Kopplung von Architekturanalysen und musterbasierten Quelltextanalysen für Sicherheitseigenschaften von Aufrufabhängigkeiten

Bachelorarbeit von

Laura Traub

an der Fakultät für Informatik
Institut für Informationssicherheit und Verlässlichkeit (KASTEL)

Erstgutachter:	Prof. Dr. Ralf Reussner
Zweitgutachter:	Prof. Dr. Anne Koziolek
Betreuender Mitarbeiter:	M.Sc. Frederik Reiche
Zweiter betreuender Mitarbeiter:	Dr. Robert Heinrich

11. Juli 2022 – 11. November 2022

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst, und weder ganz oder in Teilen als Prüfungsleistung vorgelegt und keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die benutzten Werken im Wortlaut oder dem Sinn nach entnommen sind, habe ich durch Quellenangaben kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen sowie für Quellen aus dem Internet.

Karlsruhe, 10. November 2022

. . . . . .

(Laura Traub)

Zusammenfassung

Die Vernetzung von Software über das Internet und andere Kanäle stellt eine grundsätzliche Gefahr für die Sicherheit von Daten und Systemen dar. Gelangen Informationen in die falschen Hände können enorme wirtschaftliche und soziale Schäden entstehen. Es ist deshalb wichtig die Sicherheit von Systemen bereits zur Entwurfszeit zu berücksichtigen. Mittels Analysewerkzeugen auf Architektursicht können Sicherheitseigenschaften auf einer höheren Abstraktionsebene frühzeitig definiert und überprüft werden. Auf Quelltextsicht bieten statische, musterbasierte Analysewerkzeuge einen Ansatz zur Überprüfung der korrekten Verwendung von kritischen Schnittstellen. Bisher wurde noch keine Kombination dieser beiden Analyseansätze vorgenommen, um die auf Architektursicht getroffenen Annahmen der im Quelltext umgesetzten Sicherheitseigenschaften definiert auf Aufrufabhängigkeiten auf fehlerhafte Umsetzung zu überprüfen. Deshalb wird untersucht, wie sich eine Kopplung der beiden Sichten und eine Rückführung der Ergebnisse einer Quelltextanalyse in die Architektursicht realisieren lässt. Die vorliegende Arbeit definiert zunächst die für eine Kopplung notwendigen Eigenschaften der Analysen. Darauf basierend wird dann ein Ansatz für eine Kopplung konzipiert. Eine konkrete Umsetzung des Ansatzes wurde im Rahmen der vorliegenden Arbeit mit der Zugriffsanalyse von Kramer auf Architektursicht und CogniCrypt auf Quelltextsicht in Java vorgenommen. Die Evaluation des Ansatzes erfolgt anhand eines Fallbeispiels. Die Ergebnisse zeigen, dass die Kopplung von Architekturanalysen mit musterbasierten Quelltextsicherheitsanalysen für Sicherheitseigenschaften von Aufrufabhängigkeiten machbar ist und dass durch die Kopplung von Quelltextfehlern mit der Architekturanalyse zusätzliche Schwachstellen aufgedeckt werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CG Aufrufgraph

DAST Dynamic Application Securty Testing

FK Fehlerklasse

F Forschungsfrage

IAST Interactive Application Securty Testing

JCA Java Cryptographic Architecture

JOANA Java Object-sensitive ANAlysis

LR LinkingResouce

MF Musterfehler

PCM Palladio Component Model

PDG Programmabhängigkeitsgraph

SAST Statc Application Securty Testing

SDG Systemabhängigkeitsgraph

SE Sicherheitseigenschaft

UML Unified Modeling Language

Z Ziel

1 Einleitung

Die Anzahl der Angriffe auf Softwaresysteme ist in den vergangenen Jahren immer weiter angestiegen [1]. Verläuft ein Angriff erfolgreich können Angreifer so zum Beispiel unberechtigten Zugriff auf Systeme und Daten erlangen. Dadurch können die persönlichen Daten von Individuen missbraucht werden und dadurch ein sozialer, kultureller, politischer oder ökonomischer Schaden entstehen [2]. Umso wichtiger ist es, dass Systeme sicher gegenüber solchen Angriffen sind. Unter anderem sorgt etwa die steigende Komplexität von Softwaresystemen dafür, dass es immer schwieriger wird, eine Software sicher zu entwerfen und so zu implementieren, dass Schwachstellen verhindert werden [3].

Um die Sicherheit von Softwaresystemen sicherzustellen, können während der Entwicklung Analyse-Werkzeuge eingesetzt werden, welche den Code auf verschiedenen Sichten (z.B. Quelltext, Architektur) zur Verbesserung der Informationssicherheit überprüfen.

Es ist unter anderem möglich, die Sicherheit eines IT-Systems auf einer höheren Abstraktionsebene, auf der Architektursicht, zu analysieren [4]. Dadurch können zum Beispiel die Abhängigkeiten zwischen den Komponenten und deren Eigenschaften definiert und analysiert werden. Gleichermaßen ist es auch möglich den gitterbasierten Informationsfluss innerhalb eines Quelltextes [5] oder die korrekte, musterbasierte Verwendung von kryptographischen Schnittstellen [6] statisch, während der Entwicklung, im Code zu überprüfen.

Bei der Analyse auf der Architektursicht müssen Annahmen über die korrekte Umsetzung von definierten Sicherheitseigenschaften im Quelltext vorgenommen werden, da diese auf der Architektursicht nicht direkt überprüft werden können. Eine Analyse muss auf der Architektursicht zum Beispiel annehmen, dass eine geforderte Sicherheitsanforderung an eine Methode auch korrekt, gemäß der Spezifikation, implementiert wird. Ist dies nicht der Fall, liefert die Analyse auf der Architektursicht falsche Ergebnisse im Vergleich zu dem realen System.

Eine Möglichkeit zur Umgehung des Problems ist es, Informationen der Quelltextanalyse in der Architekturanalyse zu verwenden. In einer vorhergehenden Arbeit wurde deshalb bereits erstmals untersucht, wie die Kopplung zwischen statischen Architekturanalysen und statischen, spezifikationsbasierten Quelltextanalysen erreicht werden kann [7].

In dieser Bachelorarbeit wird untersucht werden, wie ein Ansatz und geeignete Mechanismen zur Kopplung von Architekturanalysen und statischen, *musterbasierten* Quelltextanalysen realisiert werden können.

Zur Beantwortung der Problemstellung werden die Eigenschaften und Ausgaben der anwendbaren Architektur- und Quellcodesicherheitsanalysen für eine Kopplung bestimmt. Darauf basierend wird ein Ansatz und entsprechende Mechanismen für die Kopplung von Architekturanalysen und musterbasierten Quelltextanalysen für Sicherheitseigenschaften von Aufrufabhängigkeiten entwickelt. Anschließend wird der entwickelte Ansatz in ein Softwarewerkzeug überführt und das Kopplungsverfahren evaluiert.

Die erwarteten Vorteile dieser Bachelorarbeit sind wie folgt: (i) Annahmen aus der Architekturanalyse können mittels Quelltextanalyse verifiziert werden, (ii) Diskrepanzen zwischen Architekturanalysen und Quelltextanalysen werden erkannt, (iii) es werden mehr Sicherheitsfehler als ohne eine Kopplung der Analysen aufgedeckt, und (iv) die Softwaresicherheit wird insgesamt verbessert.

Diese Arbeit ist wie folgt aufgebaut: In Kapitel 2 werden zunächst wichtige Grundlagen beschrieben. Kapitel 3 umfasst verwandte Arbeiten und grenzt die vorliegende Bachelorarbeit von diesen ab. In Kapitel 4 werden die Eigenschaften anwendbarer Sicherheitsanalysen auf Architektur- und Quelltextsicht beschrieben. Das Konzept der Kopplung und zwei konkrete Umsetzungen eines Kopplungsansatzes werden in Kapitel 5 beschrieben. Die Implementierung von zwei Kopplungsvarianten wird in Kapitel 6 aufgezeigt. Die Evaluation und die dazugehörigen Ergebnisse werden in Kapitel 7 beschrieben und diskutiert. Abschließend wird in Kapitel 8 ein Fazit gezogen und ein Ausblick auf mögliche weitere Arbeiten gegeben.

2 Grundlagen

In diesem Kapitel werden Grundlagen für diese Arbeit vorgestellt. In Abschnitt 2.1 wird Softwaresicherheit und wie sich diese zusammensetzt beschrieben. Der Abschnitt 2.2 führt Grundlagen der Modell-getriebenen Softwareentwicklung, das Konzept von Modellen und Metamodellen, sowie Palladio ein. Abschnitt 2.3 stellt statische Analysen sowohl auf Architektur- als auch Quellcodesicht vor. Weiterhin wird der Ansatz Confidentiality4CBSE als Ansatz zur Architektursicherheitsanalyse in Absatz 2.3.2 eingeführt. Außerdem werden in Abschnitt 2.3.4 und Abschnitt 2.3.5, musterbasierte und spezifikationsbasierte Quelltextanalysen voneinander abgegrenzt. Für musterbasierte Quelltextsicherheitsanalysen wird CogniCrypt in Absatz 2.3.4 detaillierter beschrieben. Abschließend wird die Abbildung von Quelltext auf Graphen und JOANA zur Erzeugung von Systemabhängigkeitsgraphen in Abschnitt 2.4 beschrieben.

2.1 Software Sicherheit

McGraw [8] definiert Softwaresicherheit als "the idea of engineering software so that it continues to function correctly under malicious attack." (deutsch: "die Idee Software zu entwickeln, die auch bei bösartigen Angriffen weiter funktioniert"). So führen Fehler im Entwurf oder der Implementierung von Software dazu, dass Angreifer diese zur ihren eigenen Gunsten ausnutzen. Dies birgt insbesondere für moderne Anwendungen, welche über das Internet bereitgestellt werden, große Gefahren. Die steigende Komplexität von Anwendungen führt dazu, dass sicherheitsrelevante Fehler in Software immer wahrscheinlicher werden [3]. Deshalb ist es wichtig, dass die Entwickler und Prozesse, welche zur Erstellung von Software eingesetzt werden, zur Einhaltung von Sicherheitsstandards geschult und insbesondere mit geeigneten Rahmenwerken zur Automatisierung unterstützt werden. In einer Analyse des gesamten Entwicklungsprozesses von Software wurden zu diesem Zweck von Mohammed u. a. [9] 118 relevante Veröffentlichungen und deren Softwaresicherheitsansätze untersucht und in 52 Ansätze gemäß fünf übergeordneten Kategorien klassifiziert. Diese fünf übergeordneten Kategorien sind, wie folgt:

1. Modellierung von Sicherheitsanforderungen
2. Schwachstellen Identifikation, Adaption und Abwehr
3. Softwaresicherheits-fokussierte Prozesse
4. Erweiterung UML-basierter Sicherheitsmodellierungsprofile
5. Nicht-UML-basierte Sicherheitsmodellierungsnotationen

Eine Kombination dieser verschiedenen Ansätze ist ebenfalls möglich [7, 10].

2.2 Modellgetriebene Softwareentwicklung

In diesem Abschnitt wird modellgetriebene Softwareentwicklung beschrieben und wofür diese verwendet wird. Daraufhin wird in Abschnitt 2.2.1 das Konzept von Modellen und Metamodellen genannt. In Abschnitt 2.2.2 wird das Palladio Component Model eingeführt.

Die modellgetriebene Softwareentwicklung zeichnet sich dadurch aus, dass das Entwerfen von Modellen im Mittelpunkt steht [11]. Hierbei werden Systeme mit Hilfe von Modellen entworfen und aus diesen automatisch Quelltext erzeugt. Ein modellgetriebener Entwurf hat zum einen den Vorteil, dass Konzepte einfacher und übersichtlicher modelliert werden können. Des Weiteren kann das System so unabhängig von der zu Grunde liegenden Technologie entworfen werden, was den Entwurf vereinfacht. Zur Beschreibung der Modelle wird in der Regel eine standardisierte Modellierungssprache eingesetzt. Für bestimmte Anwendungsfälle gibt es zum Beispiel auch domänespezifische Modellierungssprachen [12].

Ein wichtiges Konzept der modellgetriebenen Entwicklung sind demnach die Modelle der zu entwickelnden Software und Metamodelle in der die Modelle beschrieben werden. Im Folgenden Abschnitt 2.2.1 wird dieses Konzept näher erläutert.

2.2.1 Modelle und Metamodelle

Stachowiak [13] definiert ein Modell anhand der drei Merkmale Abbildung, Verkürzung und Pragmatismus.

- **Abbildungsmerkmal:** Ein Modell ist die Abbildung eines Originals, das aus der Realität stammt.
- **Verkürzungsmerkmal:** Ein Modell bildet nicht alle Eigenschaften des Originals ab. Es stellt nur die Eigenschaften dar, die für den Ersteller oder Betrachter des Modells wichtig sind.
- **Pragmatisches Merkmal:** Modelle werden zu einem bestimmten Zweck und für einen Benutzer oder Betrachter erstellt.

Selic [11] definiert hierfür fünf Eigenschaften, die Modelle haben müssen, um möglichst effektiv für die modellgetriebene Entwicklung zu sein. Zum einen müssen die Modelle, um einfach nachzuvollziehen und übersichtlich zu sein, einen gewissen Abstraktionsgrad besitzen. Sie müssen verständlich für den Betrachter sein. Außerdem sollten sie die modellierten Sachverhalte möglichst genau darstellen können und so repräsentativ sein, dass die Eigenschaften des modellierten Systems aus dem Modell korrekt abgelesen werden können. Zu guter Letzt sollte es immer günstiger sein ein Modell eines Systems zu erstellen und zu analysieren, als das modellierte System selbst.

Ein Beispiel für ein Modell, welches im Bereich der Softwareentwicklung verwendet wird ist die Unified Modeling Language (UML) [14]. Mithilfe dieser Modellierungssprache können Softwarearchitekturen und Softwarekomponenten und deren Eigenschaften und Beziehungen modelliert und grafisch dargestellt werden.

Ein Modell wird durch ein Metamodell beschrieben. Das Metamodell definiert die Syntax eines Modells. Die Syntax eines Modells bestimmt die Elemente und Relationen,

mit denen das Modell beschrieben werden kann. In einem UML Klassendiagramm sind das die Elemente mit denen verschiedenste Softwarebausteine, wie zum Beispiel Klassen, Attribute und Methoden dargestellt werden können.

