the planning of software.

When planning software architectures, many aspects of the system to be developed are still unclear. We distinguish between uncertainty that stems from external circumstances and uncertainty that stems from not yet knowing how elements in the software architecture will be modeled in the final system. An example of external circumstances would be that a server can no longer be operated in a certain country due to an unforeseeable political decision. In order to keep the situation simple for this thesis, we focus on the uncertainties that can be described by concrete scenarios. These scenarios arise from the variation modeling of uncertainty. To model variations, the impact of an uncertainty is described by a collection of concrete Architectural Design Decisions (ADDs). A scenario then can be generated by choosing one of the ADDs for each uncertainty.

The larger the architecture, the more ADDs it contains and the more uncertainty exists about the system. If you want to check whether violations of confidentiality occur in the architecture, you are faced with a difficult task. First, it must be examined where uncertainty occurs and what effect it has on elements in the architecture. However, this may require extensive knowledge that is not yet available or difficult to predict at this point in time. Therefore, such analyses are only carried out when sufficient knowledge is available, but then it might already be too late to revert questionable ADDs. Also, previous approaches that analyze confidentiality yield skewed results when attempting to include uncertainties, or ignore uncertainties entirely. A large architecture can become very complex, making it more difficult to find all the places where confidentiality violations can occur. This poses a hurdle for software architects to intercept violations of confidentiality in their entirety and to maintain confidentiality after changes in the design.

The contribution of this thesis is to introduce a concept that bridges the gap between existing confidentiality analyses and uncertainty. The approach consists of using results from existing confidentiality analyses and relating them to the uncertainty impacts. With our approach, we want to achieve that architectures can be analyzed for confidentiality under uncertainty at design time. The analysis is intended to help a software architect to find the causes of violations of confidentiality more quickly.

We evaluate our approach by applying it to case studies and determining the usability of the approach for a software architect. We also check whether our approach is fundamentally feasible, i.e. whether it delivers a result and whether uncertainty impacts can be linked to scenarios. Applying the approach to case studies gives us the accuracy with which our approach can classify uncertainty impacts as potentially responsible. Under the assumption that we can describe architectural uncertainty as variations that lead to scenarios of the architecture, the results indicate that the approach can filter uncertainty impacts that are not responsible for confidentiality violations. The concept is designed for the possibility of future expansion.

This thesis is divided into seven chapters. In this chapter, the topic was introduced and motivated. In Chapter 2 we explain the foundations of our work. This includes the Palladio Component Model (PCM), data flow with data flow diagrams and uncertainty. In Chapter 3 we present the state of the art in the interpretation of the results of confidentiality analyses, the influence of uncertainty and confidentiality analyses. To illustrate our concept, we present an example model in Chapter 4 that serves as a running example. In Chapter 5 we present our approach. We explain the data model used, how we reuse existing approaches in our algorithm and how our contribution, the uncertainty impact filter, is structured. We evaluate our approach in Chapter 6, draw our conclusion in Chapter 7 and give an outlook on future work.

2. Foundations

This chapter describes the basics that are necessary for this thesis. The PCM is presented in Section 2.1. We explain data flow diagrams and their representation in Section 2.2. Section 2.3 deals with uncertainty and what it means for a software architecture.

2.1. Palladio Component Model

The PCM is part of the Palladio approach, with which software architectures are modeled and simulated [20]. It consists of five different models that can be used to represent the structure, deployment, execution environment, usage profile, and behavior. In the Palladio approach, the task of a component developer is to develop components. A software architect then assembles these into architectures. "A software component is a contractually specified building block for software, which can be composed, deployed, and adapted without understanding its internals" [20]. This means that the interior of the component does not play any further role when planning the software architecture for the time being. Components offer interfaces and use the interfaces of other components. Only the pre- and post-conditions of the components are modeled in the software architecture. For this, components can be given properties in PCM that may be functional or qualitative.

We use the PCM to model architectures and to develop our prototype [3].

2.2. Data Flow Diagrams

A data flow diagram describes the flow of data through a system [8]. A flow of data is created by data being input, forwarded and output. This is described in a data flow diagram without requiring control flow. The system is therefore presented from the point of view of the data. Graphical notations are used to represent these data flows. DeMarco presents the following element types that are used in the graphical representation [8]:

- **Storage** represented by two parallel lines in the name
- **Data flow** represented by arrows and names
- **Functions** in a system represented by circles or ovals
- **Interfaces**, i.e. input and output of data, creates rectangles

A typical data flow is created by entering data at a data source. This data is forwarded to storage and processed by processes in the system. Finally, data can also be released from the system again by being transferred to a data sink.

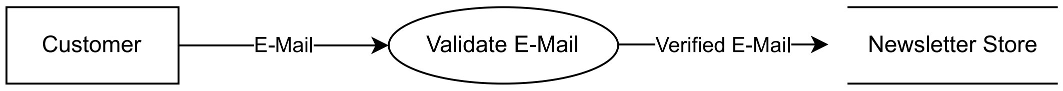


Figure 2.1.: Simple example of a data flow diagram showing a newsletter registration

In Figure 2.1 we see a simple example of a data flow diagram. Here the customer registers in a newsletter. To do this, he gives his e-mail address. It is then verified by the system and the verified email address is then saved in the newsletter storage.

Data flows provide the basis for detecting breaches of confidentiality [24]. For this purpose, the data flow diagrams are extended to express the properties of processes or data. By defining policies, the data flow in an architecture can be used to check whether violations of these policies occur [11].

2.3. Uncertainty

Walker et al. define uncertainty as "any deviation from the unachievable ideal of completely deterministic knowledge of the relevant system" [26]. In the following, we always consider uncertainty in the context of a software architecture. In software architectures, uncertainty arises from different or unknown design options in the design of the architecture.

Perez-Palacin and Mirandola define an uncertainty taxonomy that can be used to model software architectures [17]. They divide uncertainty into three dimensions:

- *Location*
- *Level*
- *Nature*

Location describes the place where uncertainty occurs. In the software architecture, this can be, for example, a specific area such as a database that is to be used in an as yet unknown location. The *Level* describes how precisely the uncertainty can be expressed. For that, Perez-Palacin and Mirandola introduce orders into which the level of uncertainty can be classified. They range from a lack of knowledge to uncertainty about the orders of uncertainty. The *Nature* of the uncertainty describes whether uncertainty is because there is not enough data to fill in the missing knowledge or whether there is such great variability that uncertainty persists. The former is called *epistemic* and the latter *aleatory*.

Uncertainty in software architectures can take various forms. Uncertainty can arise, for example, between elements of the architecture - e.g. in the choice of the path from component to component - or concern properties of an element, such as the location of a server. However, uncertainty can also occur in such a way that it cannot be described at all since the subject of the uncertainty is not yet known.

For this thesis, we concentrate on a special form of uncertainty. It is expressed by listing a finite number of options, which is given as an example. These options represent elements in the architecture such as components or properties. By being able to choose between different options, the architecture is influenced in different ways. The description of the uncertainty through elements of the architecture is therefore an uncertain impact.

3. State of the Art

In this chapter, we discuss the current state of the art. In Section 3.1 we give an overview of different approaches to interpreting analysis results of confidentiality analyses. The impact of uncertainty is discussed in Section 3.2. Finally, we discuss confidentiality analyses in Section 3.3.

3.1. Interpretation of Results

In current research on uncertainty in software architectures, uncertainty is often analyzed with detailed inputs in the uncertainty model. For example, values such as the probability that a particular component is used to analyze uncertainty [9]. Determining such values requires a great deal of knowledge and forethought about the system being built. The larger and more complex software systems become, the more difficult it is to estimate the input values. Risk analyses are then carried out, for example, on the basis of these probability values [19]. In the approach of this bachelor thesis, however, it should be possible to make evaluations at runtime that do not require any further knowledge about the probability of using a component.

Walter et al. extend a model evaluation process with the property confidentiality [27]. PerOpteryx is used for this. "PerOpteryx is an optimization framework to improve component-based software architectures based on model-based quality prediction techniques" [18]. A software architecture with uncertainty is analyzed using PerOpteryx. Scenarios of the model that violate confidentiality constraints are filtered out and evaluated as unusable. Unlike in this approach, scenarios that contain violations should not necessarily be filtered. It should be possible to differentiate between these scenarios in terms of the severity of the violations.

Hinton and Lee present a study of the compatibility of policies [12]. A policy is a statement of the goals for the behavior of a system. Policies are compatible if each will not interfere with the other's goals. Each component will receive and produce events, represented by an event sequence to represent what is "knowable". Events are treated as propositions, meaning they are true or false. If an event occurs, then a corresponding proposition is true. If an event does not occur then the corresponding proposition will be false. If there is at least one possible sequence of events that violates a property then the property is not valid for the given system. In the case of this bachelor thesis, policies can be viewed as the set of confidentiality constraints. Event sequences would be the scenarios of the uncertainty model and events would be the data flows through the components. Hinton and Lee provide theorems and axioms for evaluating whether a policy is compatible. The logical expressions contained therein could be used or expanded in the bachelor thesis.

3.2. Impact of Uncertainty

In their master's thesis, Benkler presents an approach that analyses and assesses the impact of uncertainties on confidentiality at design time [2]. For this purpose, the relationship between architectural design decisions and uncertainty is analyzed. It examines how uncertainty can be represented and how the distribution of uncertainty affects the components. Furthermore, uncertainties are classified. Benkler's approach differs from this bachelor thesis in that no analysis of the impact of uncertainty is made. Software architectures will not be evaluated directly based on their uncertainty, but scenarios of the uncertainty model are analyzed.

Boltz et al. deal with the characterization of uncertainties in Access Control [5]. It is examined how existing characteristics of uncertainty taxonomies can be applied in a software architecture. The goal was to find out what types of uncertainty exist in access control and how access control analyses can deal with them. In this thesis, we deal with uncertainty in the form of expressing it through variation modeling. We then generate scenarios and carry out analyses of the scenarios. By bringing together the assessments of the scenarios, a statement is made about the uncertainty of the architecture in terms of confidentiality.

3.3. Confidentiality Analyses

Confidentiality "is the property, that information is not made available or disclosed to unauthorized individuals, entities, or processes" [13]. The GDPR [25] provides an example of confidential information [25]. The processing of personal data such as location, genetic or biometric data of a person is generally not permitted unless certain conditions have been met. Unless those conditions for processing are met, this information must not be available to unauthorized individuals, entities or processes.

The later violations of confidentiality are determined, the more expensive the solutions become [4]. Therefore, it makes sense to consider confidentiality early on and to model it in the design of a system.

Approaches like Privacy by Design (PbD) include confidentiality in the design phase of a system architecture [22]. This reduces costly updates. Seifermann et al. present one way how this can be implemented in system architectures [23]. Here, Data Flow Diagrams (DFDs) are used to model confidentiality in system architectures. The modeling is done in the PCM and is then translated into a logical program in Prolog. This makes it possible to check general confidentiality constraints in a system architecture. In the following, we understand a confidentiality-preserving architecture as one that has no violations against confidentiality requirements. Non-confidentiality-preserving architectures are those that have a violation of confidentiality policies. We note that previous approaches do not include uncertainty in the analysis. Our approach enables an analysis of confidentiality under architectural uncertainty.

4. Running Example

To illustrate the theoretical contents of this thesis, we use a motivating example of a software architecture. Most of the model and the basis comes from Hahner [10]. We adjust some elements of the model to cover as many aspects of our concept in Chapter 5 as possible. The software architecture represents a simplified version of an online shop. It also includes uncertainty impacts at five different points. The uncertainty impacts arise from the fact that in the presented state of the architecture no decision has yet been made as to which variant of the implementation will be chosen. However, we can already model the different variants because we assume that we are familiar with all variants. Each uncertainty impact is marked with a question mark in the following figures.

In our example, we only deal with the use case seen in Figure 4.1. This is an excerpt from the ordering process of an online shop. In this excerpt, the user enters a delivery address to which a product should be delivered. The user can either enter their personal home address or, for example, an anonymous pick-up station. When sending to a personal address, personal data is transferred. When sending to a non-personal address, no personal data is accordingly transferred. In this case, the uncertainty is not necessarily caused by design decisions that have not yet been made. One simply does not know whether the user specifies a private address or not. So this factor has to be taken into account because user behavior influences whether breaches of confidentiality occur.

In the diagram, the distinction is marked with a question mark between the arrows. In the following, we call this distinction the **sensitivity of the data** and it is the first uncertainty impact in the system.

In Figure 4.2, we show the component diagram of the system. We have a frontend that passes data to the backend and handles user interactions. The data is processed in the backend and then passed on to two places. The `BackupService` backs up the same data as the `DatabaseService`. As we use a component-based architecture, we can use different components if they implement the same interface. In the case of our database service, we add two options to choose between. The backend either uses `Database A` or `Database B`. Such an ADD occurs e.g. because the design process of the architecture is in a more large-grained state [16]. This means that in the current state of the architecture it is still uncertain which option will be chosen in the later implementation. However, we assume that there exists an agreement that one of these options will be implemented. This allows us to model them as deterministic variants, with the user entering either personal information or non-personal information.