2.2.2 Palladio Component Model

Ein Ansatz zur komponentenbasierten Software-Entwicklung ist das Palladio Component Model (PCM) [15]. Mit Hilfe des PCM können Systemarchitekturen durch verschiedene Modelle beschrieben werden. Das *repository model* dient zur Beschreibung der einzelnen Komponenten und Schnittstellen. Außerdem definiert das *repository model* die durch eine Komponente angebotenen und benötigten Schnittstellen. Das *system model* definiert welche Komponenten in einem System verwendet werden und deren Verbindungen untereinander. Aus einem mit dem Palladio Komponenten Modell entworfenen System lässt sich ein Quelltext Skelett eines Java Programms generieren, welches anschließend durch den Entwickler ergänzt und vervollständigt werden kann. Es existieren diverse Erweiterungen, die auf dem PCM basieren und mit deren Hilfe weitere Eigenschaften einer Software auf Architekturebene definiert und untersucht werden können. Ein Beispiel für solch eine Erweiterung ist der für die Zugriffsanalyse entwickelte Ansatz Confidentiality4CBSE [16], welcher in Abschnitt Absatz 2.3.2 eingeführt wird.

2.3 Statische Analysen

In der Domäne der Softwareentwicklung werden statische Analysen dazu verwendet Artefakte auf spezifische Eigenschaften zu überprüfen, ohne die dazugehörige Anwendung selbst auszuführen [6, 17, 4, 18]. Durch Verwendung von statischen Analysen im Sicherheitskontext kann die Sicherheit eines Systems verbessert werden. In diesem Kapitel werden zunächst in Abschnitt 2.3.1 statische Architekturanalysen und in Abschnitt 2.3.2 statische Sicherheitsanalysen auf Architektursicht vorgestellt. Es wird der Ansatz Confidentiality4CBSE zur Architektursicherheitsanalyse in Absatz 2.3.2 beschrieben. Anschließend wird in Abschnitt 2.3.3 auf statische Quelltextanalysen und in Abschnitt 2.3.4 spezieller auf statische musterbasierte Quelltextanalysen, sowie in Absatz 2.3.4 auf die Quelltextanalyse CogniCrypt, eingegangen.

2.3.1 Statische Architekturanalysen

Software wird in Form verschiedener Komponenten geplant und implementiert. Bei der Entwicklung einer geeigneten Architektur müssen Softwareentwickler deshalb die Organisation und Abhängigkeiten zwischen den Komponenten und deren Schnittstellen festlegen. Statische Architekturanalysewerkzeuge können Entwicklern deshalb dabei helfen, die Eigenschaften von Abhängigkeiten zwischen verschiedenen Komponenten eines Systems zu definieren und zu überprüfen [16].

2.3.2 Statische Sicherheitsanalysen auf Architektursicht

Um Sicherheitslücken bereits zur Entwurfszeit aufzudecken, ist es möglich statische Sicherheitsanalysen auf Architektursicht durchzuführen [4, 18]. Hierfür können Metamodelle zur Beschreibung von Sicherheitseigenschaften einer Softwarearchitektur entwickelt werden. Es ist auch möglich bereits bestehende Modellierungssprachen, wie zum Beispiel UML, um entsprechende Notationen, welche Sicherheitseigenschaften definieren können, zu erweitern [18]. Mittels vorgegebener Regeln können die erstellten Modelle dann statisch auf Sicherheitseigenschaften geprüft werden [4].

Die Sicherheit eines Systems muss für alle Teile dieses Systems gelten. Dies kann nicht optimal gewährleistet werden, wenn die Sicherheit eines Systems erst bei der Implementation berücksichtigt wird [19]. Wird hingegen schon beim Entwurf eines Systems die Sicherheit betrachtet und mit entsprechenden Analysen überprüft, kann hier die Sicherheit des gesamten Systems und dessen Komponenten sichergestellt werden. Außerdem können so Kosten eingespart werden, da Sicherheitslücken bereits vor der Implementation eines Systems gefunden werden können. [18]

Für Java gibt es verschiedene Ansätze zur statischen Sicherheitsanalyse auf Architektursicht. Dazu gehören etwa UmlSec [18], Carisma [4] und PreReqSec [20].

Zugriffsanalyse auf Basis von Confidentiality4CBSE Confidentiality4CBSE [16] ist eine Erweiterung des Palladio Component Model. Damit können Vertraulichkeitseigenschaften auf Architektursicht festgelegt werden und darauf basierend dann eine Zugriffsanalyse [16] durchgeführt werden. Confidentiality4CBSE zeichnet sich durch die Definition einer Reihe von Strukturelementen aus. Sogenannte `Services` beschreiben einzelne Funktionalitäten mit Ein- und Ausgabe Parametern. Die verschiedenen `Services` werden in `Interfaces` gebündelt. `Components` stellen die konkrete Realisierung einer oder mehrerer `Interfaces` dar. Diese werden im `AssemblyContext` einem `ResourceContainer` zugeordnet. Die Kommunikation der Komponenten zwischen verschiedenen `ResourceContainern` erfolgt über `LinkingResources`. Auf dieser Basis können Sicherheitseigenschaften festgelegt werden. Unter anderem kann für `LinkingResources` festgelegt werden, dass die Kommunikation von Daten zwischen Komponenten verschlüsselt («`encrypted`») erfolgen soll. Hingegen kann für bestimmt Datensätze, welche über eine `LinkingResource` übergeben werden aber auch ausgeschlossen werden, dass diese die «`encrypted`» Eigenschaft erfüllen (`unencryptedData`). Weiterhin ist es möglich, die Eigenschaft «`tamper-protection`» auf `LinkingResources` zu definieren. Wird diese Eigenschaft festgelegt, kann ausgeschlossen werden, dass ein möglicher Angreifer überhaupt Zugriff auf die Kommunikation einer `LinkingResource` und somit auf die darüber kommunizierten Daten erlangen kann.

Auf Basis des Confidentiality4CBSE Modells einer Anwendung kann eine Zugriffsanalyse durchgeführt werden. Die Zugriffsanalyse überprüft zum Beispiel, ob ein Angreifer gemäß der Spezifikation der Architektur Zugriff auf Daten erlangen könnte, die für diesen verborgen bleiben sollten. Die Ausgabe der Analyse ist eine Reihe von Schwachstellen im Architekturmodell.

2.3.3 Statische Quelltext-Analyse

Mit Hilfe von statischen Quelltextanalysen wird Software überprüft, ohne diese auszuführen [21]. Dadurch unterscheidet diese sich von dynamischen Quelltextanalysen, für die eine Ausführung der Software notwendig ist. Um Quelltext statisch zu analysieren können Regeln festgelegt werden, welche anschließend auf ihre Einhaltung überprüft werden. Eine Möglichkeit ist, dass solche Regeln für die Verwendung von Softwarebibliotheken definiert werden, zum Beispiel in Form der erlaubten Aufrufreihenfolge von Methoden oder der erlaubten Parameterwerte [6]. Eine weitere Möglichkeit zur Regeldefinition ist es diese spezifisch für das eigene Programm festzulegen, zum Beispiel in Form von Annotationen, welche direkt in den Quelltext eingefügt werden [5]. So kann etwa der Informationsfluss innerhalb des Programms überprüft werden. Laut Lipp, Banescu und Pretschner [22] sind dynamische Analysen im Vergleich zu statischen Quelltextanalysen oftmals einfacher zu verwenden. Im Allgemeinen ist die statische Analyse bei der Fehlersuche außerdem schneller als die dynamische, wodurch sie leichter auf große Quelltextmengen anzuwenden ist. Da die dynamische Analyse tatsächliche Fehler während der Laufzeit des Systems liefert, gibt es keine falsch-positiven Ergebnisse. Im Sicherheitskontext werden mittels dynamischer Analyse also nur Sicherheitsprobleme erkannt, die auch wirklich existieren. Es kann jedoch zu falsch-negativen Ergebnissen kommen, das heißt es können zum Beispiel Sicherheitslücken übersehen werden.

Ein Beispiel für die statische Quelltext Überprüfung ist Checkstyle [23], welches zur statischen Analyse des Programmierstils von Java Programmen eingesetzt wird. Im Kontext der vorgeschlagenen Arbeit ist außerdem die im Folgenden beschriebene Quelltextanalyse CogniCrypt [6] für die statische, musterbasierte Analyse von Sicherheitsproblemen relevant.

2.3.4 Statische, Musterbasierte Sicherheitsanalysen auf Quelltextsicht

Statc Application Securty Testing (SAST) Werkzeuge sind eine Unterkategorie der statischen Quelltextanalyse. SAST-Werkzeuge werden dazu verwendet den Quelltext, aber auch die Kompilate einer Software statisch auf Sicherheitslücken zu überprüfen [24]. Sie grenzen sich von Dynamic Application Securty Testing (DAST), wodurch Quelltext zur Laufzeit überprüft wird, und Interactive Application Securty Testing (IAST), welche auch Eingaben des Benutzers, Konfigurationen und ähnliches analysieren, ab [25]. SAST-Werkzeuge sind in der Regel spezifisch für eine Programmiersprache ausgelegt. Zu den wichtigsten Eigenschaften von SAST-Werkzeugen zählt etwa die Genauigkeit (falsch-positive oder falsch-negative Warnungen), sowie eine direkte Integration in die Entwicklungsumgebung. Der größte Vorteil von SAST-Werkzeugen ist, wie für statische Analysewerkzeuge allgemein, deren gute Skalierbarkeit, da sie ohne manuelles Zutun fortlaufend und wiederholt auf große Quelltextmengen angewendet werden können um typische Probleme wie Pufferüberläufe zu identifizieren. Ein Nachteil ist, dass SAST-Werkzeuge bei weitem nicht alle möglichen Sicherheitsprobleme aufdecken können.

Ein spezifischer Fall der statischen Sicherheitsanalyse ist die statische, musterbasierte Quelltextanalyse. Allgemein beschreibt die statische, musterbasierte Analyse eine Methode, um Schwachstellen in einer Software durch die Identifikation von bekannten Fehlermus-

tern im Code zu identifizieren [26]. Diese bekannten Muster werden in existierenden Analysebibliotheken mitgeliefert oder vom Entwickler selbst definiert [6].

CogniCrypt CogniCrypt [6] ist ein Eclipse Plugin, welches Entwicklern bei der Erstellung von sicherem Java Quelltext unterstützen soll.

Zum einen ermöglicht $\text{CogniCrypt}_{\text{GEN}}$ eine Generierung von Java Quelltextvorlagen für verschiedene Anwendungsfälle, wie zum Beispiel Datenverschlüsselung. Zum anderen kann mithilfe von $\text{CogniCrypt}_{\text{SAST}}$ bestehender Quelltext musterbasiert auf die korrekte Verwendung von kryptographischen Java Schnittstellen geprüft werden. Hierzu wird der Quelltext auf Basis von Regeln, welche mit der Spezifikationssprache CrySL formuliert werden können, überprüft [27]. So gibt es zum Beispiel existierende Regelsätze für die Bibliotheken Java Cryptographic Architecture (JCA) und BouncyCastle, welche standardmäßig in CogniCrypt für die Codeanalyse ausgewählt werden können. Ein Beispiel für einen einfachen Ausschnitt eines mit CrySL definierten Regelsatzes aus der Java Cryptographic Architecture ist Quelltext 2.1 zu entnehmen [28, 29].

```
SPEC    javax.crypto.Cipher
OBJECTS
    java.lang.String trans;
    byte[] plainText;
    java.security.Key key;
    byte[] cipherText;
EVENTS
    Get: getInstance(trans);
    Init: init(encmode, key);
    doFinal: cipherText = doFinal(plainText);
ORDER
    Get, Init, (doFinal)+
CONSTRAINTS
    encmode in {1,2,3,4};
    alg(trans) in {"AES", ..., "RSA"};
    alg(trans) in {"AES"} => mode(trans) in {"CBC"};
REQUIRES
    generatedKey[key, part(0, "/", trans)];
ENSURES
    encrypted[cipherText, plainText];
```

Quelltext 2.1: CrySL Beispielausschnitt aus dem Regelsatz der Java Cryptographic Architecture für die Klasse Cipher.

Mit `OBJECTS` werden alle Variablen festgelegt, die in der Regel verwendet werden. `EVENTS` legt alle Methoden fest, welche auf einem Objekt, in diesem Beispiel auf einem `Cipher`-Objekt, aufgerufen werden dürfen. `ORDER` bestimmt in welcher Reihenfolge, die Methoden aufgerufen werden dürfen und mit `CONSTRAINTS` werden bestimmte Regeln für die Parameter der Methoden festgelegt, die diese erfüllen müssen. In diesem Beispiel etwa muss `encmode` einer der vier Werte 1, 2, 3 oder 4 sein. `REQUIRES` legt Einschränkungen für andere verwendete Objekte fest, während `ENSURES` Eigenschaften festlegt, die durch die beschriebenen Regeln erfüllt werden.

2.3.5 Statische, Spezifikationsbasierte Sicherheitsanalyse auf Quelltextsicht

Die statische, spezifikationsbasierte Sicherheitsanalyse auf Quelltextsicht ist eine Form der statischen Quelltextanalyse und unterscheidet sich von der statischen musterbasierten Sicherheitsanalyse. Die Literatur zur Unterscheidung dieser beiden Arten von statischen Quelltextanalysen ist nicht eindeutig. Im Folgenden wird deshalb beschrieben, wie spezifikations- und musterbasierte Quelltextanalyse im Rahmen dieser Arbeit zu verstehen und voneinander abzugrenzen sind. Bei der musterbasierten Analyse werden, wie zuvor beschrieben, bekannte und zuvor definierte Fehlermuster erkannt, welche allgemeingültig für spezifische Softwarebibliotheken anzuwenden sind (zum Beispiel auf die korrekte Verwendung von kryptographischen Standard-Schnittstellen [6]). Im Gegensatz dazu, wird bei der spezifikationsbasierten Sicherheitsanalyse auf Quelltextsicht eine Spezifikation von Sicherheitseigenschaften speziell für ein Programm durch einen Entwickler vorgenommen. Ein Beispiel für ein Ansatz zur spezifikationsbasierten Analyse ist etwa JOANA [5]. Mit diesem ist es möglich, spezifikationsbasierte Sicherheitseigenschaften auf einem Programm im Quelltext durch Annotationen zu definieren und diese anschließend spezifisch auszuwerten.

2.4 Quelltextabbildung durch Graphen

In der Informatik werden Graphen dazu verwendet, um Zusammenhänge von Informationen durch die Verbindungen von Knoten über Kanten abzubilden [30]. Im Bereich der Softwareanalyse gibt es die Möglichkeit, die Abhängigkeiten der einzelnen Instruktionen eines Quellcodes sowohl in Bezug auf den Kontrollfluss als auch den Datenfluss in Form eines Graphen darzustellen. Im Folgenden werden in Abschnitt 2.4.1 die für diese Arbeit relevanten Aufrufgraphen und in Abschnitt 2.4.2 die detaillierteren Programm- und Systemabhängigkeitsgraphen beschrieben [31, 5]. Außerdem wird dann in Abschnitt 2.4.3 JOANA zur Erzeugung solcher Graphen vorgestellt. Abschnitt 2.4.4 stellt knapp einen Zusammenhang zwischen den beschriebenen Graphen und dem Architekturmodell her.

2.4.1 Aufrufgraph

Ein Aufrufgraph (CG) eines Programms ist definiert als “ein gerichteter Graph, welcher die Aufruf-Beziehungen zwischen den Prozeduren eines Programms abbildet.” (engl. “The program call graph is a directed graph that represents the calling relationships between the program’s procedures.”) [30]. Bei Aufrufgraphen von Programmiersprachen wie Java stellen die einzelnen Methoden jeweils die Knoten des Graphen dar und die Aufrufe zwischen den Methoden die Kanten. Sie repräsentieren den Kontrollfluss zwischen den einzelnen Methodenaufrufen eines Programms. Für die vorliegende Arbeit werden Aufrufgraphen dazu verwendet, um die Ausgabe einer musterbasierten Quelltextanalyse mit der Eingabe einer Architekturanalyse zu verbinden. Insbesondere soll so ein Zusammenhang zwischen Methoden, in welchen von der Quelltextanalyse ein Verstoß gefunden wird, und Klassen beziehungsweise Methoden, auf denen im Architekturmodell eine Sicherheitseigenschaft definiert wurde, welche durch diesen Verstoß verletzt wird, hergestellt werden.

2.4.2 Programm- und Systemabhängigkeitsgraph

Ein Programmabhängigkeitsgraph (PDG) stellt die Beziehungen der einzelnen Anweisungen eines Programms durch Kontrollfluss- und Datenabhängigkeiten dar [32]. So lassen sich mit Hilfe eines Porgrammabhängigkeitsgraphen Zusammenhänge von Methodenaufrufen, übergebenen Parametern und Rückgabewerten zwischen Anweisungen, Methoden und Klassen herstellen. Ein Beispiel für einen einfachen Programmabhängigkeitsgraphen einer Java-Methode ist in Abbildung 2.1 zu sehen. Als Erweiterung von Programmabhängigkeitsgraphen beschreiben Systemabhängigkeitsgraphen (SDG) die Abhängigkeiten eines gesamten Systems. In diesem Sinne stellen sie eine Zusammensetzung von mehreren Programmabhängigkeitsgraphen über die Grenzen von Elementen der Objektorientierung, wie Methoden und Klassen hinweg, dar. Unter anderem lässt sich aus einem Systemabhängigkeitsgraph der Aufrufgraph bestimmen, da dieser eine echte Teilmenge des SDG darstellt [5].

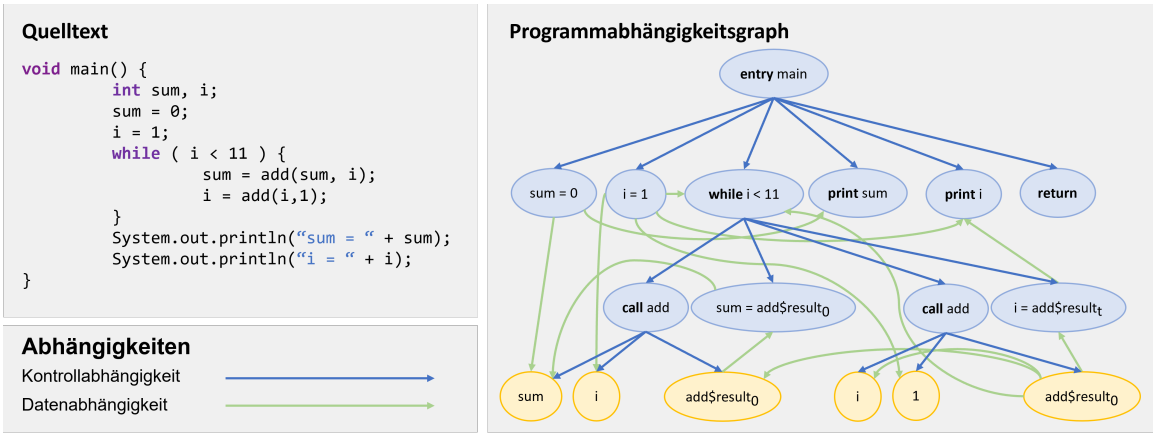

Abbildung 2.1: Beispiel für einen Programmabhängigkeitsgraph eines einfachen Java Programms [33]

2.4.3 SDG Generierung mittels Java Object-sensitive ANAlysis

Java Object-sensitive ANAlysis (JOANA) [5] ist eine Software zur spezifikationsbasierten Sicherheitsanalyse von Java-Programmen auf Basis von Systemabhängigkeitsgraphen. Neben der Quelltextanalyse bietet JOANA auch die Möglichkeit für ein kompiliertes Java Programm den entsprechenden Systemabhängigkeitsgraphen zu generieren und exportieren. So kann der gesamte Kontroll- und Datenfluss eines Programms zur weiteren Analyse extrahiert werden. Zur genauen Analyse des Programms enthält der durch JOANA generierte Systemabhängigkeitsgraph eine Vielzahl von verschiedenen Knoten und Kanten. Dazu gehören etwa Datenabhängigkeitskanten, Parameterstrukturkanten, welche die Datenabhängigkeiten und die Abhängigkeiten von Parametern zwischen den einzelnen Anweisungen beziehungsweise Knoten abbilden, und die Aufrufkanten, welche die Aufrufe zwischen den einzelnen Methoden des Programms darstellen und somit den Aufrufgraphen bilden.

2.4.4 Zusammenhang zwischen Graphen und Architekturmodell

Aufruf- und Systemabhängigkeitesgraphen werden aus dem Quelltext extrahiert und sind dadurch mit diesem eng verbunden. Allerdings ist es ebenfalls möglich einen Zusammenhang zwischen den genannten Graphen und dem Architekturmodell einer Anwendung herzustellen. Langhammer u. a. [34] haben einen Ansatz präsentiert, um mit Hilfe von den genannten Graphen des Quelltextes das Architekturmodell einer Anwendung zu generieren.

3 Verwandte Arbeiten

In der Literatur gibt es bereits erste Ansätze welche darauf abzielen die unter Kapitel 2 beschriebenen Sicherheitsansätze auf Architektur- sowie Quelltextsicht zu kombinieren. Diese Ansätze werden im Folgenden gruppiert und vorgestellt. Ein Ansatz zur Kopplung von statischen Architekturanalysen und musterbasierten Quelltextanalysen wurde in verwandten Arbeiten bisher nicht untersucht.

3.1 Kombination von Quelltext-Sicherheitsanalysen

Wie in Abschnitt 2.3.3 zuvor beschrieben gibt es verschiedene Arten von Sicherheitsanalysen auf Quelltextsicht. Mateo Tudela u. a. [35] kombinieren mehrere Werkzeuge für Sicherheitsanalysen miteinander und evaluieren deren kombinierte falsch-positiv und falsch-negativ sowie richtig-positiv und richtig-negativ Raten. Dazu kombinieren sie die im OWASP Top Ten Projekt gefundenen Ergebnisse von bis zu drei statischen (SAST), dynamischen (DAST) und interaktiven Sicherheitsanalysen (IAST) und führen diese anschließend mit einem Softwarewerkzeug zusammen. Das Programm sammelt für jede Werkzeugkombination die verschiedenen Ergebnisse der Sicherheitsanalysen und bestimmt in verschiedenen Testfällen ob ein falsch-positives oder falsch-negatives oder ein richtig-positives oder richtig-negatives Ergebnis vorliegt. Findet mindestens ein Werkzeug aus der Kombination von zwei oder mehr Werkzeugen einen erwarteten Fehler ist das Gesamtergebnis richtig-positiv. Nur wenn kein Werkzeug ein richtig-positives Ergebnis meldet, wird das Ergebnis als falsch-negativ gezählt. Meldet kein Werkzeug ein falsch-positives Ergebnis, ist das Ergebnis richtig-negativ. Aus diesen Ergebnissen werden dann für jede Kombination der Werkzeuge, um die Effektivität der Kombination zu bestimmen, Metriken, wie Trefferquote (Recall), Genauigkeit (Precision), oder Falsch-Positiv-Rate berechnet. Es zeigt sich, dass eine Kombination von zwei IAST Werkzeugen am effektivsten ist und, dass jede Kombination von mehreren Werkzeugen effektiver ist als die Werkzeuge alleine angewendet.

Im Unterschied zu dieser Arbeit, werden in diesem Ansatz nur die Ergebnisse der verschiedenen Quelltextanalysen gesammelt und zu einem Gesamtergebnis zusammengefasst. Es findet keine direkte Kopplung der Analysen miteinander statt und die Ergebnisse einer Analyse wirken sich nicht auf die Ergebnisse einer anderen aus. Außerdem spielen Sicherheitsanalysen auf Architektursicht in diesem Ansatz keine Rolle.

3.2 Kopplung von Architekturmodellen mit Architekturanalysen

Neben der Kombination von Ergebnissen verschiedener Analysen zu einem Ergebnis, gibt es auch die Möglichkeit das Ergebnis einer Analyse in das Modell zurückzuführen.

Domänenspezifische Modellprüfung (DSMC) übersetzt von einer domänenspezifischen Modellierungssprache (DSML) in die Eingabesprache eines Modellprüfers und übersetzt von diesem gefundene Fehler zurück in das Modell der DSML [36]. So können mit Hilfe von DSMC domänenspezifische Modellierungssprachen und Modellprüfer, deren Sprachen syntaktisch meist sehr unterschiedlich sind, gekoppelt werden. Gerking u. a. wenden DSMC konkret auf der DSML MechatronicUML (MUML) und dem Modellprüfer UPPAAL an. Dabei wird zunächst das MUML Modell in die Eingabesprache von UPPAAL übersetzt. Diese Vorwärtsübersetzung erfolgt in mehreren Schritten in Form einer Modelltransformation. Hierbei wird bei jedem Übersetzungsschritt ein sogenannter Traceability Link generiert. Die Traceability Links stellen die Modelle, die während der Übersetzung generiert werden, in Beziehung und indizieren somit deren semantischen Zusammenhang. Anschließend wird der Modellprüfer UPPAAL automatisch aufgerufen. Findet dieser Verstöße, werden die gefundenen Fehler in einem Schritt zurück in das MUML-Modell überführt. Die Rückübersetzung benötigt nur einen Schritt, da hier die Traceability Links, die im ersten Schritt generiert werden, verwendet werden. Hierzu werden die Traceability Links in umgekehrter Reihenfolge analysiert.

Wie in dieser Arbeit, findet in diesem Ansatz eine Rückkopplung von Ergebnissen der Modellanalyse statt. Allerdings werden keine Sicherheitseigenschaften gekoppelt.

3.3 Kopplung von Architektursicherheitsmodellen mit Quelltextsicherheitsanalysen

Die folgenden Unterabschnitte stellen verschiedene Ansätze zur Kopplung von Architektursicherheitsmodellen mit Quelltextsicherheitsanalysen vor.

3.3.1 Sicherheits-Datenfluss Analyse zwischen Modellen und Quelltext Basierend auf Automatischer Zuordnung

Peldszus u. a. präsentieren einen Ansatz zur Kopplung von Modellen auf Entwurfslevel und Modellen auf Implementatierungslevel [37]. Genauer werden Security Data Flow Diagramme (SecDFD) und Programmmodelle gekoppelt. Der Ansatz soll Unterschiede zwischen Entwurf und Implementierung aufdecken und Sicherheitsverstöße in der Implementierung identifizieren. Hierzu implementieren Peldszus u. a. ein Ecplipse Plugin, das automatisch eine Zuordnung der beiden Modelle generiert, welches jedoch vom Benutzer angepasst werden kann. Die Zuordnung wird iterativ anhand von Namensähnlichkeiten der Elemente der Modelle und heuristischen Regeln erstellt. Der Benutzer kann dann schließlich einzelne Teile der Zuordnung ablehnen, falls diese nicht korrekt generiert wurden und diese manuell selbst erstellen. Durch die Zuordnung können Elemente, welche

im Entwurf definiert wurden, aber nicht in der Implementierung vorhanden sind und im Entwurf fehlende Elemente aufgedeckt werden. Zusätzlich können aus dem SecDFD Sicherheitsinformationen gelesen werden, welche für Sicherheitsanalysen, die den Datenfluss analysieren, benötigt werden und die nicht aus dem Programmmodell ersichtlich sind. Mit diesen Informationen kann mit einem Analysewerkzeug der Datenfluss auf Implementierungssicht auf Sicherheitsverstöße geprüft werden.

Der Ansatz zeigt, dass so Sicherheitsvorgaben aus dem SecDFD für Analysewerkzeuge gewonnen werden können. Im Vergleich zur hier vorliegenden Arbeit wird allerdings nicht evaluiert, inwieweit die Sicherheitsvorgaben aus dem Entwurfsmodell mit der Implementierung verifiziert werden können. Es findet außerdem keine Rückkopplung der Implementierungsanalyse in die Entwurfsanalyse statt.

3.3.2 Überprüfung von Datenflussdiagrammen zur Bedrohungsmodellierung auf Implementierungskonformität und Sicherheit

Abi-Antoun, Wang und Torr [38] überprüfen um Sicherheitseigenschaften erweiterte Datenflussdiagramme aus dem Entwurf und der Implementierung semiautomatisch auf Übereinstimmung. Hierzu werden in diesem Ansatz Reflexionsmodelle erweitert, welche die Konformität von Design und Implementation eines Systems überprüfen. Für die Erstellung des Reflexionsmodells muss ein Benutzer eine Zuordnung zwischen dem Modell des Entwurfs des Systems und dem Modell des Quellcodes erstellen. Auf Basis des Modells des Quellcodes und der erstellten Zuordnung zum Modell des Entwurfs wird dann das Reflexionsmodell generiert. Mit Hilfe dieses Reflexionsmodells werden dann das entworfene Datenflussdiagramm eines Systems und das tatsächliche Datenflussdiagramm, welches aus einem Modell des Quellcodes des Systems generiert wird, auf Unterschiede überprüft. Die Unterschiede und Übereinstimmungen werden zusätzlich gespeichert und bei einer erneuten Analyse mit den neueren Ergebnissen des Vergleichs der Datenflussdiagramme verglichen. Darüber hinaus stellen Abi-Antoun, Wang und Torr [38] eine Sicherheitsanalyse basierend auf vordefinierten Regeln vor, welche ein Datenflussdiagramm auf bekannte Gefahren, wie zum Beispiel auf die vorsätzliche Änderung von Daten durch einen Angreifer, überprüft.

Allerdings werden gefundene Diskrepanzen des Vergleichs in diesem Ansatz nicht in das Datenflussdiagramm des Entwurfs zurückgeführt und somit können deren Auswirkungen nicht durch eine Sicherheitsanalyse geprüft werden.

3.3.3 Sicherstellung der Annahmen des Bedrohungsmodells durch statische Codeanalysen

CARDS [10] ist ein Ansatz auf Architekturebene, welcher Sicherheitsannahmen mithilfe von Quelltextanalysen bestätigen soll, die während der Modellierungsphase getätigt werden. Zunächst werden auf einem Komponentenmodell Sicherheitsannahmen und Einschränkungen definiert, woraus das Bedrohungsmodell resultiert. Anschließend werden diese Annahmen geprüft und eine eventuell notwendige Anpassung dieser Annahmen

durchgeführt. Zu guter Letzt kann eine statische Quelltextanalyse verwendet werden, um zu überprüfen, dass diese Annahmen auch in der Implementierung korrekt erfüllt werden.