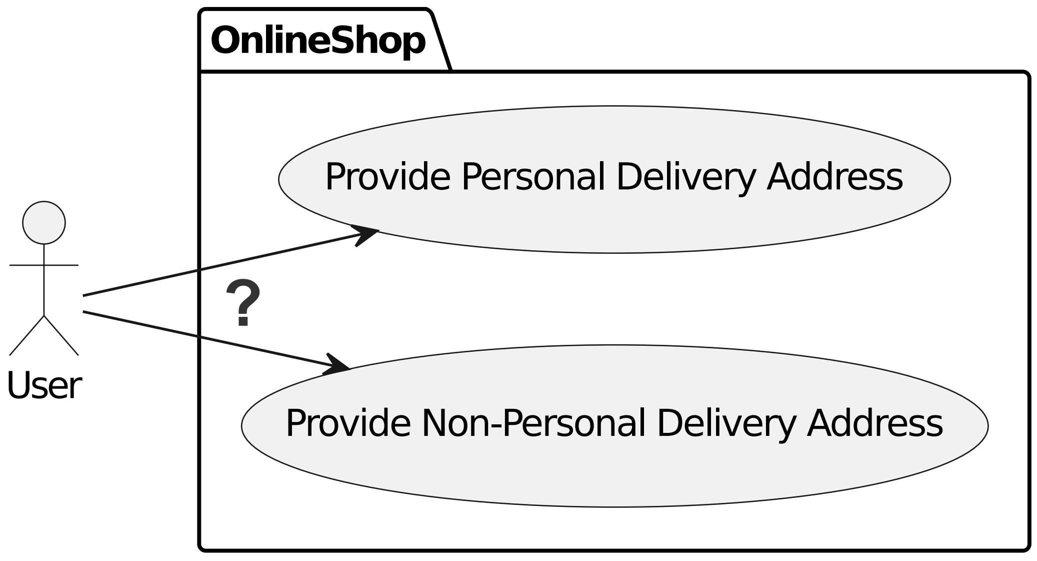

OnlineShop
Provide Personal Delivery Address
?
User
Provide Non-Personal Delivery Address

Figure 4.1.: OnlineShop use case diagram under uncertainty in the input of data

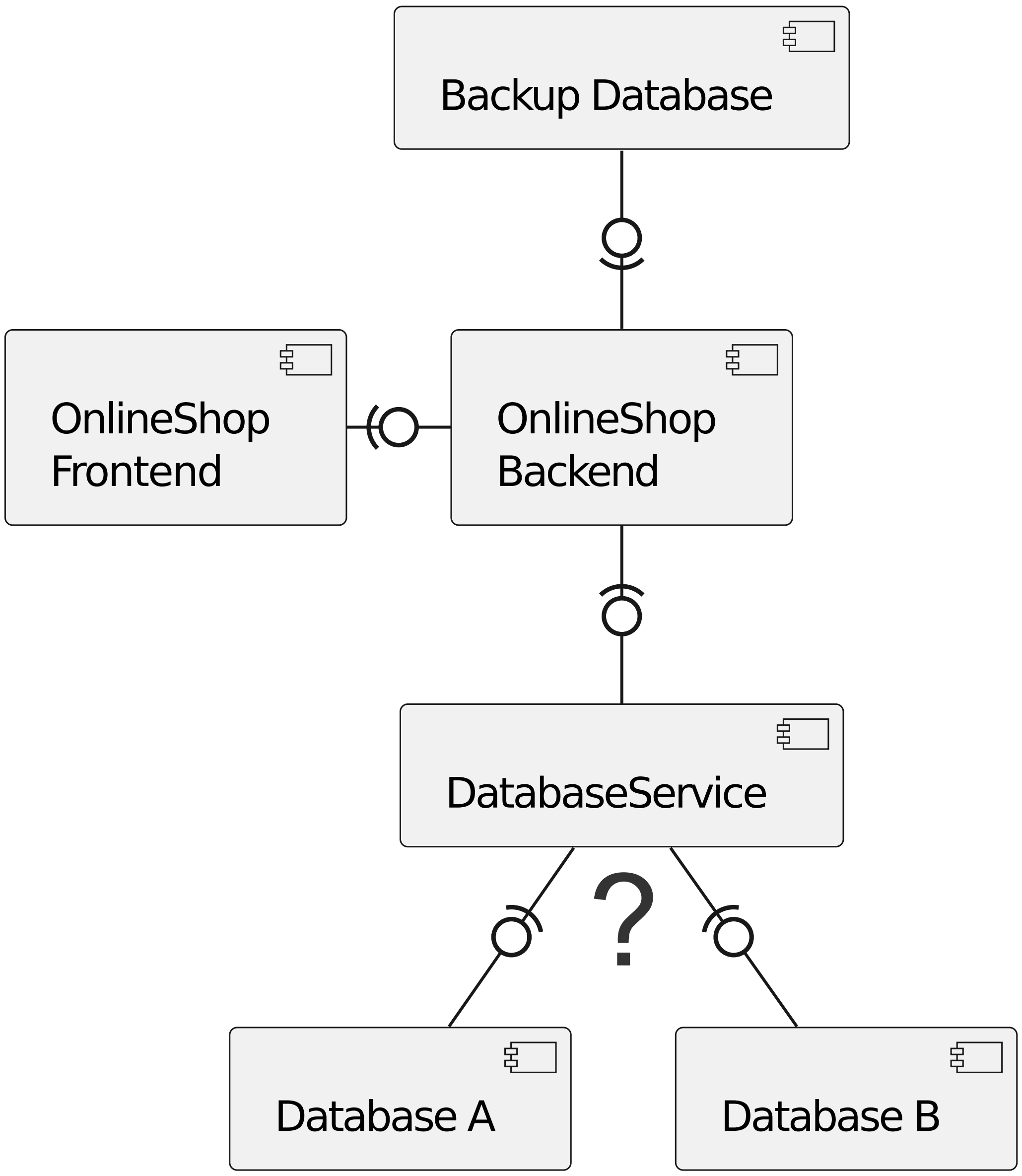


Figure 4.2.: OnlineShop component diagram showing uncertainty in the choice of the database

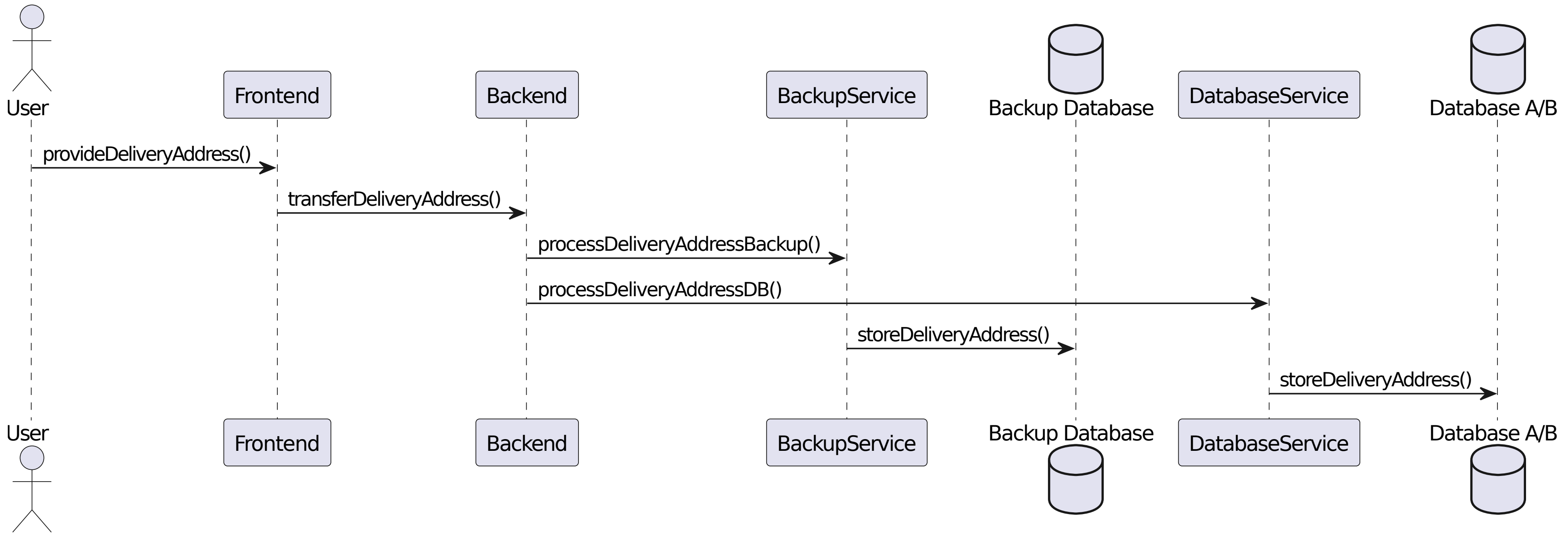


Figure 4.3.: OnlineShop sequence diagram

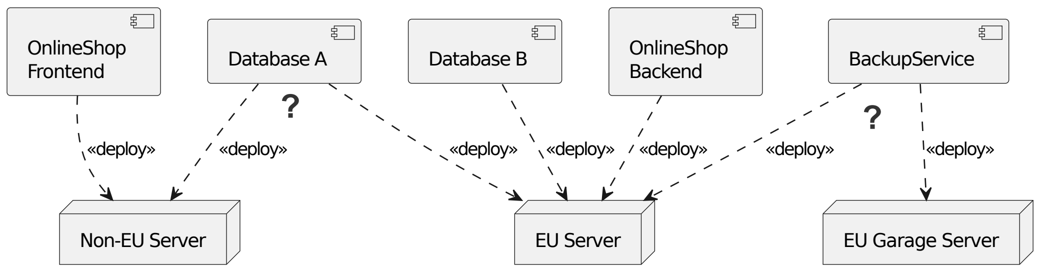


Figure 4.4.: OnlineShop deployment diagram showing uncertainty in database deployment location

The procedure for transferring the delivery address through the system can be seen in Figure 4.3. The user enters the delivery address in the frontend. The address is then transferred to the backend. From there, the same data is sent to the BackupService and the DatabaseService. The BackupService saves the data in the Backup Database. The DatabaseService stores the data in Database A or B.

In Figure 4.4 we see the deployment diagram of the architecture. The frontend of the online shop is deployed on a server that is not in the EU. Database B and the backend are always on a server in the EU. We see two more uncertainty impacts marked by the question marks. Database A can either be deployed in the EU or not in the EU. The BackupService is always in the EU, either on the EU Server or on the EU Garage Server.

The Backend is responsible for the **encryption of the data**. The data is processed in two ways, as seen in Figure 4.5. For the backup service, they are always encrypted before they are transferred. For the DatabaseService, it can be decided in the design whether the data should be encrypted or not. Choosing to encrypt the data in the database would increase safety in case of an attack on the database. However, encrypting data would increase the complexity of the implementation. We again assume that the final decision on this uncertainty has not been done yet. Therefore, we model the options as another uncertainty impact.

In Table 4.1, we see a list of the uncertainties that occur in the OnlineShop. There are a total of five different places where uncertainty occurs. Each uncertainty is named in the "Uncertainty" column according to what the uncertainty is about. In the "Varying Subjects" column we see the list of architectural elements that can be used to describe the uncertainty.

We define constraints to be followed in the architecture. In order to make any data safe from attacks on the database, data stored in a database should always be encrypted. In addition, we adhere to the EU directive that the personal data of EU citizens may only be stored in the EU. In Figure 4.6 we see a table listing which combination of properties lead to violations. For better representation, we see two tables instead of a four-dimensional representation. The table on the left shows the sensitivity of the data (left side) compared to the deployment location of the database (top right). Only if the user's personal data is stored on a database that is not in the EU is there a violation. In the table on the right, the architecture element (left side) is compared with the encryption of the data (top right). If the architectural element under consideration is a database and the data is not encrypted, a violation occurs.