Geismann, Haverkamp und Bodden [10] stellen fest, dass dieser Ansatz dabei hilft, Auswirkungen von nicht erfüllten Sicherheitsbedingungen auf die betroffenen Komponenten zu bestimmen. Jedoch ist die Implementierung des Ansatzes aufwendig, da die Kopplung von Bedrohungsmodell und Quelltextanalyse manuell erfolgen muss.

In diesem Ansatz wird nur die Erfüllung der Sicherheitsannahmen überprüft. Anders als in dieser Arbeit findet keine Rückführung der gefundenen Verstöße in das Architekturmodell statt.

3.3.4 Überprüfung der Sicherheitskonformität zwischen Modellen und Code

Tuma u. a. [39] stellen einen semiautomatischen Ansatz vor, um den Datenfluss in einem Entwurfsmodell und Quelltext auf Übereinstimmung zu prüfen. Dabei werden Elemente eines Datenflussdiagrammes, welches um Sicherheitseigenschaften erweitert wurde (SecDFD), den entsprechenden Elementen des Programmmodells der zugehörigen Implementierung des Entwurfs basierend auf heuristischen Regeln und Übereinstimmungen der Namen der Elemente zugeordnet. Diese Zuordnung wird anschließend durch einen Benutzer geprüft und akzeptiert oder entsprechend angepasst. Mit Hilfe dieser Zuordnung werden anschließend die auf dem Datenflussdiagramm definierten Sicherheitseigenschaften automatisch und statisch auf ihre Einhaltung im Quelltext überprüft. Beispielsweise wird überprüft, ob eine geforderte Verschlüsselung im Quelltext durchgeführt wird. Hierfür wird in der Zuordnung von Datenflussdiagramm und Implementierung nach einer Methodensignatur gesucht, welche eine Verschlüsselung durchführt und dem entsprechenden Element zugeordnet wurde, für das die Verschlüsselung gefordert ist. Wird eine solche Methodensignatur gefunden, wird davon ausgegangen, dass die im Datenflussdiagramm geforderte Sicherheitseigenschaft in der Implementierung erfüllt wurde.

Die gefundenen Ergebnisse werden allerdings nicht in das Datenflussdiagramm zurückgeführt, um die Auswirkungen dieser Sicherheitsverstöße auf den Entwurf zu analysieren. Außerdem überprüft dieser Ansatz nur ob geeignete Methoden für die geforderten Sicherheitseigenschaften in der Implementation vorhanden sind. Ob diese Methoden korrekt verwendet werden, wird nicht überprüft.

3.3.5 Ermöglichung des Informationstransfers zwischen Architektur und Quelltext für die Sicherheitsanalyse

In einer vorangehenden Arbeit wurde bereits initial untersucht, wie der Informationstransfer zwischen einer Architektur- und einer Quelltextanalyse zur Kopplung realisiert werden kann [7]. Der Fokus lag dabei auf der Analyse von von gitterbasierten Sicherheitseigenschaften [40]. Diese Eigenschaft wird mittels Sicherheitsklassen modelliert. Zwischen diesen Sicherheitsklassen gibt es einen Informationsfluss. Die Sicherheitsklassen werden durch einen gerichteten, nicht zyklischen Graphen in Abhängigkeit gesetzt. Bei einer Beziehung von Sicherheitsklasse 1 zu Sicherheitsklasse 2, darf Information von Sicherheitsklasse 2 zu Sicherheitsklasse 1 fließen, aber nicht umgekehrt.

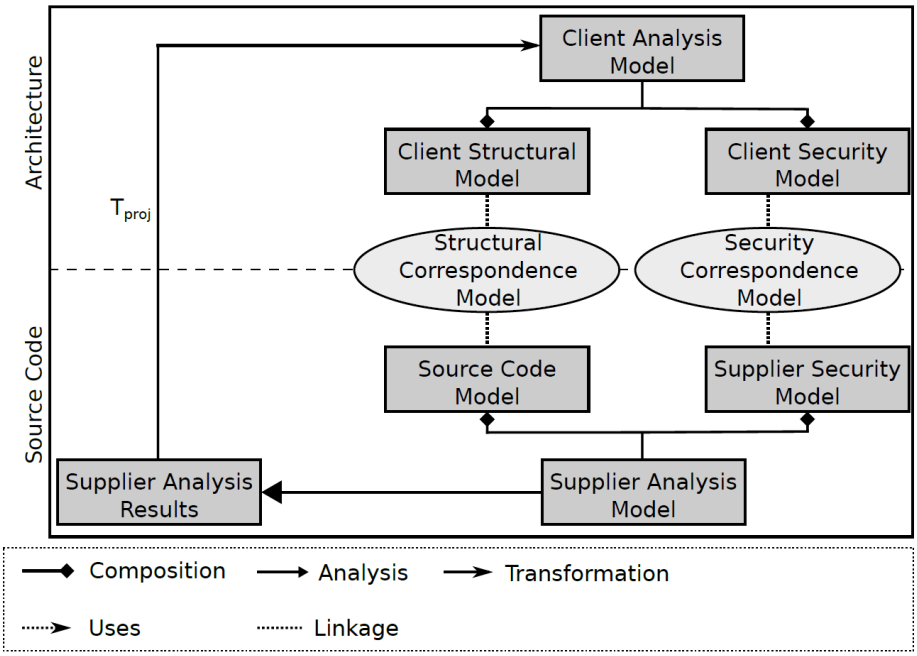

Abbildung 3.1: Schematische Darstellung der Kopplung von Architektursicht und Quellcodesicht [7]. Übertragung der Sicherheitseigenschaften der beiden Sichten erfolgt mittels Sicherheits-Korrespondenzmodell.

Die vorhergehende Arbeit hat dabei den Informationsaustausch zwischen Architektursicht (Client) und Quellcodesicht (Supplier) formell konzipiert. Die folgende Abbildung 3.1 zeigt, wie die Analysen von Sicherheitseigenschaften zwischen den beiden Sichten allgemein gekoppelt werden. Das Analysemodell der Architektur setzt sich aus dem Strukturmodell und dem Sicherheitsmodell der Architektur zusammen. Eine Besonderheit der Arbeit von Häring ist, dass mit Hilfe des Sicherheits-Korrespondenzmodell die Sicherheitsspezifikation der Architektur direkt in Spezifikationen des Quelltextes überführt werden können. Durch Hinzunahme des Struktur-Korrespondenzmodells lässt sich daraus das Modell der Quellcodeanalyse generieren. Die Ergebnisse der Quelltextanalyse werden dann gekoppelt mit dem Architekturanalysemodell.

Zusätzlich zu diesem theoretischen Ansatz implementiert Häring einen konkreten Ansatz mit JOANA für die Quelltextanalyse und Confidentiality4CBSE um Sicherheitsspezifikationen für die Architekturanalyse zu definieren. Die konkrete Implementierung setzt die konzipierte Kopplung von [7] um und anhand dessen wird die Durchführbarkeit des Ansatzes gezeigt.

Im Vergleich zu dem von Häring präsentierten Ansatz werden zur Kopplung der Sicherheitseigenschaften aus der Architektursicht in dieser Arbeit keine Sicherheitsspezifikation für den Quelltext generiert, welche dann mit einem Korrespondenzmodell zurückgeführt werden. In der vorliegenden Arbeit wird hingegen ein Ansatz präsentiert, welcher allgemeingültig musterbasierte Sicherheitsverstöße auf zugehörige Architektursicherheitseigenschaften abbildet, und diese, sofern ein Fehler in Zusammenhang mit einer Architektureigenschaft gebracht werden kann, entfernt.

4 Eigenschaften anwendbarer Sicherheitsanalysen auf Architektur- und Quelltextsicht

Um den Ansatz zur Kopplung der Analysen zu realisieren, müssen die für die Kopplung anwendbaren Sicherheitsanalysen bestimmte Eigenschaften erfüllen. Deshalb werden in diesem Kapitel die zur Kopplung vorausgesetzten Eigenschaften der Architektursicherheitsanalyse und Quelltextsicherheitsanalyse spezifiziert. Dies umfasst die Arten von Sicherheitseigenschaften, die Metamodelle der Struktur, die Metamodelle zur Definition von Sicherheitseigenschaften, und die Ergebnis Metamodelle der Analysen. Eine exakte Spezifikation dieser Metamodelle ist notwendig, da diese die Grundlage für einen Ansatz zur Kopplung der beiden Sichten darstellen. Können spezifische Ansätze für die Architektur- und Quelltextanalyse nicht auf die im Folgenden vorgestellten Metamodelle, welche die vorausgesetzten Eigenschaften beschreiben, abgebildet werden, so ist der in dieser Arbeit konzeptionierte Kopplungsansatz auf diese Ansätze nicht anwendbar. Die Metamodelle sind unabhängig von einem spezifischen Rahmenwerk, wie zum Beispiel Palladio [15, 16] oder einer spezifischen Programmiersprache. In Abschnitt 4.1 werden die Metamodelle der Eigenschaften der Architektursicht näher spezifiziert und in Abschnitt 4.2 die Metamodelle der Eigenschaften der Quelltextsicht.

4.1 Eigenschaften der anwendbaren Architektursicherheitsanalysen

Die Eigenschaften von anwendbaren Architektursicherheitsanalysen teilen sich auf in die Art der für diese Arbeit relevanten Sicherheitseigenschaften sowie die Metamodelle, welche diese Sicherheitseigenschaften beschreiben. In Abschnitt 4.1.1 werden die Sicherheitseigenschaften beschrieben. Abschnitt 4.1.2 gibt eine Übersicht der Modelle zur Sicherheitsanalyse auf Architektursicht. Die darauf folgenden Abschnitte 4.1.3 und 4.1.4 spezifizieren darauf basierend die Eigenschaften der Metamodelle auf Architektursicht zur Festlegung von sowohl Struktur- als auch Sicherheitseigenschaften.

4.1.1 Arten von Architektursicherheitseigenschaften

Die vorliegende Arbeit beschränkt sich auf die Analysen der Sicherheit von Aufrufabhängigkeiten. Eine Aufrufabhängigkeit bezeichnet eine Abhängigkeit zwischen zwei Komponenten durch eine Methode, welche von der einen Komponente durch die andere

Komponente aufgerufen wird. Für eine Abhängigkeit dieser Art kann eine Sicherheitseigenschaft definiert werden [16, 18]. Diese Sicherheitseigenschaft impliziert, dass diese von der aufrufenden Komponente erfüllt werden muss. Zum Beispiel müssen die Daten, welche an die aufgerufene Komponente übertragen werden, korrekt verschlüsselt sein. Die Eigenschaft wäre in diesem Fall also genau dann durch einen Fehler im Quelltext verletzt, wenn die Verschlüsselung fehlerhaft vorgenommen wird.

Die folgende Abbildung 4.1 zeigt exemplarisch eine Aufrufabhängigkeit zwischen zwei Komponenten. Die Assoziation wurde erweitert durch die Sicherheitseigenschaft «`encrypted`». Die `AufrufendeKomponente` muss demnach in der Implementierung alle Aufrufe an die Methode `kritischeOperation` korrekt verschlüsseln, sonst ist die «`encrypted`»-Eigenschaft auf Architektursicht verletzt.

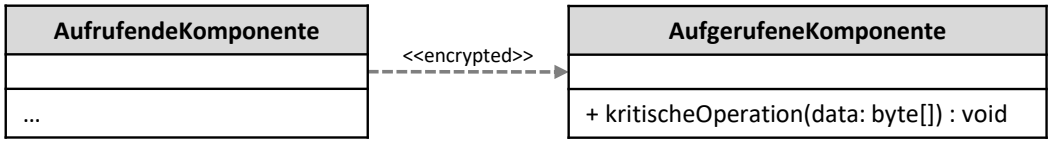

Abbildung 4.1: Architekturmodell einer Aufrufabhängigkeit zwischen zwei Komponenten auf der die Sicherheitseigenschaft «`encrypted`» definiert ist.

Ein Beispiel für die Umsetzung von Aufrufabhängigkeiten mit Sicherheitseigenschaften sind *LinkingResources* mit dem Stereotypen «encrypted» im Ansatz Confidentiality4CBSE [16] oder *Secure Dependencies* in UmlSec [18].

4.1.2 Übersicht der Modelle zur Sicherheitsanalyse auf Architektursicht

Die Sicherheitsanalyse der Architektur setzt sich aus dem Modell der Struktur, dem Modell der Sicherheitseigenschaften und dem dazugehörigen Ergebnis zusammen. Abbildung 4.2 gibt deshalb einen Überblick über die Modelle und Metamodelle für die Sicherheitsanalyse auf Architektursicht. Diese stellen die notwendigen Abstraktionen dar, welche für die Architektursicherheitsanalyse benötigt werden. Die Metamodelle beschreiben die Elemente die notwendig sind, um die Modelle und Ergebnisse einer spezifischen Sicherheitsanalyse für eine Architektur zu realisieren.

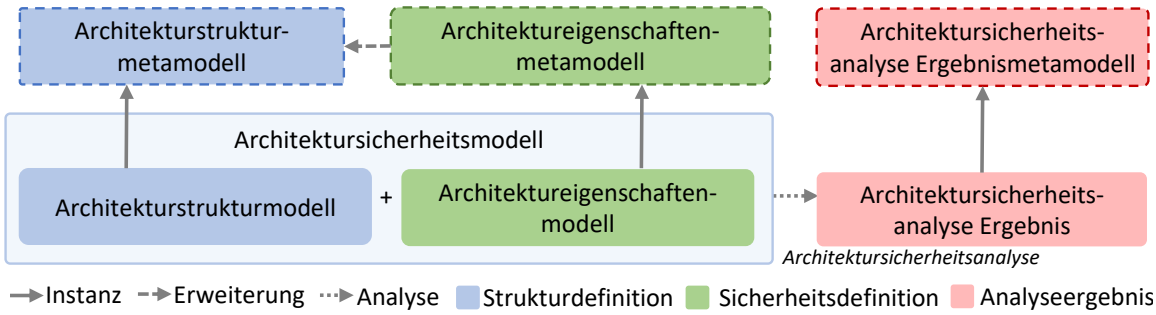

Abbildung 4.2: Übersicht der Metamodelle und Modelle auf Architektursicht zur Definition von Struktur und Sicherheitseigenschaften, sowie für die Ergebnisse der Analyse (Abbildung angelehnt an Häring [7])

Das Architekturstrukturmodell stellt eine Instanz des Architekturstrukturmetamodells dar. Das Metamodell definiert die möglichen Strukturelemente der Architektur eines Programmes (zum Beispiel Komponenten oder Schnittstellen). Das Architekturstrukturmodell instanziiert dann die verschiedenen Elemente aus dem Architekturstrukturmetamodell spezifisch für ein Programm. Das Architektureigenschaftenmetamodell erweitert das Architekturstrukturmetamodell um Sicherheitseigenschaften. Die Architektursicherheitsanalyse erfolgt auf Basis des Architektursicherheitsmodells, welches sich aus einer Instanz eines Architekturstrukturmodells und dem Architektureigenschaftenmodell zusammensetzt. Die Analyse liefert ein Architektursicherheitsanalyse Ergebnis, gemäß des Architektursicherheitsanalyse Ergebnismetamodells, von dem in dieser Arbeit keine spezifischen Eigenschaft gefordert werden. Die Metamodelle zur Struktur- und Sicherheitsdefinition werden im folgenden Abschnitt 4.1.3 und Abschnitt 4.1.4 im Detail spezifiziert.