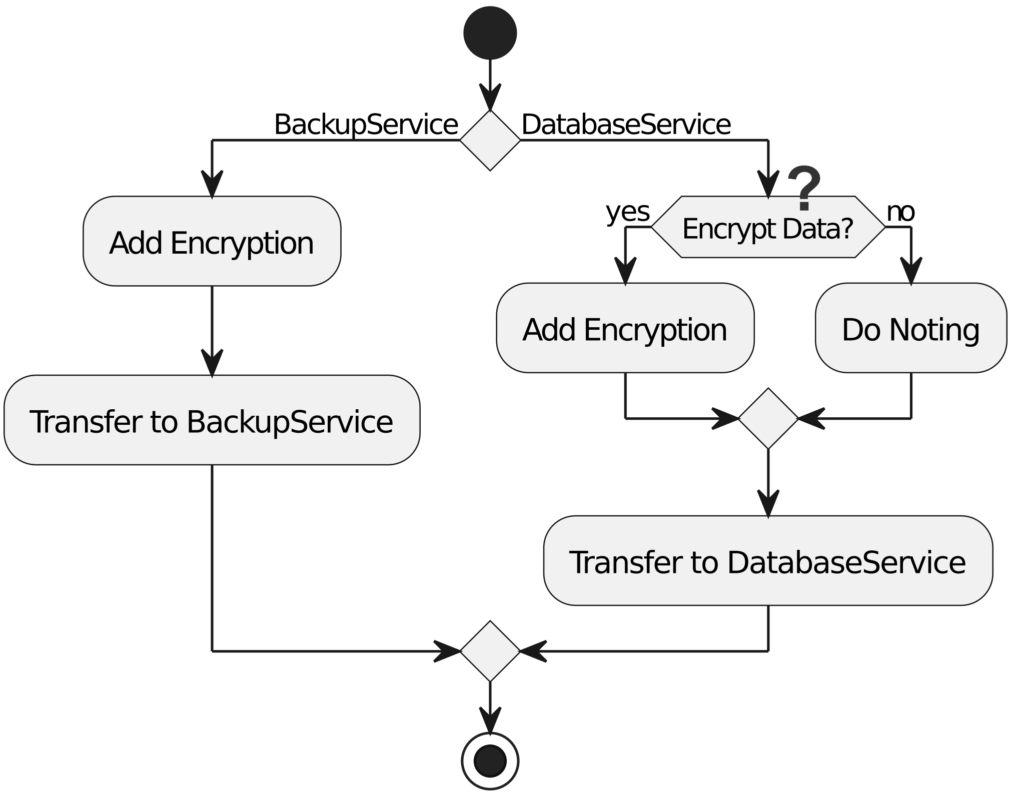
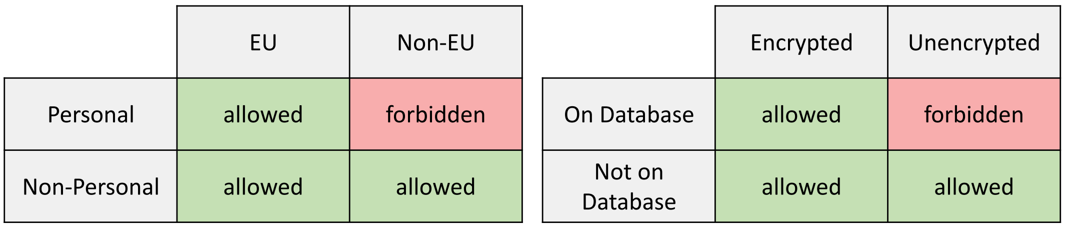


Figure 4.5.: Backend activity diagram under uncertainty of encryption choice

#	Uncertainty	Varying Subjects
1	User Input Sensitivity	[Personal, Non-Personal]
2	Encryption Choice	[Encrypted, Unencrypted]
3	Database Choice	[Database A, Database B]
4	Deployment Location Backup Database	[EU Garage Server, EU Cloud Server]
5	Deployment Location Database A	[EU Cloud Server, Non-EU Server]

Table 4.1.: Occuring uncertainties in the OnlineShop described with varying subjects



	EU	Non-EU
Personal	allowed	forbidden
Non-Personal	allowed	allowed

	Encrypted	Unencrypted
On Database	allowed	forbidden
Not on Database	allowed	allowed

Figure 4.6.: Table of allowed and forbidden configurations of the architecture

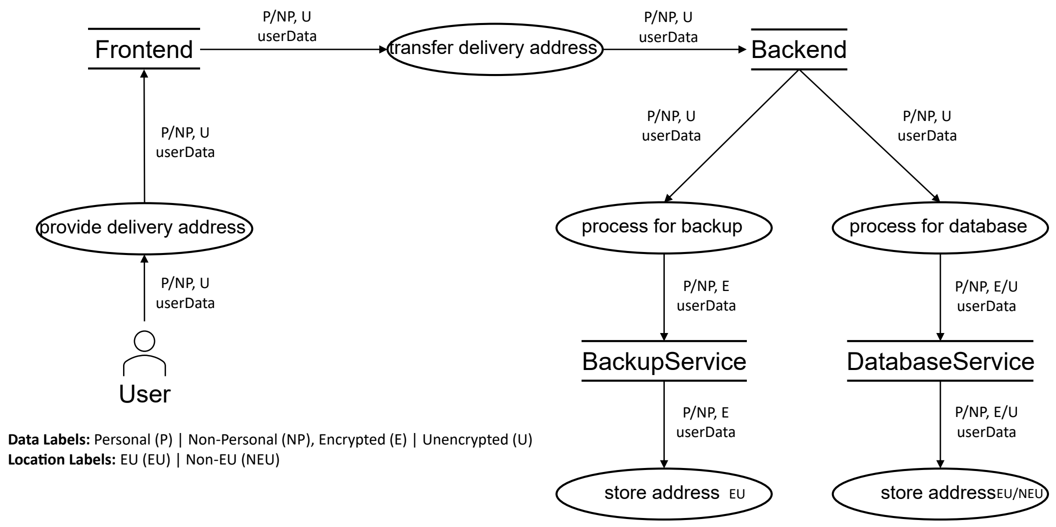


Figure 4.7.: System data flow diagram under uncertainty represented by options of labels

In Figure 4.7 we see the possible data flow through the system. The user's delivery address entered is transferred as *userData* through the various components. In doing so, *userData* can adopt the properties that are listed as **Data Labels** in the legend at the bottom left. These labels determine whether the data is personal or non-personal, encrypted or unencrypted. We also assign **Location Labels** for storing the shipping address. This can be either in the EU or non-EU.

5. Uncertainty-aware Confidentiality Analysis

In this chapter we explain the concept of our approach. Section 5.1 gives an overview of what the goal of the concept is and the combination algorithm is presented fundamentally. Section 5.2 explains the whole data structure that we use for the combination algorithm. In Section 5.3 we show how we reuse existing approaches to make them suitable for use under uncertainty. Then we explain in Section 5.4 how we can use our approach to make more precise statements about the causes of violations of confidentiality than the previous approaches in Section 5.3. All theoretical explanations are illustrated with our running example from Chapter 4.

5.1. Overview

The aim of the concept is to be able to gain more information about violations against confidentiality than the scenario recetion in Peropteryx [27] has already been able to do up to now. More information means that we not only gain knowledge about where and why violations occur but can also filter uncertainty impacts that cause violations in a scenario. We first define the **naive approach** with which an architecture can be analyzed for confidentiality. An entire architecture is rejected as soon as a scenario contains a violation. Compared to the scenario rejection, individual scenarios of the architecture that contain violations of confidentiality are filtered. Instead of rejecting and no longer using scenarios with violations, we want to use information about the violations to enable the analysis of uncertainty impacts.

We define a combination algorithm that uses the information about violations for the interpretation of the analysis results of confidentiality analyses. In the combination algorithm, information about the architecture and violations of confidentiality are used as input parameters to systematically extract inferences from the input. The combination algorithm is built up incrementally. I.e. that it consists of several stages that build on one another. Each stage can extend the algorithm with new input or output parameters to provide a more precise analysis. We start building the algorithm with pre-existing result interpretations from Chapter 3, which are described in more detail below in Section 5.3.

5.2. Required Model for the Approach

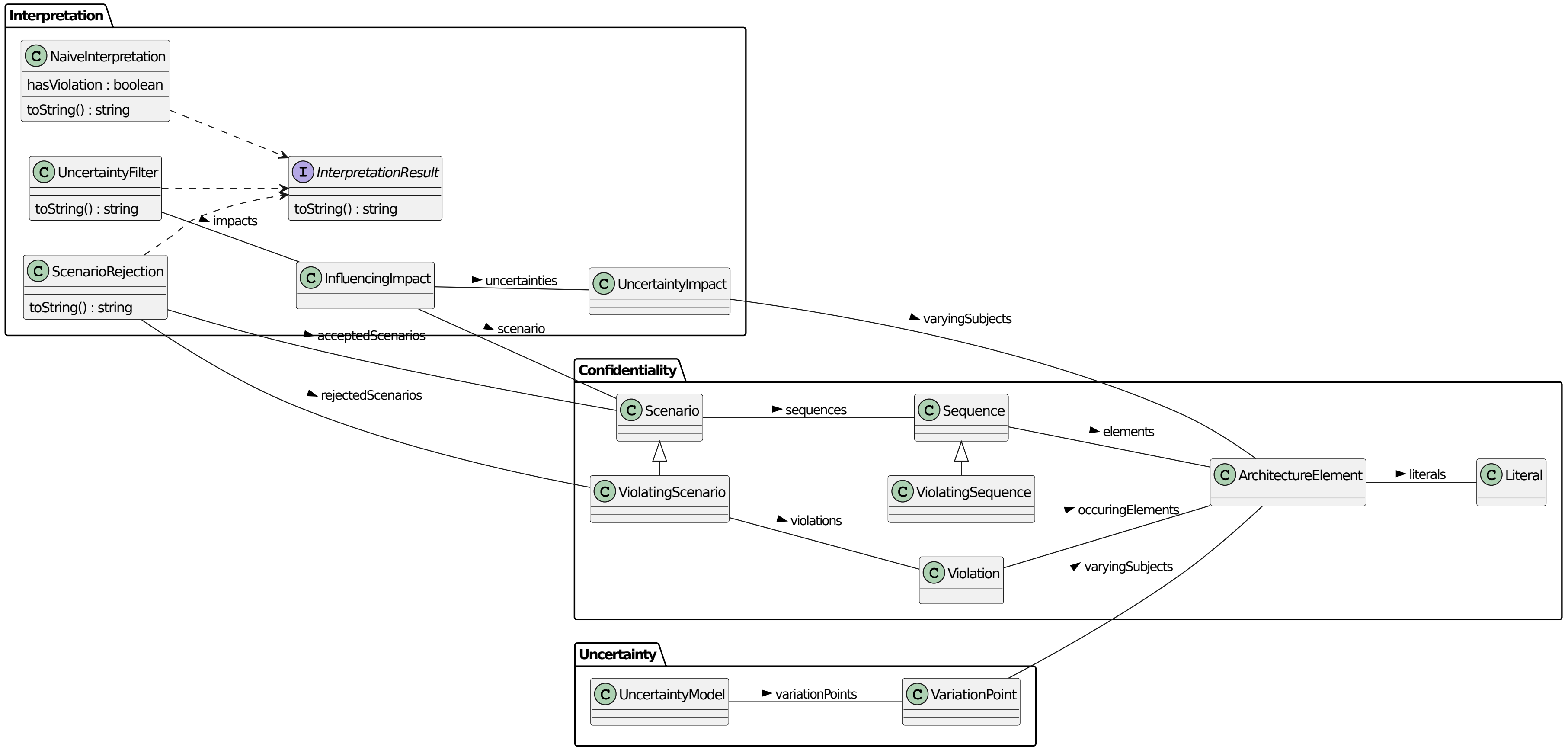


Figure 5.1.: Class diagram of the model used for the combination algorithm

In this section, we describe the model used in the combination algorithm, which can be seen in Figure 5.1. Attributes like identifier or name are intentionally omitted and the focus is on how the classes work. We start with the explanation that determines the format of the confidentiality analysis result. These are in the package Confidentiality. At the top of the package is the Scenario class. Because we express uncertainty in terms of variations, a scenario emerges from choosing a variation at each uncertainty impact. The Scenario class, therefore, contains the analysis results of a scenario. The ViolatingSequence class inherits from Scenario and represents the scenario in which a violation occurs within the scenario. It extends the Scenario class in that it owns the Violations that trigger this violation. Each Violation contains the ArchitectureElements where the violation occurs. The ArchitectureElement class contains information about an element in the architecture. These can be different types of elements, such as a method call, an allocation element, a repository element or a procedure, such as storing data. For the concept presented here, we summarize all these elements in one class, since the type of element has no influence on the interpretation. A ArchitectureElement can contain literals that describe further properties of the ArchitectureElement. One feature could be the location. The literal then contains the location (e.g. EU or Non-EU). Each scenario has sequences that represent the flow of data through the architecture. A Sequence consists of the ArchitectureElements sorted in the order in which they are called. The class modelViolatingSequence is a specialized Sequence in that it is a sequence that contains a confidentiality violation. The exact use of this is explained in Section 5.4. From the package Confidentiality we get the analysis results in the format we need for the analysis in our approach.

The ResultInterpretation interface is on the top of the Interpretation package and only contains the toString() method, because the result interpretation of our approach should be output in human-readable form. At each stage of our combination algorithm, we use a separate class that summarizes our result. These each implement the interface ResultInterpretation. On the left, we have the NaiveInterpretation. It is distinguished by the fact that it only has a boolean in which it is recorded whether the architecture contains violations or not. On the right, we see the ScenarioRejection. It has two lists, acceptedScenarios and rejectedScenarios. In the middle is the UncertaintyImpact class. This contains a list of InfluencingImpact. The InfluencingImpact class connects UncertaintyImpacts with a Scenario. Thus we get a model in interpretation that reassembles and links the existing results of confidentiality analyses. From this, our concept reads the interpretation of confidentiality under uncertainty.

The remaining package Uncertainty contains classes needed to represent uncertainty in the architecture. The class UncertaintyModel contains the uncertainty in the architecture which is defined as variations in the variation model. These variations are modeled by the class VariationPoint and contain the varying subjects of an uncertainty impact. The subjects are elements of the architecture, hence the connection to ArchitectureElement is made.

5.3. Reusing Existing Approaches for Confidentiality Analyses

As described in Chapter 3, previous confidentiality analyses provide skewed results or ignore uncertainty entirely as soon as they account for it. In this section, we show how we reuse the named approaches to deal with uncertainty. At the same time, they form the basis for the combination algorithm.