4.1.3 Architekturstrukturmetamodell

Das Architekturstrukturmetamodell beschreibt die Struktur und die Strukturelemente, mit denen ein Architekturstrukturmodell definiert wird. Die folgende Abbildung 4.3 zeigt die für den in dieser Arbeit präsentierten Ansatz notwendigen Elemente der Architekturdefinition.

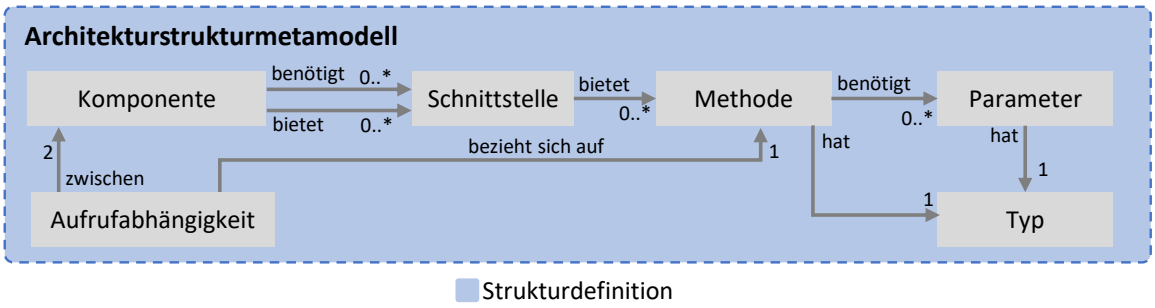

Abbildung 4.3: Das Architekturstrukturmetamodell, welches die notwendigen Strukturelemente eines Architekturstrukturmodells beschreibt (angelehnt an Reussner u. a. [15]).

Komponenten setzen Schnittstellen um, welche Funktionalitäten in Form von Methoden beschreiben. Methoden können Parameter eines bestimmten Typs entgegennehmen und liefern selbst ein Ergebnis eines Typs zurück. Weiterhin können Komponenten auch Schnittstellen benötigen, um die eigene Funktionalität umsetzen zu können. Das Metamodell der Architekturstruktur ist äquivalent zu dem von Häring [7] beschriebenen Ansatz. Eine Voraussetzung an die spezifische Umsetzung des Architekturstrukturmetamodells ist, dass es möglich ist, die Komponenten und Methoden des Architekturstrukturmetamodells eindeutig auf das Quelltextmetamodell zu überführen. Ohne die Existenz dieser Eigenschaft kann eine Kopplung der beiden Sichten nicht durchgeführt werden, da nicht entschieden werden kann, auf welche Elemente der Architektur sich ein Fehler an einer bestimmten Stelle im Quelltext bezieht. Weiterhin ist es für den in dieser Arbeit präsentierten Kopplungsansatz notwendig, dass die Strukturelemente eines spezifischen Rahmenwerks

zur Modellierung der Architektur auf eine Instanz des hier vorgestellten Metamodells abgebildet werden können. Die Erweiterung des Architekturmetamodells zur Definition der Sicherheit von Aufrufabhängigkeiten wird im folgenden Abschnitt 4.1.4 beschrieben.

4.1.4 Architektureigenschaftenmetamodell

Die Definition von Sicherheitseigenschaften auf Architektursicht erfolgt gemäß eines Architektureigenschaftenmetamodells. Dieses ergänzt die Strukturelemente des zugrundeliegenden Architekturstrukturmetamodells um Sicherheitseigenschaften. Eine Übersicht über das Architektureigenschaftenmetamodell ist in Abbildung 4.4 dargestellt. Dieses beschränkt sich auf die auf einer Aufrufabhängigkeit definierten Sicherheitseigenschaften.

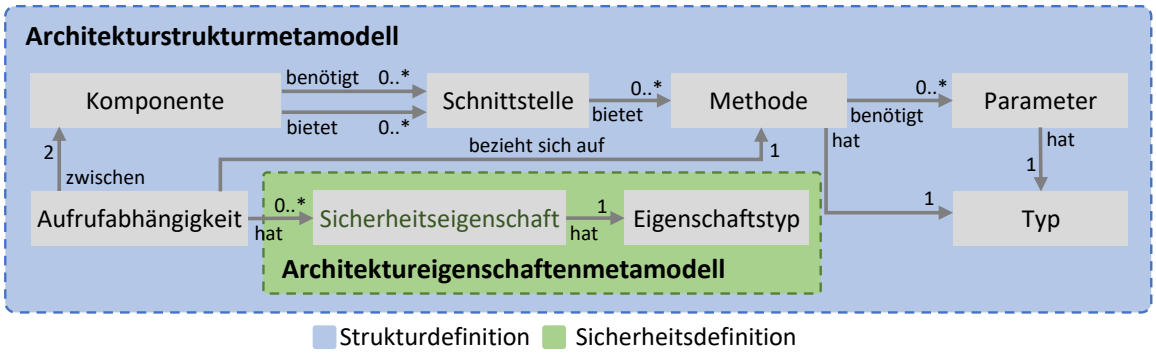

Abbildung 4.4: Architektursicherheitsmetamodell, welches das Architekturmetamodell um Sicherheitseigenschaften für Aufrufabhängigkeiten erweitert.

Eine Aufrufabhängigkeit besteht zwischen zwei Komponenten. Sie bezieht sich auf den Aufruf einer Methode von einer Komponente durch eine andere Komponente. Auf einer solchen Abhängigkeit können mehrere Sicherheitseigenschaften mit unterschiedlichen Eigenschaftstypen definiert sein.

4.2 Eigenschaften der anwendbaren Quelltextsicherheitsanalysen

Im Abschnitt 4.2.1 werden die geforderten Eigenschaften der für diese Arbeit relevanten Arten von Quelltextsicherheitsfehlern beschrieben. Zur besseren Übersicht der verschiedenen Modelle der Sicherheitsanalyse auf Quelltextsicht werden diese zunächst in Abschnitt 4.2.2 eingeführt. In Abschnitt 4.2.3 wird auf das Quelltextmetamodell eingegangen. Abschnitt 4.2.4 führt das Metamodell für das Ergebnis der Quelltextanalyse aus, welches kritisch für die Kopplung der Analysen ist.

4.2.1 Eigenschaften von Quelltextsicherheitsfehlern

Wie in Abschnitt 2.3.4 beschrieben überprüfen musterbasierte Quelltextanalysen den Quelltext auf bekannte Fehlermuster. Wird ein Fehlermuster identifiziert liefert die Quelltextanalyse einen Quelltextsicherheitsfehler als Ergebnis. Ein Fehler bezieht sich auf eine

spezifische Stelle im Quelltext. Eine Stelle muss dabei die exakte Methode und Klasse sowie die Quelltextzeile umfassen, in welcher der Fehler auftritt. Diese Informationen sind erforderlich, um festzustellen, ob der Fehler an einer Stelle auftritt welcher zu einer Verletzung einer Sicherheitseigenschaft auf Architektursicht führt.

4.2.2 Übersicht der Modelle zur Sicherheitsanalyse auf Quelltextsicht

Abbildung 4.5 gibt einen Überblick über die Modelle und Metamodelle für die Sicherheitsanalyse auf Quelltextsicht. Diese stellen die notwendigen Abstraktionen dar, welche für die Quelltextsicherheitsanalyse benötigt werden. Die Metamodelle beschreiben das abstrakte Format konkreter Modelle einen spezifischen Sicherheitsanalyse. Die Metamodelle werden von Abschnitt 4.2.3 bis Abschnitt 4.2.4 im Detail spezifiziert.

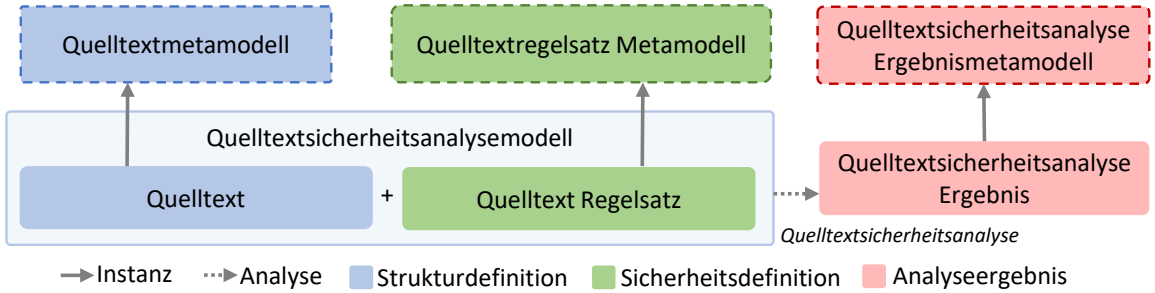

Abbildung 4.5: Übersicht der Metamodelle und Modelle auf Quelltextsicht zur Definition von Struktur und Regeln sowie für die Ergebnisse der Sicherheitsanalyse.

Der Quelltext stellt eine Instanz des Quelltextmetamodells dar. Das Metamodell definiert die möglichen Elemente des Quelltextes eines Programms (z.B. Klasse, Schnittstelle). Das Quelltextregelsatz Metamodell definiert das Format mit dem Regeln für die Quelltextsicherheitsanalyse definiert werden. Mit Hilfe dieses Metamodells lässt sich ein konkreter Regelsatz definieren, zum Beispiel für die Verwendung einer Kryptographie Softwarebibliothek. Das Metamodell für den Regelsatz einer statischen, musterbasierten Quelltextanalyse ist in der Regel spezifisch für die zugehörige Quelltextsicherheitsanalyse [6, 27, 17]. Dabei ist das Quelltextregelsatz Metamodell nicht relevant für die Umsetzung für die Kopplung der Sicherheitseigenschaften auf Architektursicht. Demnach stellt der präsentierte Kopplungsansatz also keine Anforderungen an die Regeldefinition. Die Quelltextsicherheitsanalyse wird auf den Quelltext angewendet und findet auf Basis eines Regelsatzes musterbasiert Fehler. Der Regelsatz wird für die Verwendung bestimmter Bibliotheken definiert, etwa durch einen Entwickler. Werden diese Bibliotheken im Quelltext verwendet, kann deren falsche Verwendung mit Hilfe der Regeln erkannt werden. Die Ergebnisse der Quelltextsicherheitsanalyse folgen einem Quelltextsicherheitsanalyse Ergebnismetamodell.

4.2.3 Quelltextmetamodell

Das Quelltextmetamodell, welches in Abbildung 4.6 dargestellt ist, beschreibt die Elemente einer objektorientierten Programmiersprache. Es handelt sich hierbei nicht um eine

vollständig Darstellung, sondern um die notwendige Teilmenge von Elementen, welche für die Umsetzung des Kopplungsansatzes verwendet werden.

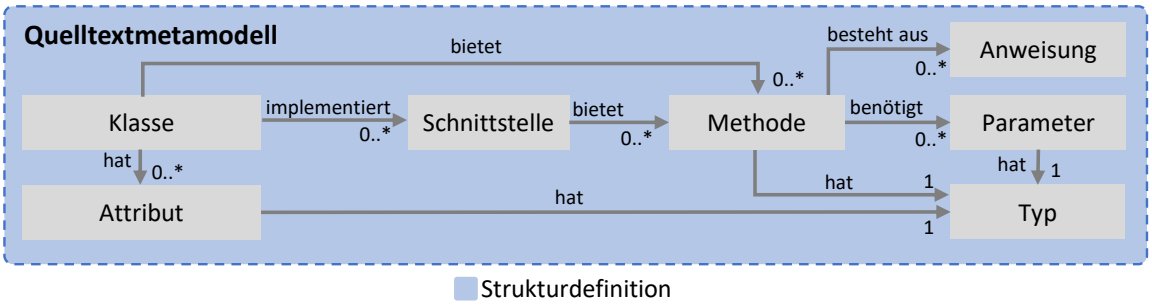

Abbildung 4.6: Metamodell des Quelltextes basierend auf Java.

Äquivalent zum Architekturmetamodell gibt es Klassen, welche Schnittstellen implementieren. Diese Schnittstellen legen Methoden fest, die durch Klassen implementiert werden. Methoden nehmen Parameter eines Typs entgegen und geben ein Ergebnis eines Typs zurück. Eine Klasse kann außerdem diverse Attribute haben, die jeweils einem spezifischen Typ zugeordnet werden. Das Metamodell des Quelltextes erweitert das von Häring [7] beschriebene Quelltextmetamodell um Anweisungen. Zur Realisierung des Kopplungsansatzes dieser Arbeit ist es notwendig, dass eine Klasse und Methode des Quelltextmetamodells eindeutig auf eine Komponente und Methode des Architekturmetamodells abgebildet werden können. Andernfalls ist nicht eindeutig, welcher Stelle in der Architektur sich ein Fehler im Quelltext zuordnen lässt.

4.2.4 Quelltextanalyse Ergebnismetamodell

In der folgenden Abbildung 4.7 ist beschrieben, wie sich das Metamodell des Ergebnisses der Quelltextsicherheitsanalyse zusammensetzt.

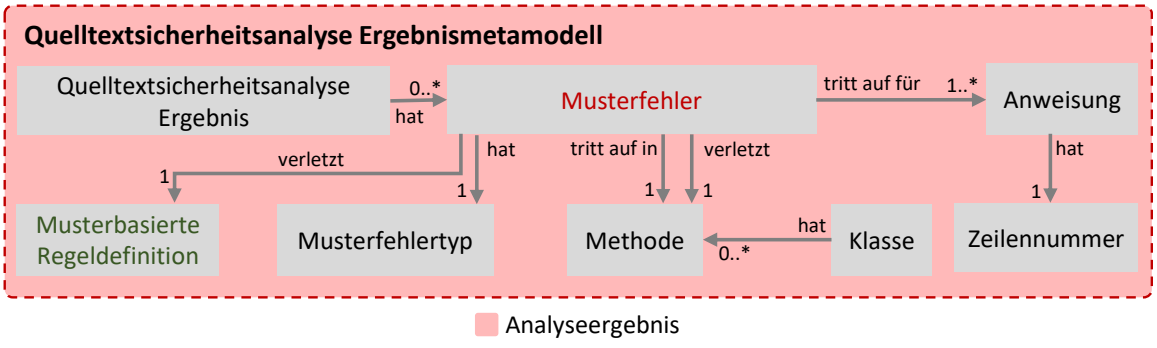

Abbildung 4.7: Metamodell des Ergebnisses einer musterbasierten Quelltextsicherheitsanalyse.

Das Ergebnis der Quelltextsicherheitsanalyse muss für die Kopplung eine Anzahl von Musterfehlern zurückgeben. Ein Musterfehler entstammt einer musterbasierte Regeldefinition, welche sich auf die korrekte Verwendung einer Methode einer Klasse bezieht.

Demzufolge verletzt ein Musterfehler eine Methode in einer Klasse. Zusätzlich bezieht sich ein Musterfehler auf eine oder mehrere Anweisungen. Eine Anweisung repräsentiert hierbei eine Zeile im Quelltext. Außerdem wird die Stelle, an der ein Musterfehler auftritt durch die dazugehörige Klasse und Methode im Quelltext beschrieben.

5 Konzept zur Kopplung von statischen Architekturanalysen und statischen, musterbasierten Quelltextanalysen

Auf Basis der im vorherigen Kapitel 4 eingeführten Metamodelle der anwendbaren Analysen soll im Folgenden ein Konzept zur Kopplung der beiden Sichten beschrieben werden. Im Abschnitt 5.1 wird zunächst der schematische Ablauf der Kopplung veranschaulicht. Anschließend wird der in dieser Arbeit realisierte Ansatz zur Kopplung in Abschnitt 5.3 konzeptionell beschrieben. Abschließend wird in Abschnitt 5.4 eine Gesamtübersicht des Kopplungsablaufs präsentiert.

5.1 Schematischer Ablauf der Kopplung

Das Architektursicherheitsmodell trifft Annahmen über die korrekte Umsetzung von Sicherheitseigenschaften im Quelltext einer Software. Der Quelltext soll diese angenommenen Sicherheitseigenschaften korrekt umsetzen. Bei der Programmierung einer Anwendung können festgelegte Sicherheitseigenschaften der Architektur durch einen Entwickler falsch umgesetzt werden. Eine falsche Umsetzung kann sich in Form von Musterfehlern im Quelltext äußern. Diese können mit der Quelltextsicherheitsanalyse automatisch erkannt werden. Die Kopplung hat zum Ziel, Sicherheitseigenschaften der Architektur auf Basis von Musterfehlern im Quelltext als falsch angenommen zu identifizieren. Um die Auswirkungen von verletzten Sicherheitseigenschaften zu bestimmen, können diese aus dem Architektursicherheitsmodell entfernt werden. Dadurch können durch die Architektursicherheitsanalyse zusätzliche Fehler erkannt werden, welche durch die fehlenden Sicherheitseigenschaften bedingt sind. In Abbildung 5.1 ist der Ablauf zur Kopplung der beiden Sichten schematisch dargestellt.
Auf Architektursicht ist das Architektursicherheitsmodell gemäß der in Kapitel 4 beschrieben Metamodelle definiert. Auf Quelltextsicht stellt der Quelltext die konkrete Implementation des Architektursicherheitsmodells dar. Wird auf diesem Quelltext eine musterbasierte Sicherheitsanalyse ausgeführt, liefert diese mit Hilfe des Regelsatzes im Quelltext auftretende Musterfehler. Es ist möglich, dass ein Musterfehler darauf schließen lässt, dass eine aus der Architektursicht angenommenen Sicherheitseigenschaft nicht korrekt umgesetzt wurde. Der Kopplungsansatz soll nun einen Zusammenhang zwischen den einzelnen Sicherheitseigenschaften des Architektursicherheitmodells und den Musterfehlern des Quelltextes herstellen. Es soll also für jedes Paar von definierter Sicherheitseigenschaft und auftretenden Musterfehler bestimmt werden, ob die Sicherheitseigenschaft durch den vorliegenden Musterfehler verletzt wird. Ist dies der Fall, soll eine Rückkopplung

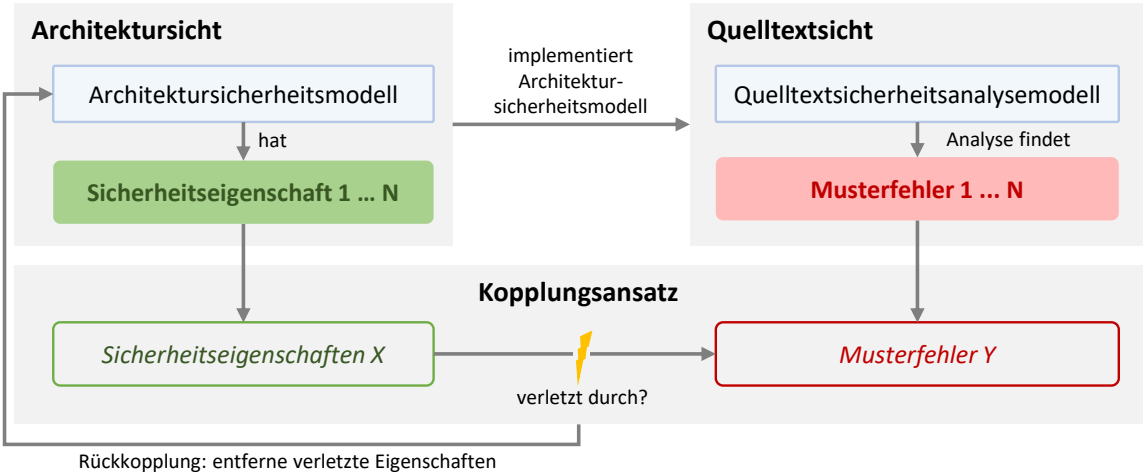

Abbildung 5.1: Darstellung des schematischen Ablaufs der Kopplung.

der vorliegenden Verletzung in das Architektursicherheitsmodell stattfinden. Das heißt, die entsprechenden Sicherheitseigenschaften sollen aus dem Architekturmodell entfernt werden, da diese durch die fehlerhafte Umsetzung im Quelltext nicht mehr gelten.

5.2 Verletzung der Gültigkeit von Sicherheitseigenschaften

Der schematische Kopplungsansatz zeigt, dass eine Verbindung zwischen den Fehlern im Quelltext und den auf dem Architekturmodell definierten Sicherheitseigenschaften hergestellt werden muss. Deshalb wird zunächst formalisiert, welche Bedingungen gelten müssen, damit eine Sicherheitseigenschaft als verletzt gilt und diese somit ihre Gültigkeit im Architekturmodell verliert.
Wie in Abschnitt 4.1.1 beschrieben, werden in der vorliegenden Arbeit Sicherheitseigenschaften, welche auf Aufrufabhängigkeiten definiert sind, betrachtet. Damit eine Sicherheitseigenschaft als verletzt gilt, müssen die folgenden zwei Bedingungen erfüllt sein. Zum einen muss sich der im Quelltext gefundene Fehler auf einen Methodenaufruf auswirken, für den im Architekturmodell eine Sicherheitseigenschaft definiert wurde. Zum anderen müssen Fehler- und Sicherheitseigenschaftstyp korrespondieren. Das heißt, dass für den Typ des Quelltextfehlers definiert sein muss, dass dieser den entsprechenden Sicherheitseigenschaftstyp verletzt. Für den in dieser Arbeit präsentierten Kopplungsansatz wird diese Verbindung explizit durch den Entwickler festgelegt.

Um zu entscheiden, ob sich ein Musterfehler möglicherweise auf einen Methodenaufruf mit Sicherheitseigenschaft auswirkt, gibt es wiederum zwei Möglichkeiten:

- **Verletzung auf Basis des Methoden-Kontrollfluss:** Tritt ein Fehler in einer Methode auf, welche selbst einen Methodenaufruf durchführt auf dem eine Sicherheitseigenschaft definiert wurde, so existiert ein relevanter Zusammenhang zwischen dem Musterfehler und der Sicherheitseigenschaft.

- **Verletzung auf Basis des Datenfluss:** Manipuliert eine Anweisung mit Musterfehler ein Datum, welches an eine Methode mit Sicherheitseigenschaft übergeben

wird, existiert ein relevanter Zusammenhang zwischen dem Musterfehler und der Sicherheitseigenschaft.

Damit eine Sicherheitseigenschaft als verletzt gilt, muss zusätzlich der Typ des Fehlers mit dem Sicherheitseigenschafttyp korrespondieren. Hierzu muss eine Abbildung zwischen den verschiedenen Sicherheitseigenschaftstypen und Musterfehlertypen definiert werden. Diese Korrespondenz ist in der folgenden Abbildung 5.2 dargestellt. Hierbei wird jeder Musterfehlertyp auf eine Menge von Sicherheitseigenschaftstypen abgebildet. Besteht also eine Abbildung zwischen einem Sicherheitseigenschafts- und einem Musterfehlertyp, dann kann eine Sicherheitseigenschaft mit dem entsprechenden Typ durch einen Musterfehler mit dem korrespondierenden Typ verletzt werden. Mehrere Musterfehlertypen können einen Sicherheitseigenschaftstyp verletzen. Jeder verletzende Musterfehlertyp stellt eine alleinige Verletzung des dazugehörigen Sicherheitseigenschaftstyps dar.

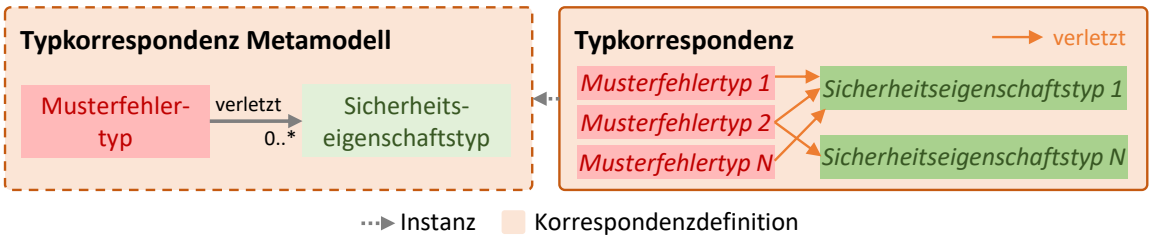

Abbildung 5.2: Darstellung des Metamodells der Typkorrespondenz und einer beispielhaften Instanz.

Das folgende Beispiel in Quelltext 5.1 veranschaulicht die beiden Bedingungen. Der Aufruf der Methode `kritischeOperation` hat dabei den Sicherheitseigenschaftstyp «`encrypted`». Weiterhin sei definiert, dass Musterfehler im Zusammenhang mit Aufrufen der Klasse `Cipher` den Musterfehlertyp *encrypted* besitzt. Für die Typkorrespondenz sei definiert, dass der Musterfehlertyp *encrypted* den Sicherheitseigenschaftstyp «`encrypted`» verletzt.

```
1 public class KlasseA {
2   public void operation(byte[] text, byte[] schluessel) {
3     byte[] geheim = Cipher.encrypt(text, schluessel, "ABC");  // Musterfehler
4     klasseB.kritischeOperation(new byte[] { 0x00, 0x01, 0x02 });
5   }
6 }
```

Quelltext 5.1: Beispielmethode mit einem Musterfehler und Aufruf einer Methode mit Sicherheitseigenschaft.

Betrachtet man das Beispiel auf Basis des Methoden-Kontrollflusses, so tritt der obige Fehler in der Methode auf, welche die Methode mit Sicherheitseigenschaft aufruft. Ebenfalls kann geschlossen werden, dass der Typ des Musterfehlers eine Verletzung der Eigenschaft gemäß der Typkorrespondenz darstellt. Demnach wäre die Sicherheitseigenschaft verletzt. Bei Betrachtung des Datenflusses wird jedoch deutlich, dass die Variable `geheim` niemals an die `kritischeOperation` übergeben wird. Demnach würde es bei dieser Herangehensweise zu keiner Verletzung der Sicherheitseigenschaft kommen.

5.3 Kopplung von Architektureigenschaften und Musterfehlern mittels Graphen

Der vorherige Abschnitt 5.2, hat verdeutlicht, dass mittels des Methoden-Kontroll- und Datenflusses eines Programms eine Kopplung von Sicherheitseigenschaften und Musterfehlern prinzipiell realisierbar ist. Wie bereits in Abschnitt 2.4 beschrieben sind Aufrufgraphen beziehungsweise Systemabhängigkeitsgraphen dazu geeignet den Kontroll- sowie den Datenfluss eines Programms abzubilden. In den folgenden Abschnitten soll die Kopplung auf Basis dieser Graphen beschrieben werden. Abschnitt 5.3.1 präsentiert eine Variante auf Basis des Aufrufgraphen. Abschnitt 5.3.2 führt eine Variante auf Basis des Systemabhängigkeitsgraphen ein.

5.3.1 Ansatz auf Basis des Aufrufgraphen

Der Ansatz auf Basis des Aufrufgraphen verwendet lediglich die Informationen darüber, welche Methoden welche Methodenaufrufe durchführen. In Abschnitt 5.3.1.1 wird zunächst das Metamodell eines Aufrufgraphen inklusive Sicherheitsmerkmalen eingeführt. Der Abschnitt 5.3.1.2 beschreibt dann, wie Musterfehler und Sicherheitseigenschaften dem Aufrufgraphen zugewiesen werden, um anschließend verletzte Sicherheitseigenschaften zu identifizieren.

5.3.1.1 Metamodell des Aufrufgraphen mit Sicherheitsmerkmalen

Die Abbildung 5.3 gibt einen Überblick über das Metamodell eines Aufrufgraphen, erweitert um Sicherheitsmerkmale in Form von Musterfehlern aus der Quelltextsicht und Sicherheitseigenschaften aus der Architektursicht. Für ein gegebenes Programm ist es möglich, den Aufrufgraph eines Programms auf Basis des Quelltextes statisch zu extrahieren (siehe Abschnitt 2.4.1). Die Zuweisung der Sicherheitsmerkmale ist Teil der in dieser Arbeit konzipierten Kopplung.

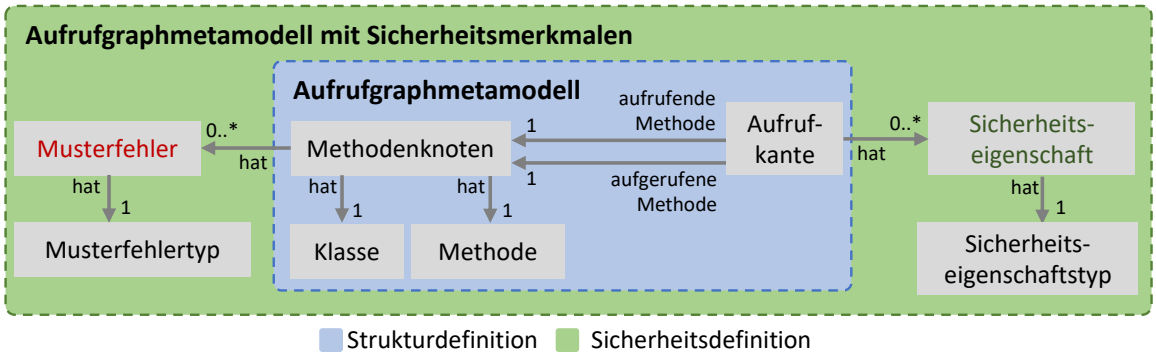

Abbildung 5.3: Metamodell eines Aufrufgraphen erweitert um Sicherheitsmerkmale.