The first part is the naive approach in Algorithm 1, which rejects the entire software architecture as soon as at least one violation occurs in any scenario. We implement this first stage by receiving a list of ViolatingScenarios. The output is an interpretation of the type NaiveInterpretation. If the length of the list is exactly zero, then we know that there is no violation in the architecture and we set hasViolation to false. If the length of the list is greater than zero, the result is that the architecture contains at least one violation and should therefore not be used. We set the boolean hasViolation respectively to true. Using hasViolation we get the information on whether the architecture is usable or not. We find that this analysis is an overestimate. Just because e.g. a small part of the architecture causes a violation it does not mean that the whole concept of the architecture is unusable. This poses a problem for software architects because they receive no further information other than that a violation has occurred. The effort to find this violation and its causes still needs to be fully accomplished.

Algorithm 1: Naive Approach

```
Data: ViolatingScenario[] violations
Result: NaiveInterpretation interpretation
if $violations.length == 0$ then
    interpretation.hasViolation ← false;
else
    interpretation.hasViolation ← true;
end
return interpretation
```

Demonstration

We use our example from Chapter 4 to demonstrate the Algorithm 2. We get all ViolatingScenarios as input from the sensitivity analysis. This corresponds to a list of 17 ViolationScenarios. Since the length of the list is greater than 0, the result is an object NaiveInterpretation with the attribute hasViolation = true. The software architect receives as a result: The architecture of the online shop should not be used, since violations of confidentiality occur in it.

The next step to improve the naive approach is to filter out individual scenarios that involve violations of confidentiality. We call this analysis scenario rejection because individual scenarios are rejected. We see the implementation of the analysis in Algorithm 2. In this, we receive the analysis of the scenarios as an input parameter. As an output we get the interpretation ScenarioRejection. It contains two lists: rejected scenarios and accepted scenarios. If a scenario is a violation, we add the scenario to the rejected scenarios. Otherwise, we add it to the accepted scenarios. A software architect now learns more from the returned interpretation than in the naive approach in Algorithm 1. From the accepted scenarios they learn which architecture variants do not cause a violation. Inside rejectedScenarios all rejected scenarios are the scenarios that lead to violations. The ViolatingScenarios contains information about the location and reason for the violation. This is still a hurdle for the architect. Just from the information about which final architecture scenarios have violations or not, it is not possible to deduce which uncertainty impacts need to be examined more closely. This still has to be done manually by examining all the sequences in the scenarios.

Algorithm 2: Scenario Rejection

```
Data: Scenario[] scenarios
Result: ScenarioRejection interpretation
for scenario : scenarios do
    if scenario instanceof ViolatingScenario then
        interpretation.addRejected(scenario);
    else
        interpretation.addAccepted(scenario);
    end
end
return interpretation;
```

Demonstration

For the demonstration of the Algorithm 2, we use two selected scenarios from the 32 available. These fully represent how scenario rejection works, because we have one scenario that contains a violation and one that does not. We receive the scenarios as input, which can be seen in Figure 5.2 and Figure 5.3. Both show a simplified data flow of the input user address which is represented by the arrows. We define the sensitivity at the first input of the data. In the first Scenario we have no violation of confidentiality because the personal data is stored within the EU and the data is encrypted on all databases. Therefore this Scenario is not a ViolatingScenario. This will add it to the list of accepted scenarios. The second Scenario contains a violation, which is marked in red. Here data is stored unencrypted on a database. Therefore this Scenario is a ViolatingScenario. Thus it is added to the list of rejected scenarios.

5.4. Uncertainty Impact Filter

The uncertainty impact filter is the next extension of the combination algorithm. The aim of this extension is to create a link between the violations and the uncertainty defined as variations in the variation model. For this purpose, it is checked which uncertainty impacts are relevant for a sequence that leads to a violation. We make an important precondition before explaining the algorithm for it. In Algorithm 1 and Algorithm 2, it was not relevant what the sequences should be structured like since only occurrences of violations were necessary to generate the result interpretation. Now it is important that data flows are divided into sequences that contain violations and do not contain violations. From the point at which violation occurs in a data flow, all preceding elements are part of the violated sequence. If there are branches in the data flow that occur before the violation, the data flow at the branch is again a non-violation sequence. This also applies to the subsequent data flow directly after the violation. We see an example of translating a simplified data flow from Chapter 4 to Figure 5.4. The violation of confidentiality arises at the point where the warning sign is placed. Here unencrypted data is stored in the database, which violates the policy. The "Encrypt Data for Database" branch does not

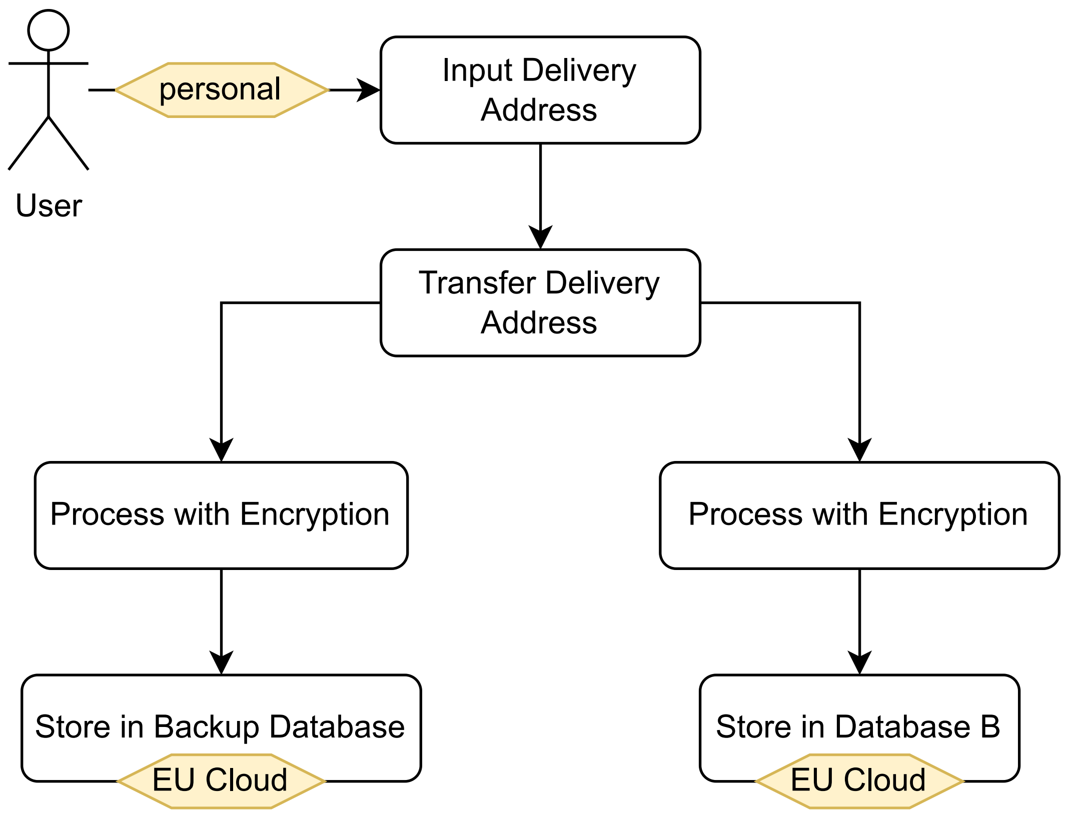


Figure 5.2.: Example data flow from our running example without violation

contain any violation, so this part is not marked. From this, we get two sequences from one data flow. The sequence with a violation is circled in red, and the sequence without a violation is circled in blue. Both sequences now belong to one scenario.

To now make a connection to the uncertainty, the uncertainty model that contains the varying subjects is required as a new input parameter. We continue to receive all scenarios. As seen in Algorithm 3, we start iterating over each scenario. For every scenario, we collect information about the uncertainty impacts and their effects of them on the existing violations in the sequences of the scenario. For this, we create a new `InfluencingImpact` and set its scenario to the one in the current iteration. Now we iterate over each sequence of the scenario. If a sequence contains no violations, we cannot find any uncertainty impacts responsible for a violation, so we skip this case. If a sequence containing violations is an instance of `ViolatingSequence`, we check for each element in the sequence whether it is a varying subject of an uncertainty impact. If this is the case, the uncertainty impact is added to the uncertainties of the `InfluencingImpact`. Each `InfluencingImpact` is added to the interpretation after the scenario has been processed. This enables us to limit the number of uncertainty impacts to a set that is influential for the violation in a scenario. Any impacts that are not in this set are unrelated to the violations since no elements varied by the impact occur in the sequences leading up to the violation.

A software architect now learns which uncertainty impacts potentially various elements that lead to a violation. They also know which impacts are definitely not responsible

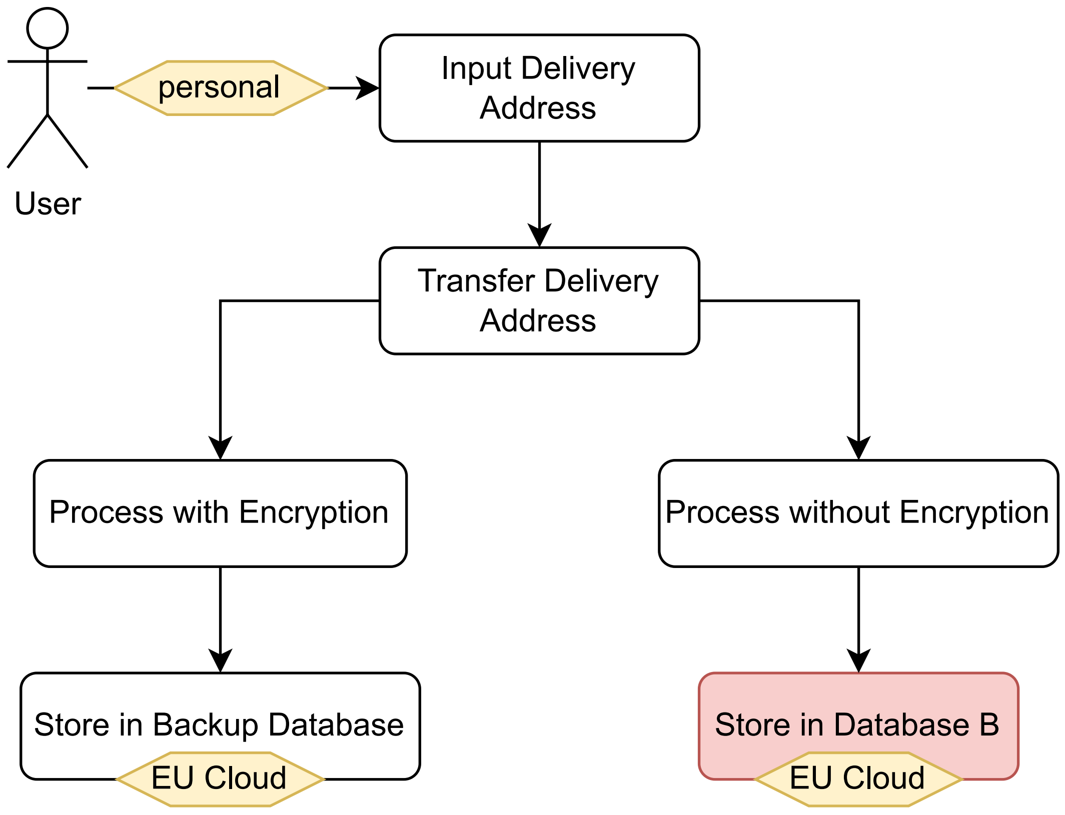


Figure 5.3.: Example data flow from our running example with a violation

for a violation. Therefore, they can make more targeted changes in the architecture based on our analysis.

There is another hurdle here because it is not yet clear at this point whether uncertainty impacts are identified as a potential cause, even though they do not contribute to the violation. This is further treated as future work in Section 7.2.

Demonstration

To demonstrate the Algorithm 3, we use one selected violation scenario from the 32 available running examples from Chapter 4. This scenario can be seen in Figure 5.5. The uncertainty impacts are marked by the question marks. The uncertainty impact that decides whether the Database A is deployed in the EU or not in the EU is missing. The reason for this is that Database B is used in this scenario. We get this scenario together with the UncertaintyModel from Chapter 4. We create a new InfluencingImpact and set the scenario attribute to the received scenario. There is a violation of confidentiality because data is stored in unencrypted form on Database B. So we have a Sequence that contains a violation (marked red) and one that does not contain a violation (marked green). Since only the red Sequence is of type ViolatingSequence, we don't iterate over the elements of the green sequence. For each element of the red Sequence, we now check whether the element occurs in the varying subjects of a VariationPoint. This is the case for the literal personal, Database B and Process with Encryption. Thereby we add the uncertainty impacts marked with 1, 2 and 3 to the InfluencingImpact.