Ein Aufrufgraph ist ein gerichteter Graph. Er enthält Knoten, welche die einzelnen Methoden eines Programms darstellen. Ein Knoten beinhaltet die Klasse der Methode, die der Knoten darstellt und deren Methodensignatur. Die gerichteten Kanten des Graphen

verbinden jeweils zwei Knoten miteinander und stellen so einen Methodenaufruf dar. Hierbei repräsentiert der Startknoten die aufrufende Methode und der Endknoten die aufgerufene Methode. Für den in dieser Arbeit vorgestellten Ansatz wird der Aufrufgraph um Sicherheitsmerkmale erweitert. Dazu zählen die Sicherheitseigenschaften des Architekturmodells, welche auf Kanten als Attribut definiert werden, und die Musterfehler, welche Knoten als Attribut zugeordnet werden. Sowohl Sicherheitseigenschaften als auch Musterfehler besitzen einen Typ. Eine Aufrufkante mit Sicherheitseigenschaft repräsentiert im Aufrufgraph also genau die in Abschnitt 4.1.4 eingeführte Aufrufabhängigkeit mit Sicherheitseigenschaft.

5.3.1.2 Realisierung der Kopplung mittels Aufrufgraph mit Sicherheitsmerkmalen

Die Kopplung auf Basis des Aufrufgraphen erfolgt in drei Schritten. Zunächst werden die Sicherheitseigenschaften den entsprechenden Kanten im Graphen zugewiesen. Anschließend werden die Musterfehler den zugehörigen Methodenknoten zugeordnet. Auf Basis des Aufrufgraphen mit zugewiesenen Sicherheitsmerkmalen können dann die durch Fehler verletzten Sicherheitseigenschaften bestimmt werden. Diese Schritte werden im Folgenden näher ausgeführt.

Die Sicherheitseigenschaften folgen aus der Definition des Architekturmodells. Der Aufrufgraph wird allerdings auf Basis des Quelltextes generiert. Eine grundlegende Voraussetzung zur Realisierung des Kopplungsansatzes ist deshalb, dass sich die Komponenten der Architektursicht eindeutig auf Klassen der Quelltextsicht abbilden lassen. Ebenso müssen die Methoden der Komponenten eindeutig auf die entsprechenden Methoden der Klassen abgebildet werden können. Im Rahmen der vorliegenden Arbeit wird davon ausgegangen, dass diese Zuweisung ausschließlich auf Basis der Kombination von Klassen- und Methodennamen möglich ist. Dadurch lässt sich jede Methode einer Komponente auf Architektursicht einer Methode einer Klasse auf Quelltextsicht zuordnen.

Zunächst werden die Sicherheitseigenschaften aus dem Architekturmodell extrahiert. Anschließend wird der Aufrufgraph auf Basis des implementierten Quelltext generiert. Die Zuweisung der Sicherheitseigenschaften erfolgt dann, indem für jede Kante geprüft wird, ob diese einen Aufruf gemäß einer definierten Aufrufabhängigkeit mit Sicherheitseigenschaft darstellt. Ist dies der Fall, so wird die Sicherheitseigenschaft der entsprechenden Kante zugewiesen.

Für die Zuweisung von Musterfehlern gilt, dass deren Auftreten im Code ohnehin eindeutig einer Klasse und Methode zugeordnet ist, wie in Abschnitt 4.2.4 gefordert. Die Zuweisung der Musterfehler zu Knoten des Aufrufgraphen erfolgt demnach, indem sie dem Knoten zugeordnet werden, der die Methode repräsentiert, in der der Fehler auftritt.

Wurden alle Sicherheitseigenschaften und Musterfehler zugewiesen, können die verletzten Sicherheitseigenschaften bestimmt werden. Für jede Kante mit Sicherheitseigenschaft wird geprüft, ob der Ausgangsknoten einen relevanten Musterfehler besitzt. Relevant ist dieser genau dann, wenn er die Sicherheitseigenschaft der Kante im Sinne des in Abschnitt 5.2 eingeführten Typkorrespondenzmodells verletzt. Für den vorliegenden Ansatz wird also nur der Ausgangsknoten der Kante betrachtet, um zu bestimmen, ob die Sicherheitseigenschaft verletzt ist. Es wird insbesondere nicht nach Musterfehlern in weiteren Vorgängerknoten einer Kante gesucht.

Die verletzten Sicherheitseigenschaften können können für die Rückkopplung aus dem Architektursicherheitsmodell entfernt werden. Dafür ist es notwendig, dass die zugewiesene Sicherheitseigenschaft eine eindeutige Referenz, zum Beispiel einen numerischen Identifikator, auf den eigenen Ursprung im Architektursicherheitsmodell besitzt.

Der Aufrufgraph für das Beispiel aus Quelltext 5.1 ist in der Abbildung 5.4 dargestellt. Die Grafik zeigt einen Ausschnitt eines Aufrufgraphen mit zwei Knoten. Diese umfasst die Methode `KlasseA.operation` mit einer Aufrufkante zum Knoten `KlasseB.kritischeOperation`. Der Musterfehler wurde dem entsprechenden Methodenknoten zugewiesen. Die Sicherheitseigenschaft wurde der Kante, welche den entsprechenden Methodenaufruf repräsentiert, zugeordnet. Da diese Kante einen Knoten als Startknoten hat, der einen Musterfehler enthält, welcher die Sicherheitseigenschaft der Kante verletzt, gilt diese Eigenschaft als verletzt.

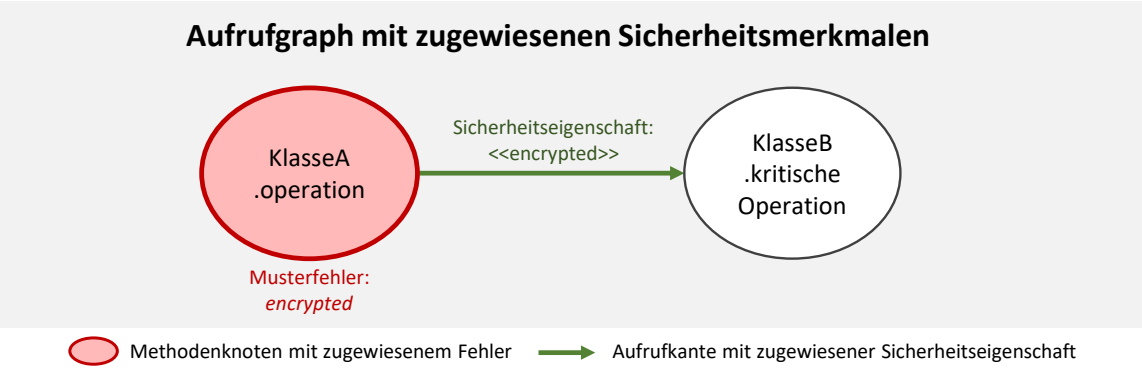

Abbildung 5.4: Ausschnitt eines Aufrufgraphen mit zwei Knoten, einer Sicherheitseigenschaft und einem Musterfehler.

5.3.2 Ansatz auf Basis des Systemabhängigkeitsgraphen

Der Ansatz auf Basis des Systemabhängigkeitsgraphen (SDG) beinhaltet eine genaue Analyse des Datenflusses innerhalb und zwischen den einzelnen Methoden eines Programmes. Dies soll ein genaueres Ermitteln der verletzten Sicherheitseigenschaften ermöglichen. In Abschnitt 5.3.2.1 werden zunächst das Metamodell und die Eigenschaften eines SDG, erweitert um Sicherheitsmerkmale, für den Kopplungsansatz präsentiert. Der Abschnitt 5.3.2.2 beschreibt die Realisierung der Kopplung mit Hilfe des SDG.

5.3.2.1 Metamodell und Eigenschaften des SDG mit Sicherheitsmerkmalen

Die Abbildung 5.5 stellt das Metamodell eines Systemabhängigkeitsgraphen dar, welcher zusätzlich um Sicherheitsmerkmale erweitert wurde. Aus der Quelltextsicht werden Musterfehler auf die Anweisungen im SDG übertragen, die sie betreffen. Ebenso werden Architektursicherheitseigenschaften auf Anweisungen übertragen.

Ein Systemabhängigkeitsgraph ist ein gerichteter Graph. Er enthält Knoten, welche die Anweisungen eines Programms darstellen. Für die vorliegende Arbeit sind sowohl die Methodenaufruf- als auch Datenzuweisungsknoten von Relevanz. Die verschiedenen

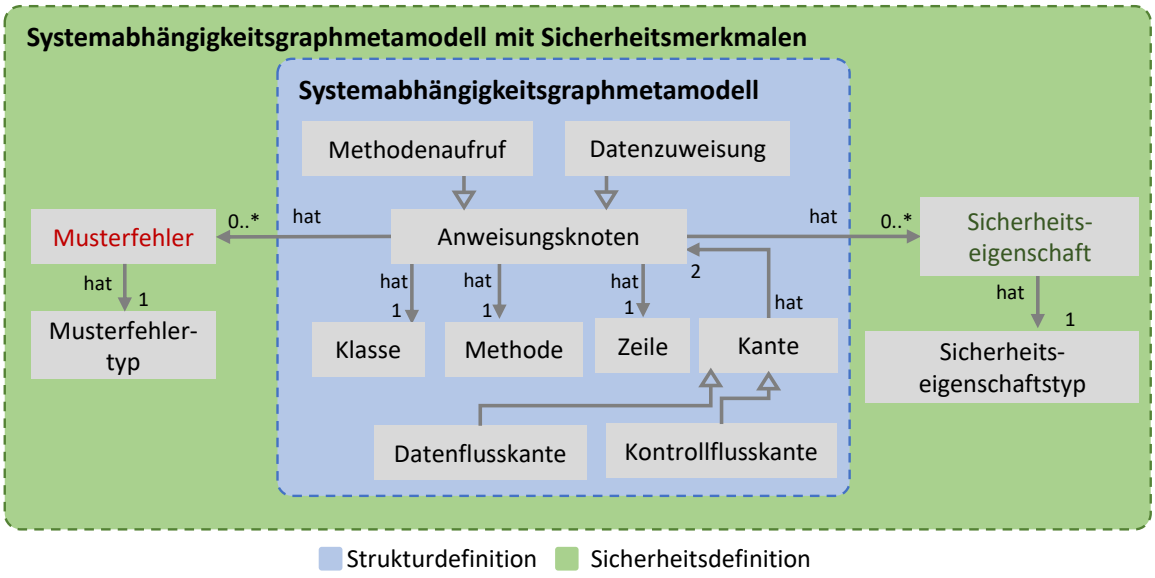

Abbildung 5.5: Metamodell eines um Sicherheitsmerkmale erweiterten SDG.

Knoten werden durch Kanten verbunden. Es gibt sowohl Kontrollflusskanten, welche die Aufrufreihenfolge eines Programms beschreiben und Datenflusskanten, welche beschreiben wie Daten zwischen den verschiedenen Knoten fließen. Anweisungsknoten beziehen sich allgemein immer auf die Klasse und Methode, in der die Anweisung auftritt. Im Sinne des Quelltextes hat eine Anweisung außerdem immer genau eine Zeile, der diese zugeordnet werden kann.

Für die Kopplung der Analysen werden Musterfehler genau dem Anweisungsknoten zugewiesen, in dem diese auftreten. Ein Anweisungsknoten kann dabei mehrere Musterfehler besitzen. Sicherheitseigenschaften werden ebenso dem Anweisungsknoten zugewiesen, für die sie eine Sicherheitseigenschaft definieren. Im Rahmen der vorliegenden Arbeit sind dies Methodenaufruf-Anweisungsknoten, da Sicherheitseigenschaften für Aufrufabhängigkeiten definiert werden. Ein Anweisungsknoten kann ebenfalls mehrere Sicherheitseigenschaften besitzen. Die Zuweisung von Musterfehlern und Sicherheitseigenschaften auf Anweisungsknoten erfolgt mittels eines Attributs.

5.3.2.2 Realisierung der Kopplung mittels SDG mit Sicherheitsmerkmalen

Der Ablauf der Kopplung auf Basis des SDG ähnelt dem der Kopplung auf Basis des Aufrufgraphen. Auch hier erfolgt die Suche der verletzten Sicherheitseigenschaften in drei Schritten. Im ersten Schritt werden die Sicherheitseigenschaften des Architekturmodells zugewiesen. Anders als bei der Kopplung auf Basis des Aufrufgraphen werden diese jedoch keinen Kanten, sondern den entsprechenden Anweisungsknoten zugewiesen. Da ein Systemabhängigkeitsgraph jede Anweisung eines Programms als Knoten darstellt, existiert für einen Aufruf einer Methode genau ein entsprechender Knoten. Für einen Methodenaufruf mit Sicherheitseigenschaft wird deshalb die Sicherheitseigenschaft dem Knoten zugewiesen, welcher diesen Aufruf repräsentiert. Im zweiten Schritt werden die im Quelltext gefundenen Fehler den Knoten zugeordnet, welche die Anweisungen darstellen, in der die

jeweiligen Fehler auftreten. Diese Zuweisung erfolgt auf Basis des Klassennamens und der Quelltextzeile der Anweisung des zuzuweisenden Fehlers. Sind alle Sicherheitsmerkmale zugewiesen, kann im letzten Schritt ermittelt werden, welche Sicherheitseigenschaften durch Musterfehler verletzt werden. Hierzu wird auf Basis der Datenflusskanten geprüft, ob ein Pfad von einem Fehlerknoten zu einem Sicherheitseigenschaftknoten existiert. Diese Pfadsuche kann hierbei auch über mehrere Methoden und Klassen hinweg erfolgen. Existiert ein solcher Pfad, dann bedeutet dies, dass ein mit einer fehlerhaft verwendeten Sicherheitsbibliothek manipuliertes Datum im Methodenaufruf mit einer Sicherheitseigenschaft übergeben wird. Besteht zusätzlich eine Abbildung von Sicherheitseigenschaftstyp zu Musterfehlertyp gemäß des Typkorrespondenzmodells, beschrieben in Abbildung 5.2, dann gilt die Sicherheitseigenschaft als verletzt. Die verletzten Sicherheitseigenschaften können können für die Rückkopplung aus dem Architektursicherheitsmodell äquivalent zur CG-Variante entfernt werden.