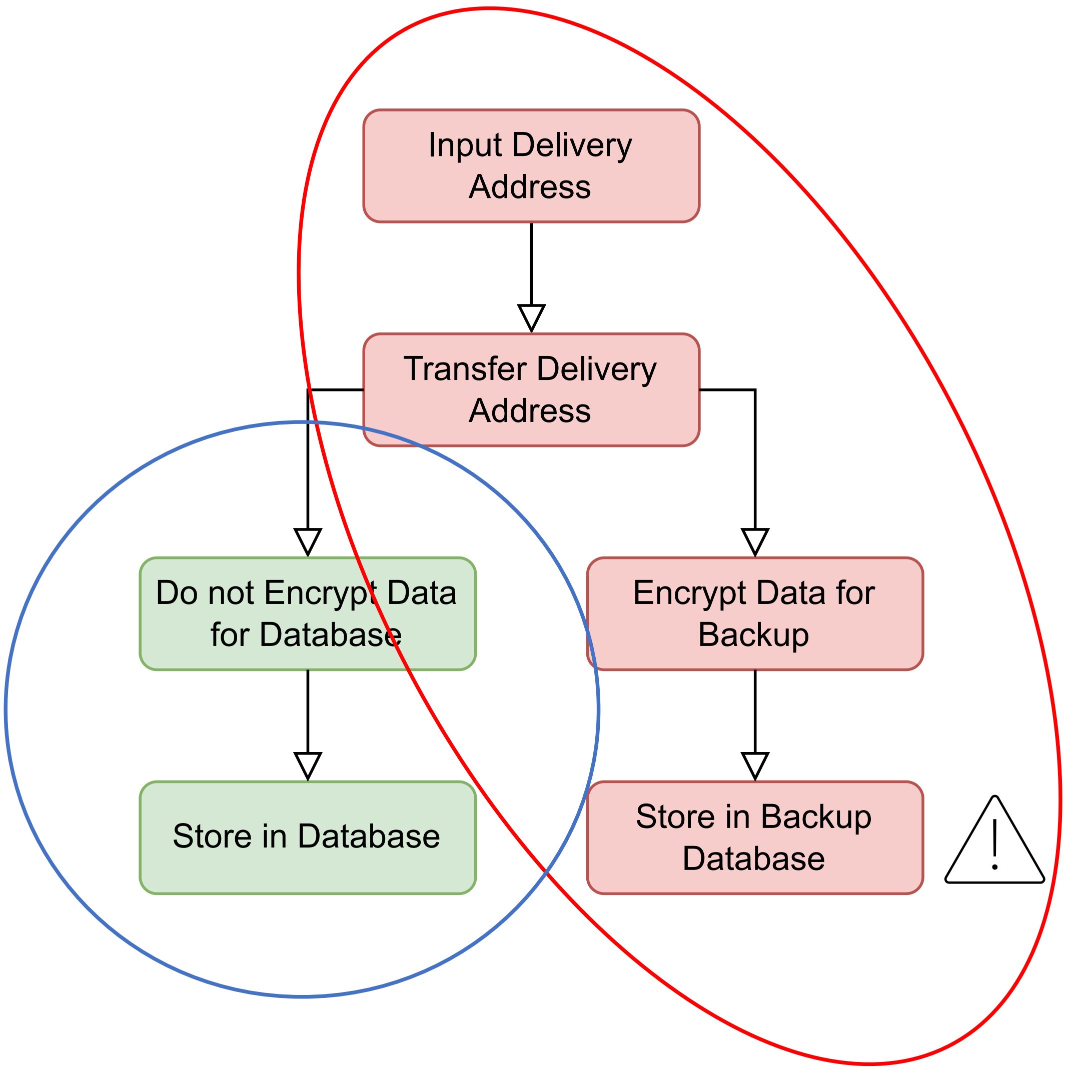


Figure 5.4.: Example of splitting data flows into sequences

Algorithm 3: Uncertainty Impact Filter

```
Data: Scenario[] scenarios, UncertaintyModel uncertainties
Result: Interpretation interpretation
for scenario : scenarios do
    i ← new InfluencingImpact;
    i.scenario ← scenario;
    for sequence : scenario.sequences do
        if sequence instanceof ViolatingSequence then
            for element : violation.sequence do
                for impact : uncertainties do
                    if impact.varyingSubjects.contains(element) then
                        i.uncertainties.add(impact);
                    end
                end
            end
        end
    end
    interpretation.impacts.add(i);
end
return interpretation;
```

A software architect now knows which uncertainty impacts are potentially responsible for the violation in this scenario. Furthermore, you know that the choice of the location of the deployment of database A and the backup database is not responsible.

This procedure can be continued analogously for each scenario. Scenarios that do not contain any violation produce an InfluencingImpact with an empty list uncertainties. No included Sequence of the Scenario is then of the type ViolatingScenario.

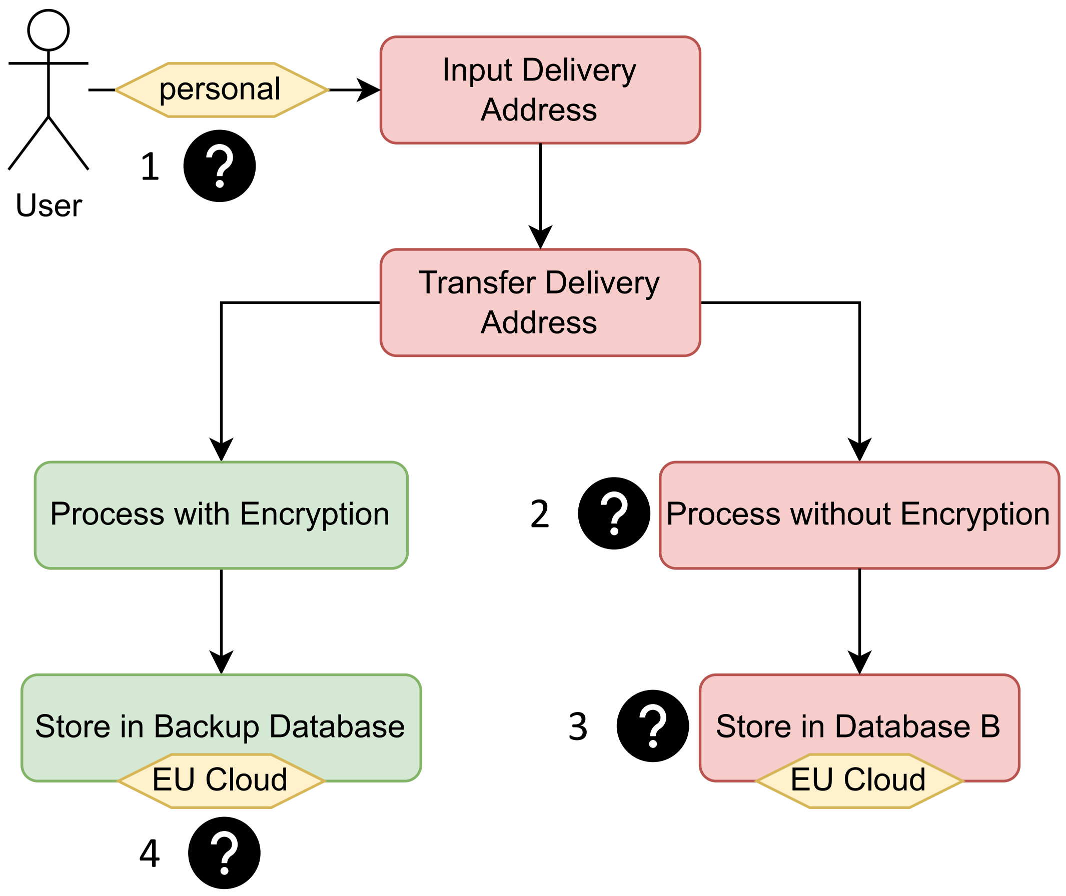


Figure 5.5.: Example data flow from our running example with a violation and annotated uncertainty impacts

6. Evaluation

In this section, we discuss the results of our evaluation. First, we provide our evaluation plan containing the goals and questions to assess these goals in Section 6.1. For this we use the goal-question-metric approach as presented by Basili, Caldiera, and Rombach [1]. In Section 6.2 we explain the design we use to answer the questions from Section 6.1. Afterward, the results for our respective goals from Section 6.1 in Subsection 6.3.1, 6.3.2 and 6.3.3 are presented. Finally, in Section 6.4 and 6.5, we discuss the threats to validity and limitations that we assume for our approach.

6.1. Evaluation Plan

In this section, we explain the goals with which we evaluate the combination algorithm as presented in Chapter 5. To this end, we have defined the following goals:

G1 The interpretation on a non-binary scale that the combination algorithm gives is **feasible**. This means we want to find out whether the approach in Chapter 5 is fundamentally possible to execute.

G2 The results of the combination algorithm are **accurate**.

G3 The combination algorithm can be **used** by a software architect.

With the planned approach, we want to interpret the results of the confidentiality analysis in more detail than previous approaches. The naive approach in Chapter 3 interprets results on a binary scale. The possible interpretations in this scale are either "reject" or "accept". The scenario rejection also interprets the results of the confidentiality analysis on a binary scale. This then takes place at the level of the scenarios of an architecture. Here, too, scenarios are either accepted or rejected. Our approach in Section 5.4 has the task of not rejecting architectures that contain violations in scenarios but considering them individually. This results in individual interpretations for each scenario of an architecture. These are then combined into an overall interpretation of the architecture. With goal **G1** we want to evaluate the feasibility of our approach to deliver results in a non-binary way. In order to carry out an interpretation with the naive approach, a software architect needs the confidentiality constraints that are to be observed in the architecture. Furthermore, the architecture itself is a necessary input parameter. To do this, we ask the following question:

Q1.1 *Is it possible to analyze a software architecture that contains architectural uncertainty on a more differentiated scale than a binary scale?*

We want to determine whether the approach presented in this thesis achieves the tasks for which it is intended. For this, we set the goal **G2**. Since the combination algorithm is divided into several stages, we need to examine each stage for accuracy. This means that we want to examine how good the three implementations from Chapter 5 are at delivering the relevant results and how many of the relevant results are found. Relevant results are the proportion of interpretation results that we expect to be rated positively e.g. without violation. In the case of the naive interpretation in Section 5.3, an architecture should be classified as "rejected" if it contains at least one violation in a scenario. It is classified as "accepted" if there are no violations in the scenarios. For the naive approach, we provide **Q2.1**. The next step of the combination algorithm is scenario rejection (Section 5.3). In this case, the individual scenarios are checked for violating sequences (**Q2.2**). Scenarios should be rejected if at least one violation occurs. The scenario is accepted if there is no violation. In Section 5.4 Uncertainty Impacts that potentially contribute to a violation in a scenario are filtered out. This gives a software architect more precise feedback on maintaining confidentiality in the software architecture. In order for this step of the combination algorithm to be used, the set of uncertainty impacts potentially leading to a violation must match the expected set **Q2.3**.

Q2.1 *Does our approach at least has the same accuracy as the naive approach?*

Q2.2 *Does our approach at least has the same accuracy as the scenario rejection?*

Q2.3 *What is the accuracy with which our approach identifies the influencing uncertainty impacts?*

With goal **G3** we want to evaluate the usability of the approach. The aim of our approach from Chapter 5 is to give a software architect conclusions about which uncertainty impacts are responsible for violations of confidentiality. It generates the connection between violations of confidentiality and architectural uncertainty. By examining the approach for usability, we check whether our approach makes this connection at all. To evaluate, whether it is possible to use our approach, we identify the required information a software architect has to provide for the combination algorithm (**Q3.1**, **Q3.2**). Furthermore, we evaluate whether a software architect receives the information about the influence of uncertainty impacts on violations from our results of the approach (**Q3.3**). We also check whether a software architect receives this information faster than is the case with the current state of the art. If that is the case, then our approach is effectively capable of being used to analyze architectures. This not only guarantees the theoretical implementation, but also the practical applicability of our approach. To evaluate goal **G3** we ask the following questions:

Q3.1 *What is the required information to be able to apply the combination algorithm?*

Q3.2 *When can a software architect know the additional information we need for the combination algorithm?*

Q3.3 *Does a software architect know more quickly what causes violations in the architecture to occur than with previous approaches?*

6.2. Design

In this section, we describe how we evaluate the goals from Section 6.1. We also explain how we answer the questions to validate the goals.

6.2.1. Case Studies used for evaluation

In this section, we present the case studies used in the evaluation. We use both individual scenarios of the models and the complete architecture. All models presented come from external sources and thus strengthen our evaluation in which it has fewer weaknesses due to the fact that the models and examples used were constructed by us.

TravelPlanner (M1) The TravelPlanner is an application that users can use to find flight connections for an airline. When booking a flight, the payment is made by credit card. The user gives their credit card information to the airline when booking. He then authorizes the airline to make a booking on their credit card by using a credit card service. The model is a confidentiality model and the credit card information is confidential data. It may only be processed if the user agrees to the circumstances. The model is introduced by Katkalov [14]. Besides, the model is also used in "Data-Driven Software Architecture for Analyzing Confidentiality" by Seifermann, Heinrich, and Reussner [23]. For the evaluation here we use the slight modification by Walter et al. in "Architectural Optimization for Confidentiality under Structural Uncertainty" [27]. An uncertainty impact is added by having a situation where the airline can treat the user's credit card information differently. Either in the way that the user has agreed to or in a way that the user has not consented to. This creates an uncertainty impact, which allows the case study model to be checked by our approach.

DistanceTracker (M2) The DistanceTracker is an app that can be used to record the distance covered while jogging. This model is also a confidentiality model. The user's GPS data is confidential data. The processing of the GPS data should take place exclusively on the user's smartphone and must not reach the tracker app. This model is also from Katkalov [14]. Again we use the modification by Walter et al. in "Architectural Optimization for Confidentiality under Structural Uncertainty" [27]. In this case, the GPS data is processed either on the smartphone or on a DistanceTracker server. By adding this uncertainty impact, our approach can also be carried out here.

The two applications TravelPlanner and DistanceTracker only have one uncertainty impact and overall have little complexity. Nevertheless, they are suitable for evaluation, since fundamental errors are easier to identify with smaller models. The low complexity also helps to understand the model quickly and without tools.

Running Example (M3) We also use our own running example from Chapter 4. The approach in Section 5.4 was developed using this model. Therefore, the running example serves as a confirmation of how the approach works. The model has five uncertainty impacts that result in 32 scenarios.

6.2.2. Goal G1 - Feasibility

With target **G1** we want to check the feasibility of the approach. Specifically, we want to evaluate whether the approach is capable of making a non-binary statement about the confidentiality of a software architecture under uncertainty. We separate feasibility into the following aspects:

F1 The approach can be executed

F2 The approach delivers results

F3 The delivered results are to be classified on a non-binary scale

F4 The approach can be implemented in reality

By examining our approach on these four points, we ensure that it forms a basis that represents an evolution of the state of the art. To answer Question **Q1.1**, we check the points **F1**-**F4** by using exemplary scenarios from the presented model **M1** and refer to our prototypical implementation of the approach [3].

6.2.3. Goal G2 - Accuracy

For goal **G2** we evaluate the accuracy of the approach. The output of the approach is examined for accuracy, i.e. the interpretation results of analysis results from the confidentiality analysis. A high degree of accuracy thus corresponds to a high level of agreement between our results and the expected results.

For all questions **Q2.1, Q2.2** and **Q2.3** we use the models presented in the case studies. For question **Q2.1** we want to find out whether architectures are correctly recognized as architecture with or without violation by implementing the naive approach in Section 5.3. All four models contain at least one scenario that causes a violation of confidentiality. Therefore, for each model, we select a scenario that has no violation. As a result, we have an equal number of cases where the implementation either rejects or accepts the architecture. Question **Q2.2** deals with the individual scenarios of the architectures. By evaluating **Q2.2** we determine whether we can achieve at least the same result as e.g. the existing approach from Liu [15].

For the models **M1** and **M2** we show the scenarios with their data flows. For the model **M3** we choose example scenarios to illustrate the evaluation. We use precision and recall as metrics. For each model, we create a reference output i.e. which scenarios have violations and which do not. This reference output is then compared to the output of the prototypical implementation [3]. Each correctly identified scenario is a true positive TP and each incorrectly identified scenario is a false positive FP. A missing result is considered a false negative FN.

For Question **Q2.3** we also create a reference output i.e. which uncertainty impacts are influencing the violations in a scenario with violations. This means that we only create a reference set for scenarios that actually contain a violation. Any set of uncertainty impacts that contains the correct (and no other) uncertainty impacts counts as a true positive TP. Any set that contains either less or more uncertainty impacts than expected counts as a false positive FP. Any missing set counts as a false negative FN.

To calculate the accuracy for **Q2.2** and **Q2.3**, we use the following formulas:

$$\textbf{Precision} = \frac{TP}{TP+FP} \qquad \textbf{Recall} = \frac{TP}{TP+FN}$$

6.2.4. Goal G3 - Usability

With goal **G3** we evaluate the usability of the approach. Usability is mostly determined by user studies. Due to the effort involved in finding experts in the field of software architecture design and conducting the studies, we evaluate the usability here through objective argumentation.

For question **Q3.1** we compare our uncertainty impact filter approach, as implemented in Algorithm 3, with the state of the art from Chapter 3. That means we examine what knowledge is necessary to carry out the previous approaches and to carry out our approach. More precisely, we compare with the approach of scenario rejection from Section 5.3. This represents the latest state of research and provides the most detailed interpretation of the analysis results of the confidentiality analysis to date. It is then examined whether more knowledge is required for the approach we have presented compared to the scenario rejection.

Since it is difficult to quantify knowledge, we choose subject areas to find out what knowledge is necessary. Within the subject areas, there is then a discussion of how the knowledge can be quantified.

For this we define the following concepts:

K1 *Data Flow*, e.g. to know how the elements of the architecture communicate with each other.

K2 *Architecural Model and Characteristics*, e.g. to know how the architecture is modeled and what characteristics the elements of the architecture have.

K3 *Uncertainty*, e.g. knowing how to express uncertainty in the modeling and which elements are affected by uncertainty.

We use exemplary scenarios from our running example from Chapter 4 to assess the necessary knowledge. These are used to work out what knowledge is required to carry out the scenario rejection and for comparing it to the uncertainty impact filter.

Through question **Q3.2** we find out when a software architect can know the additional information to execute the approach. For this, we use the knowledge gained from answering the question **Q3.1** and argue objectively when the necessary quantity of knowledge can be available. Our evaluation of the question is illustrated using the example scenarios that we use for question **Q3.1**.

With question **Q3.3** we want to find out whether our uncertainty impact filter represents an improvement compared to the state of the art. For this, we want to find out whether a software architect receives more information than with previous approaches. In addition, we examine whether the information obtained helps to know more quickly about the effects of uncertainty in the event of violations of confidentiality.

For this, we first examine which information is obtained by applying the scenario rejection from Section 5.3. We also check this for our uncertainty impact filter from Section 5.4. The information is then compared. We use the additional information that our approach provides to assess whether uncertainty can be drawn more quickly. The same scenarios as to answer the question **Q3.1** will be used for this.

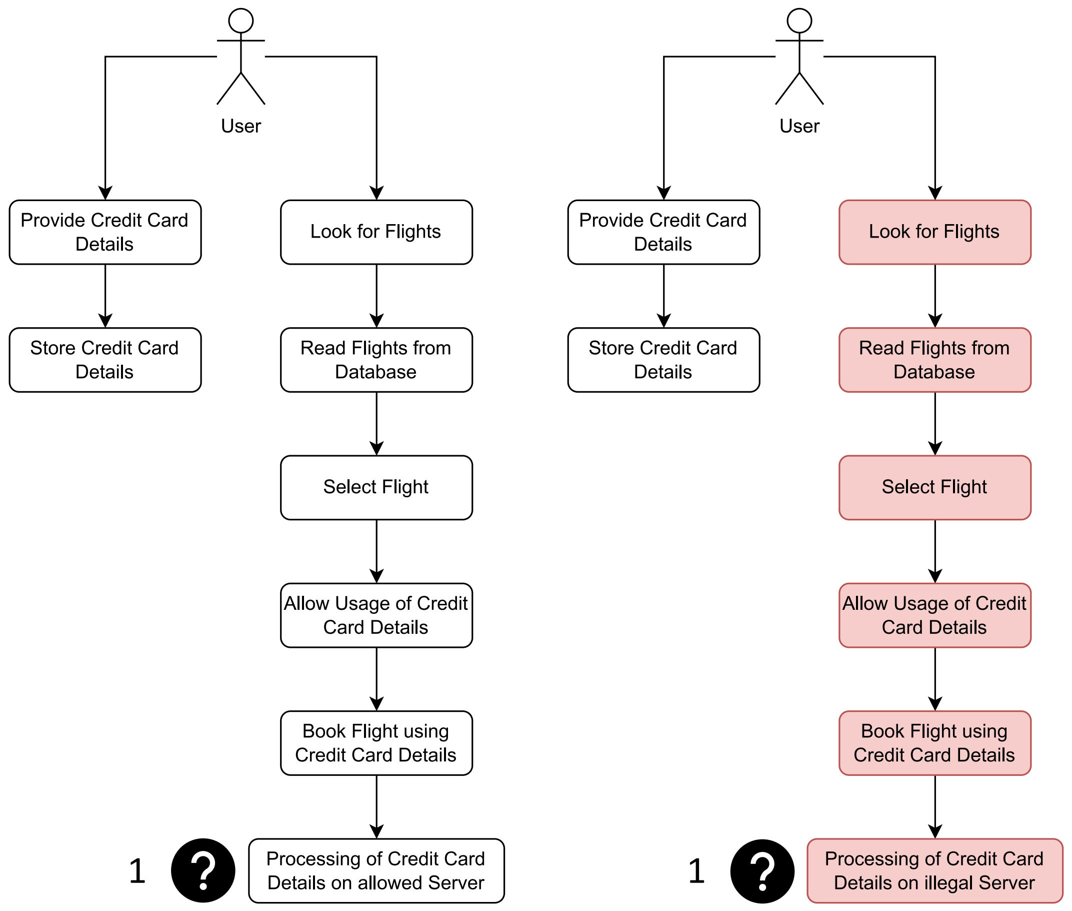

User
Provide Credit Card Details
Store Credit Card Details
Look for Flights
Read Flights from Database
Select Flight
Allow Usage of Credit Card Details
Book Flight using Credit Card Details
1
?
Processing of Credit Card Details on allowed Server
User
Provide Credit Card Details
Store Credit Card Details
Look for Flights
Read Flights from Database
Select Flight
Allow Usage of Credit Card Details
Book Flight using Credit Card Details
1
?
Processing of Credit Card Details on illegal Server

Figure 6.1.: The two scenarios in the presented model of the TravelPlanner (**M1**), the left without a violation - the right with a violation of confidentiality

6.3. Results

In this section, the results of our evaluation are discussed.

6.3.1. Goal G1 - Feasibility

The first goal **G1** examines the feasibility of our concept. Question **Q1.1** evaluates whether the approach is feasible in such a way that the resulting interpretation results can be classified on a non-binary scale. For this purpose we have listed aspects **F1**-**F4** in Subsection 6.2.2 with which we want to answer the question. We already show in Chapter 5 using the running example from Chapter 4 that the theoretical execution of our concept is possible. In addition, we now show that a model **M1** that we did not construct can also be analyzed with the approach. The model was provided with uncertainty by Walter et al. and comes from Katkalov [27, 14]

It has one uncertainty and the uncertainty impact is described by a variation of architectural elements. Two scenarios arise from these two variations. Either the credit card data is used by the airline as the user has consented to or contrary to the consent. In Figure 6.1 we see the two simplified data flows that occur in this architecture. A data flow also corresponds to a scenario, since there is only one data flow per emerging scenario.

The point of uncertainty is marked by the question mark. On the left, we see the scenario where no violation occurs. On the right is the scenario that causes an infraction.

If we follow the implementation from Algorithm 3 we pass these two scenarios along with the uncertainty model. For each scenario, we review the data flow in the scenario. For the first scenario, there is no influencing uncertainty impact, since there is no violation of confidentiality. In the second scenario, a scenario is created in the last element of the right branch. Now it is examined for each element of the data flow whether it lies in the varying subjects of an uncertainty impact. This is the case for the last element of the right branch. It lies in the varying subjects of the single uncertainty impact. Therefore, for this scenario, the uncertainty impact is added to the influencing impacts. We now have the complete result of our analysis.

Aspect **F1** aims at whether the approach is executable. From the implementation just mentioned, we conclude that the approach is executable by showing that an architecture model that we did not design can also be analyzed.

Aspect **F2** examines whether the approach delivers results. We have shown through the case study that this is the case.

Aspect **F3** evaluates whether the results should be classified on a non-binary scale. To do this, we consider the data structure again, as described in Section 5.2. As a result, we get a `UncertaintyFilter` object that links the uncertainty impacts with the scenarios. This represents a collection of facts from which a software architect can draw conclusions. Therefore, the result obtained cannot be classified on a binary scale.

Aspect **F4** evaluates the implementation in reality. For this purpose, a prototypical implementation of the concept was made while working on the thesis. Within these, architectural models and their uncertainty can be modeled with PCM. Based on an existing confidentiality analysis, the architecture is first analyzed for violations of confidentiality. These results are then passed as input to the implementation of our concept. The prototype was tested and developed with our running example from Chapter 4. It delivers the same results as the theoretical processing on paper. Nevertheless, it should be mentioned here that the prototype is also based on a prototype and only a small amount of PCM was used for modeling the architecture. This limitation is discussed in more detail in Section 6.5.

By answering the four aspects **F1**-**F4** asked, the question **Q1.1** is answered. We find that the approach is workable and the results can be categorized on a non-binary scale.

6.3.2. Goal G2 - Accuracy

For question **Q2.1** we check whether the naive approach from Section 5.3 correctly rejects or accepts the architectures. Since all architectures have violations, we choose individual scenarios to show that they do in fact violate confidentiality. Even if we only pick out individual scenarios from the architectures, we do not falsify the result. For the naive approach, an architecture is rejected as soon as there is an advance in at least one scenario. Thus, a violating architecture will be rejected if all scenarios are considered because within those scenarios there must be one that involves a violation. Likewise, an architecture that considers only one scenario that includes a violation will be rejected. To show that our implementation of the naive approach accepts architectures that have no violations, we choose one scenario from each model that does not involve a violation As we see in Table 6.1, all violations are exactly what we expect.

Model	Scenario	Violation	Violation expected?
TravelPlanner	Credit card information used as the user agreed to	none	no
TravelPlanner	Credit card information used against the agreement of the user	Credit card information may only be used as the user agrees to	yes
DistanceTracker	GPS-Data is processed on the users smartphone	none	no
DistanceTracker	GPS-Data is processed on a server	GPS-Data may only be processed on the user's smartphone	yes
OnlineShop	The user enters a personal address and it is stored inside the EU	none	no
OnlineShop	The user enters a personal address and it is stored outside the EU	Personal data may only be stored inside the EU	yes

Table 6.1.: Scenarios of the models for the naive approach

Model	True Positive	False Positive	False Negative	Precision	Recall
OnlineShop	32	0	0	1.0	1.0
TravelPlanner	2	0	0	1.0	1.0
DistanceTracker	2	0	0	1.0	1.0

Table 6.2.: Results of the scenario rejection

For Question **Q2.2** we want to calculate the accuracy of the scenario rejection from Section 5.3. Therefore, the scenario rejection from Section 5.3 can correctly identify all scenarios. Question **Q2.3** asks for the accuracy of our presented approach in Section 5.4. We calculate precision and recall by inserting the values from Table 6.3 into the formulas from Subsection 6.2.3. We calculate the precision for the OnlineShop model as an example: **Precision** $= \frac{32}{32+0} = 1$. We do the same for the recall: **Recall** $= \frac{32}{32+0} = 1$. As we can see in Table 6.2, we get a value of 1.0 for precision and recall for each model. This corresponds to the perfect result. Both for question **Q2.2** and for **Q2.3** it must be mentioned that the models are still small despite five uncertainty impacts in **M3**. This is because the use cases and processes within the models **M1, M2** and **M3** are kept small and simple.

Model	True Positive	False Positive	False Negative	Precision	Recall
OnlineShop	18	0	0	1.0	1.0
TravelPlanner	1	0	0	1.0	1.0
DistanceTracker	1	0	0	1.0	1.0

Table 6.3.: Results of the uncertainty impact filter

6.3.3. Goal G3 - Usability

The third goal **G3** examines the usability of the approach. Question **Q3.1** discusses what knowledge is required to carry out the approach and compares it with the state of the art. Question **Q3.2** evaluates when the additional knowledge is known. With question **Q3.3** we check whether a software architect can draw faster conclusions from our interpretation results.

For question **Q3.1** we discuss which knowledge is necessary for the uncertainty impact filter. In Subsection 6.2.4 we have identified three concepts **C1**-**C3** with which we analyze knowledge. Table 6.4 shows what knowledge we consider necessary to perform scenario rejection. In the "Uncertainty Impact Filter" column, we list whether our approach requires more or more specific knowledge than scenario rejection.

For concept **C1** *Data Flow* we find that the data flow has to be fully modeled through the architecture. The analysis of confidentiality is based on the data flows in the architecture being analyzed for violations [24]. For our approach, no more is necessary for the area of *Data Flow* than for scenario rejection.

In the case of the **C2** *Architectural model and characteristics* concept, it must be clear for scenario rejection which architectural elements are required to communicate with one another and thereby generate data flows in the scenarios. For this, it is also necessary to know the characteristics of the respective elements. These determine properties for architectural elements that are decisive for whether violations occur or not. This is necessary, as the scenarios are a required input parameter in Algorithm 2. The same knowledge is necessary for the uncertainty impact filter.

For concept **C3** *Uncertainty* it is necessary to enter the scenarios of the architecture for the scenario rejection as input paramaters in Algorithm 2. It is not assumed that all scenarios have to follow the same structure. In principle, the respective scenarios could differ fundamentally from one another. Distinguishing the scenarios can be one way in which uncertainty can be expressed. In contrast, for the uncertainty impact filter, it is necessary to specify the uncertainty model as input. The uncertainty model expresses uncertainty in the form of variations at specific points in the architecture. This means that to apply our approach, a software architect must express the uncertainty in terms of variations. By specifically determining the points (i.e. uncertainty impacts) at which uncertainty occurs, our approach can establish a relationship between the occurrence of violations and the uncertainty impacts affected.

We show this using our running example from Chapter 4 and introduce another scenario in Figure 6.2. This scenario contains two violations of confidentiality and includes all five uncertainty impacts of the model. The first violation occurs because

Concept	**Scenario Rejection**	**Uncertainty Impact Filter**
Data Flow	The complete data flow must be known, otherwise, no analysis of confidentiality can take place	No change
Architectural model and characteristics	Architectural elements that are necessary to establish data flow must be known. Their characteristics must also be known, as they lead to violations of confidentiality, among other things	No change
Uncertainty	Uncertainty must be able to be expressed in a way that scenarios can be examined. Scenarios do not necessarily have to contain the same structures	Uncertainty must be expressed in the form of a variational model

Table 6.4.: Comparing required knowledge in the scenario rejection to the uncertainty impact filter

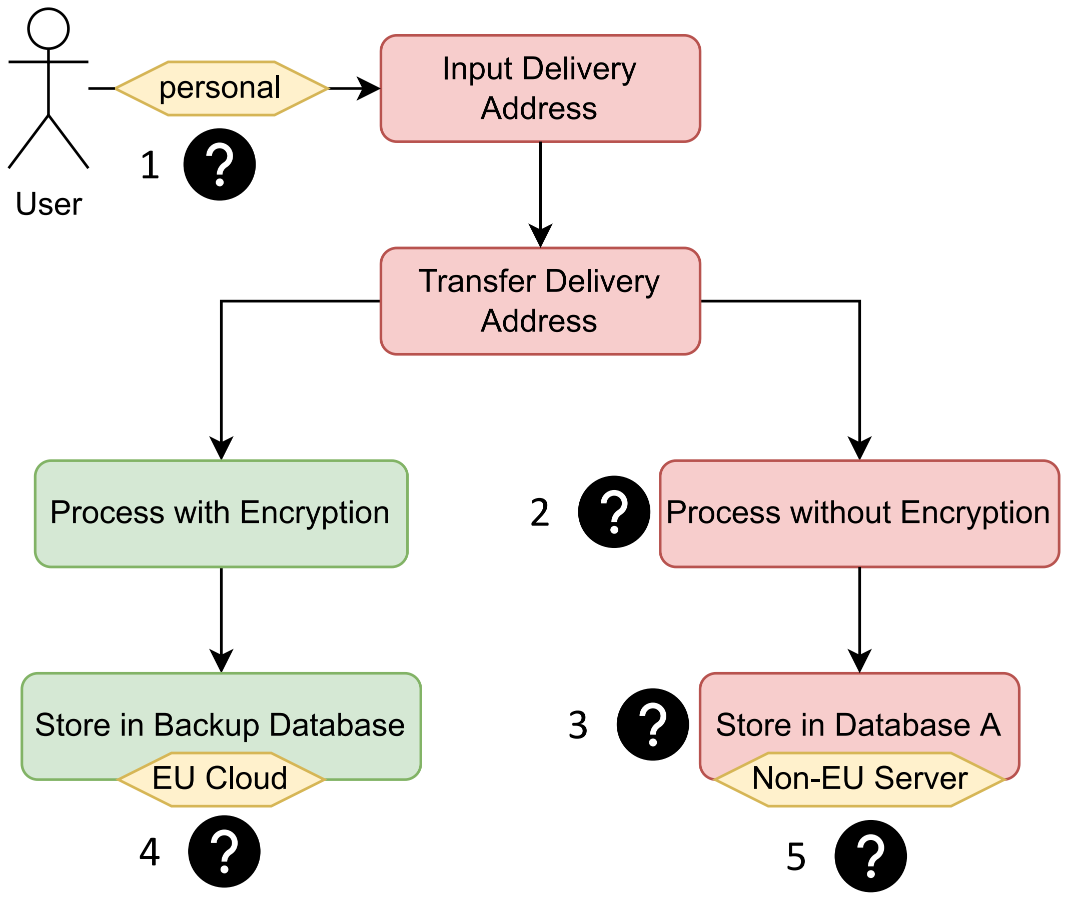


Figure 6.2.: A scenario from the model of the Onlineshop containing two violations

database A stores unencrypted data and the second violation occurs because personal data is not stored in the EU. If we use this scenario for the scenario rejection approach from Algorithm 2, it is not relevant to be able to precisely define the uncertainty impacts. It is only checked whether the confidentiality analysis has identified violations and based on this the scenario is classified. This is different for the uncertainty impact filter from Algorithm 3. This requires the uncertainty impacts since the architectural elements of the sequence with violation mimicking the varying subjects of the uncertainty impacts are compared.

To answer **Q3.2** in which we check, when it is possible to know the required additional knowledge, we use the findings from question **Q3.1**. The additionally required information can be broken down into the uncertainty model. Variations of architectural elements must be described in the uncertainty model. An uncertainty model can be made if the software architect can specify a deterministic choice for as yet undetermined ADD. This represents a limitation of our approach and is taken up again in Section 6.5. With the models **M1**-**M3** we show that it is possible to build an architecture that expresses uncertainty in such a way that a set of points is defined at which uncertainty can be expressed by varying architectural elements. In Table 4.1 the five uncertainties are explicitly listed, which show uncertainty in the example of the online shop. In Subsection 6.2.1 we explain the uncertainties of **M1** and **M2**.

For question **Q3.3** we evaluate whether a software architect can draw conclusions about uncertainty faster with our approach than with scenario rejection. For this purpose, we first consider what information the scenario rejection from Algorithm 2 provides. As described in Section 5.2 and Section 5.3, we get the ScenarioRejection interpretation as a return for scenario rejection. This includes the accepted and rejected scenarios. From the rejected scenarios, the software architect learns where and why violations occur in this scenario. From the accepted scenarios, they learn which scenarios do not lead to violations. This information does not immediately provide any conclusions as to which uncertainty impacts lead to violations.

Now we look at our approach from Section 5.4. We get an interpretation of the type UncertaintyFilter as output. It contains a list of InfluencingUncertainties. The task of the InfluencingUncertainties is to examine a scenario to determine which uncertainty impacts are passed through by a data flow before the violation occurs. This gives us a link between uncertainty impacts and individual scenarios with the interpretation result. From this, a software architect can see which uncertainty impacts should be checked in order to prevent violations of confidentiality. To do this, they can either look at the respective scenarios or summarize the influencing uncertainties of each scenario.

We show this using the scenario Figure 5.5, which was already presented in Section 5.4. For this scenario, our analysis indicates that uncertainties 1, 2 and 3 are responsible for the violation and a violation occurred when storing unencrypted data in the database. In contrast, the scenario rejection states that this scenario creates a violation when storing unencrypted data in the database. We find that a software architect would initially have had to create the connection that he received from our analysis in scenario rejection himself.

From this, we conclude that our approach can provide a software architect with faster information about the affected uncertainty impacts than scenario rejection. Our analysis directly provides the architect with a set of potential architectural elements that could be responsible for this violation. This shortens the design time because the architect saves the step of finding influencing uncertainty impacts.

6.4. Threats to Validity

We use the guidelines presented by Runeson and Höst in “Guidelines for conducting and reporting case study research in software engineering” [21]. In this section, we discuss the threats to construct, internal and external validity and the reliability of our evaluation.
Construct Validity "[...] reflects to what extent the operational measures that are studied really represent what the researcher have in mind and what is investigated according to the research questions" [21, p. 23]. The operational measures used here are the metrics of precision and recall. These are also used in the confidentiality analysis of Seifermann et al. and in the presented approach by Liu [24, 15]. This reduces the risk that we use invalid properties to answer our questions. Furthermore, we use the Goal-Question-Metric approach, with which we set questions, explain how we answer these questions and finally give the answer to the questions to limit this risk.
Internal Validity deals with whether factors other than those investigated influence the result. If this is the case, the evaluation of our concept will be weakened.

This could be the case for aspect **F4** in question **Q1.1**. The prototype was not implemented exactly as presented in Section 5.2. This was the case because the implementation depended on the structures of the prototypical confidentiality analysis. As a result, the readout of the confidentiality analysis was not implemented exactly in the data structures as proposed in our concept. Furthermore, the modeling of architectures with PCM turns out to be complex. Many possibilities to describe an architecture more precisely were left out for the implementation. As a result, details could have been overlooked, which, however, are essential for the realization of the concept in reality.

The same problem exists with question **Q2.3**. In our concept, we abstract all architectural elements to one level. This may not be an adequate abstraction and relevant factors not leading to biased results in our used models. Furthermore, we assume that varying subjects of uncertainty impacts are always passed through by the data flow as soon as they occur in the scenario. As a result, we assume that the uncertainty impacts are always due to the data flow. However, there may be other side effects caused by uncertainty impacts that are not related to the data flow. As a result, an influencing uncertainty impact would not be located on the data flow but still be responsible for a violation.

Nevertheless, the evaluation shows that our approach is applicable to architectures that use the same details as in our models and have uncertainty impacts that lie on data flows.

External Validity "[...] is concerned with to what extent it is possible to generalize the findings [...]" [21, p. 24]. There is a risk for that when answering the questions **Q2.2** and **Q2.3**. The models for the answer of them are small and of low complexity, despite five uncertainty impacts in model **M3**. This is because the use cases and processes within the models **M1, M2** and **M3** are kept small and simple. Due to significantly more complex models, it could be that our analysis can no longer deliver results or the results are falsified.

We also use a self-constructed model of the online shop (**M3**) for the evaluation. There could be a risk here that the approach cannot be used for other models due to the close development based on the example model. Since the basis of the model does not come from this thesis, but from Hahner [10] and we were able to prove at least the basic function on the two other models **M1** and **M2** that we did not create, we see the risk as reduced.

The question **Q3.3** is also affected. It may be that the concepts we have chosen are not sufficient to adequately describe the information required for the usability of the combination algorithm. Since we only need the uncertainty model as an additional input parameter in the uncertainty impact filter in Algorithm 3 compared to the scenario rejection in Algorithm 2, we assume that we can sufficiently differentiate with our choice of concepts.

Reliability "[...] is concerned with to what extent the data and the analysis are dependent on the specific researchers" [21, p. 24]. This means that other people may get different results than we do. We provide the source code and the models used for the evaluation [3]. This reduces the threat of reliability.

6.5. Limitations

Required data flows We assume that software architectures analyzed by our approach always involve data flows. Without data flows, it is currently not possible to apply the confidentiality analysis presented. Furthermore, we use dataflows to get a sorted list of called architecture elements, in the order in which they are called. This is the only way we can draw conclusions in our concept about which uncertainty impacts are possible for a violation.

Expressing uncertainty For our approach, we assume that architectural uncertainty can be expressed through varying architectural elements. This limits the scope of the approach because all the uncertainty that cannot be expressed by a variational model so far cannot be analyzed by our approach either. It is the task of future work to analyze other expressions of architectural uncertainty.

Uncertainty impacts on data flows In the uncertainty impact filter as presented in Section 5.4, we identify uncertainty impacts that could be influential for the occurrence of violations against uncertainty. We do this by analyzing the data flow through the architecture. As a result, we assume that architectural elements that are varied by uncertainty impacts are always inevitably located on the data flow. Otherwise, it would not be possible to establish the connection between an uncertainty impact and a violation of confidentiality.

Technical limitation We closely linked the development of our approach to the development of the prototype implementation [3]. As a result, we were forced to examine only models whose uncertainties are described by variational modeling. Since there are currently not many models that are modeled in this way in the prototypical implementation, we were technically limited.

7. Conclusion

To conclude this thesis, we summarize the contents of our approach in Section 7.1. Finally, we give an outlook on future work in Section 7.2.

7.1. Summary

The goal of this thesis was to provide an approach that establishes a connection between uncertainty impacts and results of confidentiality analyses at design time. The connection corresponds to an analysis result that provides potential uncertainty impacts as the cause of confidentiality violations. With this, we wanted to achieve that a software architect receives information about confidentiality under uncertainty more quickly.

We first reused existing approaches to present violations of confidentiality under uncertainty. This resulted in an expandable combination algorithm consisting of several stages. We extend the previous approaches by introducing the uncertainty impact filter. Compared to previous approaches, we need a new input parameter for our approach, the uncertainty model. From this model, we get scenarios that express the uncertainty of a software architecture. In scenarios that contain a violation of confidentiality, we review the data flow in the scenario. The data flow is represented by different sequences. Sequences are characterized by the fact that architectural elements are sorted in the way data flows through them. We separate architectural elements based on whether they occur in a sequence before or after the element where the violation occurs. For each architectural element, that is located before the element where a violation occurs, we check whether it is varied by an uncertainty impact. If this is the case, we identify the uncertainty impact as influencing. The remaining uncertainty impacts are identified as not influencing.

Our approach enables software architects to point out architectural elements potentially involved in confidentiality violations. This is done at design time and while there is architectural uncertainty. The prototypical implementation can automatically analyze models designed with Palladio. As a result, a software architect can conclude the preservation of confidentiality in the architecture at any time in the design process. For this, uncertainty must be expressed in terms of variations in architectural models.

We evaluated our approach for feasibility, usability and case studies. It turns out that our approach is feasible and provides results that relate architectural uncertainty to violations of confidentiality. To do this, we demonstrated the approach using a model that we did not create. The approach can be used with the addition of new information and provides a software architect with information more quickly than previous approaches do. To identify the new information, the knowledge required for previous approaches was compared with our approach. Our approach was able to identify the expected uncertainty impacts that are potentially responsible for a violation in all case studies. We calculate precision and recall with 1.0.

7.2. Future Work

A first step that could take place in the future is the creation of further models that can be analyzed with our approach to test it further. For example, the model for a port communication system already exists [6]. Given the origin of the port model [7], policies for confidentiality could be derived, which would allow our approach to analyze the model. What is important for this is a better implementation of the concept than was done in the prototype as part of this thesis.

Future work may elaborate extensions of our combination algorithm. A first step would be to examine uncertainty impacts more closely. Currently, this is read directly from the data flows in which violations of confidentiality occur. By comparing the sequences with and without violations, it might be possible to sort out uncertainty impacts that we identify as potentially affecting as not affecting.

Furthermore, it can be examined whether the abstraction of the architectural elements on one level is appropriate. For this purpose, the influence of different types of architectural elements must be differentiated more precisely.

In the long run, new insights into architectural uncertainty can lead to new extensions of the algorithm. This would then be able to take into account not only uncertainty that can be described by varying architectural models. This could probably also identify uncertainty impacts that are not based on data flows.

Bibliography

[1] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. "The goal question metric approach". In: *Encyclopedia of software engineering* (1994), pp. 528–532.

[2] Niko Benkler. *Architecture-based Uncertainty Impact Analysis for Confidentiality (Reproduction Set).* 2022. DOI: `10.5281/zenodo.6202285`. URL: `https://publikationen.bibliothek.kit.edu/1000144639` (visited on 05/10/2022).

[3] Tizian Bitschi. *Prototype Implementation: Palladio-Addons-Uncertainty-VariationAnalysis.* May 2022. URL: `https://doi.org/10.5281/zenodo.7236106`.

[4] B. Boehm and V.R. Basili. "Software Defect Reduction Top 10 List". In: *Computer* 34.1 (Jan. 2001). Conference Name: Computer, pp. 135–137. ISSN: 1558-0814. DOI: `10.1109/2.962984`.

[5] Nicolas Boltz et al. "Handling Environmental Uncertainty in Design Time Access Control Analysis". In: (2022).

[6] *Case Study Port Communication System.* en. URL: `https://github.com/FluidTrust/CaseStudies` (visited on 06/07/2022).

[7] *Case Study Port Communication System.* URL: `https://github.com/FluidTrust/CaseStudies/tree/main/bundles/edu.kit.kastel.dsis.fluidtrust.casestudy.pcs.model/model` (visited on 06/07/2022).

[8] Tom DeMarco. "Structure Analysis and System Specification". en. In: *Pioneers and Their Contributions to Software Engineering: sd&m Conference on Software Pioneers, Bonn, June 28/29, 2001, Original Historic Contributions.* Ed. by Manfred Broy and Ernst Denert. Berlin, Heidelberg: Springer, 2001, pp. 255–288. ISBN: 978-3-642-48354-7. DOI: `10.1007/978-3-642-48354-7_9`. URL: `https://doi.org/10.1007/978-3-642-48354-7_9` (visited on 10/13/2022).

[9] K. Goseva-Popstojanova and S. Kamavaram. "Assessing uncertainty in reliability of component-based software systems". In: *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.* 14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003. ISSN: 1071-9458. Nov. 2003, pp. 307–320. DOI: `10.1109/ISSRE.2003.1251052`.

[10] Sebastian Hahner. "Architectural Access Control Policy Refinement and Verification under Uncertainty". In: *Companion Proceedings of the 15th European Conference on Software Architecture.* 15th European Conference on Software Architecture (ECSA 2021), Online, 13.09.2021 – 17.09.2021. ISSN: 1613-0073. 2021. URL: `https://publikationen.bibliothek.kit.edu/1000139152` (visited on 05/07/2022).

[11] Sebastian Hahner. *Domain-specific Language for Data-driven Design Time Analyses and Result Mappings for Logic Programs.* de. 2020. DOI: `10.5445/IR/1000123271`. URL: `https://publikationen.bibliothek.kit.edu/1000123271` (visited on 09/27/2022).

[12] Heather M. Hinton and E. Stewart Lee. *The compatibility of policies | Proceedings of the 2nd ACM Conference on Computer and communications security*. 1994. URL: https://dl.acm.org/doi/10.1145/191177.191243 (visited on 04/29/2022).

[13] *ISO/IEC 27000:2018*. ISO. 2018. URL: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/07/39/73906.html (visited on 06/04/2022).

[14] Kuzman Katkalov. "Ein modellgetriebener Ansatz zur Entwicklung informationsflusssicherer Systeme". de. In: (2017). URL: https://opus.bibliothek.uni-augsburg.de/opus4/frontdoor/index/index/docId/4339 (visited on 10/13/2022).

[15] Oliver Liu. *Design Space Evaluation for Confidentiality under Architectural Uncertainty*. 2021. DOI: 10.5445/IR/1000139590. URL: https://publikationen.bibliothek.kit.edu/1000139590 (visited on 04/29/2022).

[16] Steve McConnell. *Software project survival guide*. Pearson Education, 1998.

[17] Diego Perez-Palacin and Raffaela Mirandola. "Uncertainties in the modeling of self-adaptive systems: a taxonomy and an example of availability evaluation". In: *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*. ICPE '14. New York, NY, USA: Association for Computing Machinery, Mar. 22, 2014, pp. 3–14. ISBN: 978-1-4503-2733-6. DOI: 10.1145/2568088.2568095. URL: https://doi.org/10.1145/2568088.2568095 (visited on 05/30/2022).

[18] *PerOpteryx*. URL: https://sdqweb.ipd.kit.edu/wiki/PerOpteryx (visited on 05/31/2022).

[19] Alisson Puska, Michele Nogueira, and Aldri Santos. "Confidentiality-Aware Decision on Handoffs under Uncertainty on Heterogeneous Wireless Networks". In: *2018 IEEE Symposium on Computers and Communications (ISCC)*. 2018 IEEE Symposium on Computers and Communications (ISCC). ISSN: 1530-1346. June 2018, pp. 00884–00889. DOI: 10.1109/ISCC.2018.8538677.

[20] Ralf H. Reussner et al. *Modeling and Simulating Software Architectures – The Palladio Approach*. ISBN: 9780262034760. 2016. URL: https://publikationen.bibliothek.kit.edu/1000071486 (visited on 06/07/2022).

[21] Per Runeson and Martin Höst. "Guidelines for conducting and reporting case study research in software engineering". en. In: *Empirical Software Engineering* 14.2 (Dec. 2008), p. 131. ISSN: 1573-7616. DOI: 10.1007/s10664-008-9102-8. URL: https://doi.org/10.1007/s10664-008-9102-8 (visited on 10/14/2022).

[22] Peter Schaar. *Privacy by Design*. Apr. 1, 2022. URL: https://link.springer.com/article/10.1007/s12394-010-0055-x (visited on 06/02/2022).

[23] Stephan Seifermann, Robert Heinrich, and Ralf Reussner. "Data-Driven Software Architecture for Analyzing Confidentiality". In: *2019 IEEE International Conference on Software Architecture (ICSA)*. 2019 IEEE International Conference on Software Architecture (ICSA). Mar. 2019, pp. 1–10. DOI: 10.1109/ICSA.2019.00009.

[24] Stephan Seifermann et al. "Detecting violations of access control and information flow policies in data flow diagrams". In: *Journal of Systems and Software* 184 (Feb. 1, 2022), p. 111138. ISSN: 0164-1212. DOI: 10.1016/j.jss.2021.111138. URL: https://www.sciencedirect.com/science/article/pii/S0164121221002351 (visited on 04/29/2022).

[25] *Verarbeitung besonderer Kategorien personenbezogener Daten.* URL: https://dejure.org/gesetze/DSGVO/9.html (visited on 06/07/2022).

[26] W.E. Walker et al. "Defining Uncertainty: A Conceptual Basis for Uncertainty Management in Model-Based Decision Support". In: *Integrated Assessment* 4.1 (Mar. 1, 2003). Publisher: Taylor & Francis _eprint: https://doi.org/10.1076/iaij.4.1.5.16466, pp. 5–17. ISSN: 1389-5176. DOI: 10.1076/iaij.4.1.5.16466. URL: https://doi.org/10.1076/iaij.4.1.5.16466 (visited on 05/17/2022).

[27] Maximilian Walter et al. "Architectural Optimization for Confidentiality under Structural Uncertainty". In: (2022).

[28] Josephine Wolff and Nicole Atallah. "Early GDPR Penalties: Analysis of Implementation and Fines Through May 2020". In: *Journal of Information Policy* 11.1 (Jan. 1, 2021), pp. 63–103. ISSN: 2381-5892. DOI: 10.5325/jinfopoli.11.2021.0063. URL: https://doi.org/10.5325/jinfopoli.11.2021.0063 (visited on 06/07/2022).

A. Appendix

A.1. Abbreviations

ADD	Architectural Design Decision
DFD	Data Flow Diagram
GDPR	General Data Protection Regulation
PbD	Privacy by Design
PCM	Palladio Component Model