Die folgende Abbildung 5.6 verdeutlicht dies am Quelltext 5.2. Im Gegensatz zum vorherigen Quelltext 5.1, wird das Datum `geheim` tatsächlich übergeben.

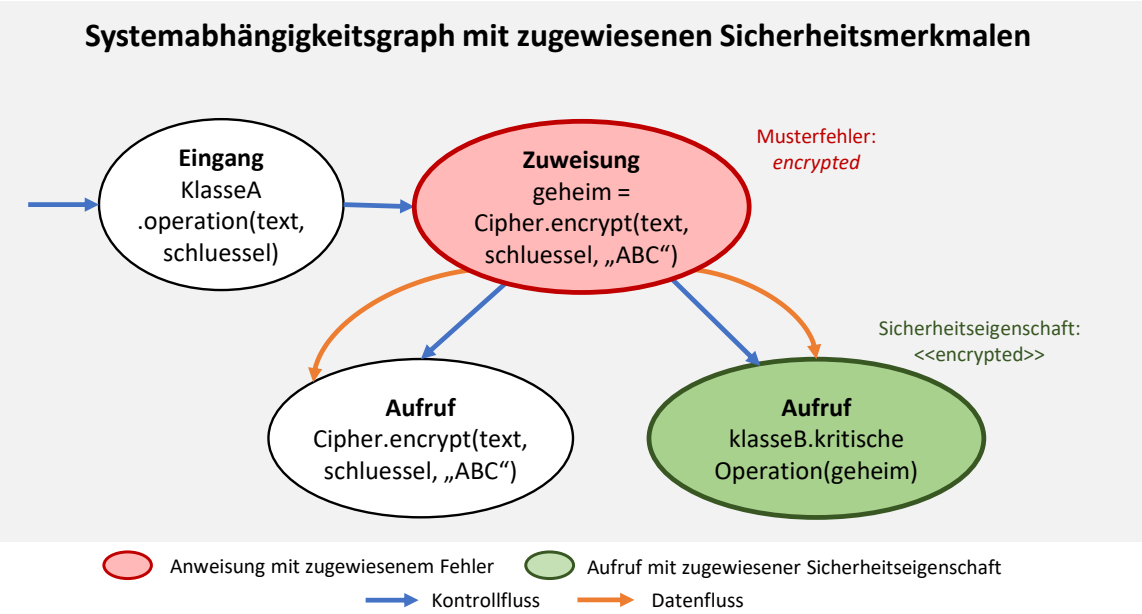

Abbildung 5.6: Vereinfachtes Beispiel eines aus dem Quelltext 5.2 generierten SDGs mit zugewiesenen Sicherheitsmerkmalen.

```
1 public class KlasseA {
2   public void operation(byte[] text, byte[] schluessel) {
3     byte[] geheim = Cipher.encrypt(text, schluessel, "ABC");  // Musterfehler
4     klasseB.kritischeOperation(geheim);
5   }
6 }
```

Quelltext 5.2: Beispielmethode mit einem Musterfehler und Aufruf einer Methode mit Sicherheitseigenschaft, über die ein fehlerhaft verschlüsseltes Datum übergeben wird.

Hierbei wird ein Text mit der Methode `encrypt` der Klasse `Cipher` mit dem vom Entwickler gewählten Verschlüsselungsalgorithmus “ABC” verschlüsselt. Dieser Verschlüsselungsalgorithmus ist im Regelsatz nicht in den erlaubten Verschlüsselungsalgorithmen für die Klasse Cipher aufgeführt, weshalb eine Quelltextanalyse hier einen Fehler zurückgibt. Da die Methode `encrypted` zur Verschlüsselung von Daten genutzt wird, hat dieser Fehler den Typ *encrypted*. Auf Basis der Quelltextzeile und der Klasse, in der der Fehler auftritt, wird dieser dem entsprechenden Knoten des SDGs zugeordnet. Im Architekturmodell wurde zuvor für den Aufruf der Methode `kritischeOperation` der Klasse `KlasseB` von Klasse `KlasseA` eine Sicherheitseigenschaft mit dem Typ «`encrypted`» definiert. Diese Eigenschaft wird auf Basis des Klassen- und Methodennamen dem Knoten des SDGs zugeordnet, welcher diesen Aufruf im Quelltext repräsentiert. Äquivalent zum Ansatz mit dem Aufrufgraphen muss der Methodenaufruf, auf dem die Sicherheitseigenschaft definiert wurde, eindeutig einem Methodenaufruf im Quelltext zugeordnet werden können. Zwischen der Anweisung, welche die Verschlüsselung des Textes mit der `encrypt` Methode enthält und der Anweisung, welche den Aufruf der `kritischeOperation` Methode enthält, besteht eine Verbindung bezüglich des Datenflusses. Diese besteht, da die Variable `geheim` an die `kritischeOperation` Methode übergeben wird. Im SDG existiert hier also eine Kante, welche die Knoten dieser Anweisungen miteinander verbindet. So kann ein Pfad von dem Knoten mit dem zugewiesenen Musterfehler und dem Knoten mit der zugewiesenen Sicherheitseigenschaft gefunden werden. Da die Typen dieser Sicherheitsmerkmale gemäß der Typkorrespondenz in Relation stehen, kann die Sicherheitseigenschaft als verletzt bestimmt werden. Würde man den Ansatz auf Basis des SDG hingegen auf das vorherige Beispiel in Quelltext 5.1 anwenden, würde wegen des fehlenden Datenflusses keine Verletzung der Sicherheitseigenschaft eintreten.

5.3.3 Ergebnis der Kopplungsanalyse und Rückkopplung in das Architektursicherheitsmodell

Mit Hilfe des CG beziehungsweise SDG werden wie in den zuvor beschriebenen Abschnitten 5.3.1 und 5.3.2 die durch Musterfehler verletzten Sicherheitseigenschaften identifiziert. Die Gesamtheit der als verletzt identifizierten Sicherheitseigenschaften stellt das Ergebnis der Kopplungsanalyse dar. Wird eine Sicherheitseigenschaft durch mehrere Musterfehler verletzt, so tritt diese Sicherheitseigenschaft trotzdem nur einmal im Ergebnis Kopplungsanalyse auf. Die Information über die Verletzung soll im letzten Schritt in das Architektursicherheitsmodell zurückgeführt werden. Da die verletzen Sicherheitseigenschaften nicht mehr gelten, können sie aus dem Architektursicherheitsmodell entfernt werden. Zur Entfernung enthält jede Sicherheitseigenschaft, welche dem Graph zugewiesen wurde, eine eindeutige Referenz an die Stelle des Architekturmodells, der sie entstammt. Das Ergebnis der Rückkopplung ist dann ein Architekturmodell, in dem alle verletzten Sicherheitseigenschaften entfernt wurden.

5.4 Gesamtübersicht der Kopplung der Analysen und Modelle

Abbildung 5.2 zeigt eine detaillierte Gesamtübersicht bezüglich des Ablaufs der Kopplung. Die Grafik setzt sich zusammen aus den in den vorherigen Abschnitten beschriebenen Konzepten und zeigt die Architektur- und Quelltextsicht und den in Abschnitt 5.1 beschriebenen Kopplungsansatz, der beide Sichten miteinander verbindet.

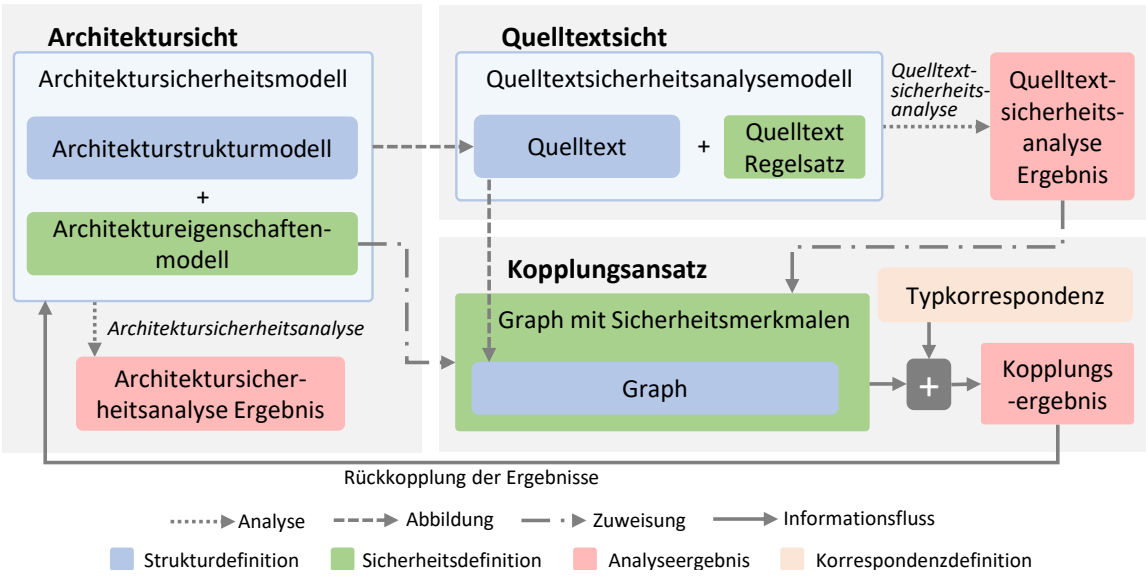

Abbildung 5.7: Übersicht des Kopplungsablaufs auf Basis der Modelle auf Architektur- und Quelltextsicht, sowie des in dieser Arbeit konzipierten Ansatzes.

6 Implementierung des Kopplungsansatz und Realisierung mit zwei Analysen

Die Implementierung setzt den zuvor beschriebenen Ansatz für die Zugriffsanalyse von Kramer u. a. [16] und CogniCrypt [6] um. Die Umsetzung erfolgt in Java. In Abschnitt 6.1 wird ein Überblick über die Pakete der Implementierung gegeben. Die Architektur und das Implementierungskonzept werden in Abschnitt 6.2 näher erläutert. Abschnitt 6.3 beschreibt die Einbindung der Zugriffsanalyse zur Definition von Aufrufabhängigkeiten mit Sicherheitseigenschaften. Der Abschnitt 6.4 erläutert die Einbindung von CogniCrypt zur Identifikation von Musterfehlern. In Abschnitt 6.5 wird beschrieben, wie der Aufrufgraph und Systemabhängigkeitsgraph erzeugt werden. Abschließend wird in Abschnitt 6.6 erklärt, wie die Abbildung von Sicherheitseigenschaften und Musterfehlern implementiert wurde, und wie der Kopplungsansatz dadurch realisiert wird.

6.1 Paketübersicht

Die Implementierung der vorliegenden Arbeit kann in fünf wesentliche Paket-Gruppen unterteilt werden. Diese sind in der folgenden Abbildung 6.1 dargestellt.

Die `edu.kit.kastel.sdq.coupling.patternbased` Pakete umfassen die Funktionalität zur Ausführung der Kopplung und der Benutzeroberfläche sowie die Realisierung des Kopplungsansatzes selbst. Die `edu.kit.kastel.sdq.coupling.patternbased.architecture` Pakete enthalten Schnittstellen und abstrakte Klassen zum Einlesen der Sicherheitseigenschaften des Architekturmodells sowie deren konkrete Realisierung auf der Basis der Zugriffsanalyse. Die Schnittstellen und abstrakten Klassen zum Einlesen und Verarbeiten der Musterfehler eines Quelltextes sind in `edu.kit.kastel.sdq.coupling.patternbased.code` enthalten. Diese werden für CogniCrypt durch konkrete Klassen realisiert. Die Schnittstellen und abstrakten Klassen zum Einlesen des Systemabhängigkeitsgraphen befinden sich in den `edu.kit.kastel.sdq.patternbased.coupling.sdg` Paketen. Diese werden durch eine Implementierung mittels JOANA realisiert. Außerdem umfasst die Implementierung des Kopplungsansatzes eine Reihe von Hilfsklassen und Methoden. Weiterhin gibt es Exceptions, welche spezifisch für die vorgenommene Implementierung definiert wurden. Die Tests für die Implementierung sind ebenfalls in einem Paket gebündelt. Die Funktionalität der wichtigsten Schnittstellen und deren Realisierung werden in den folgenden Abschnitten konzeptionell beschrieben.

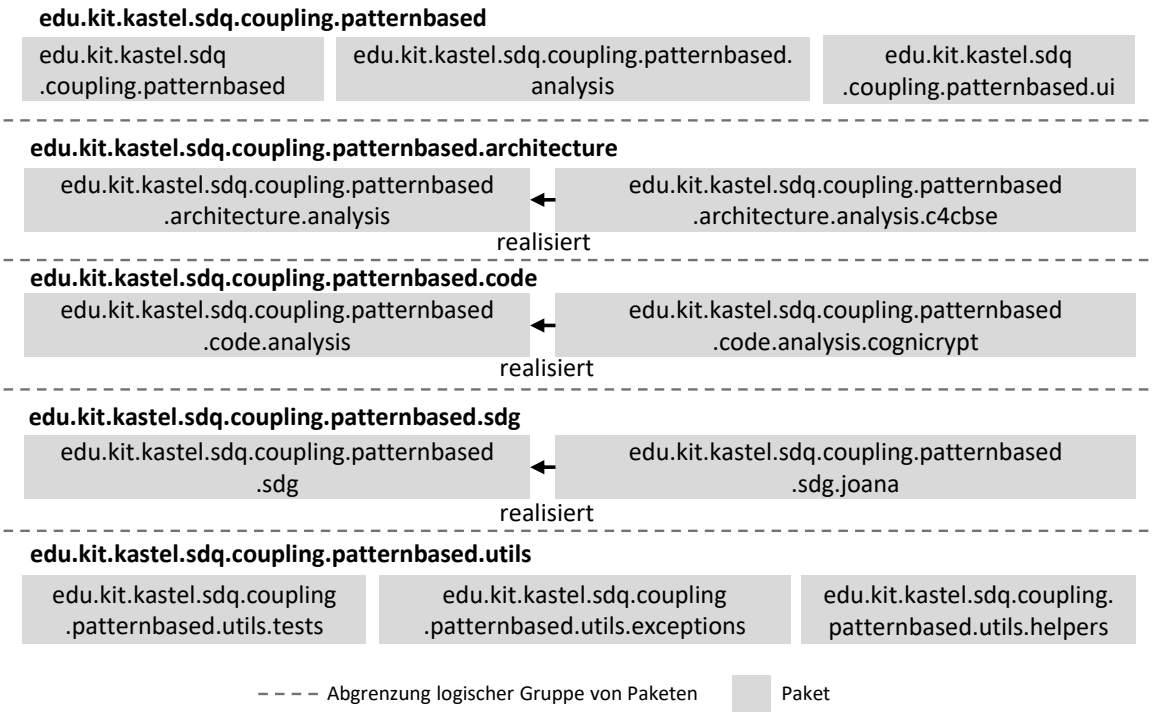

Abbildung 6.1: Übersicht der in der Implementierung enthaltenen Pakete.

6.2 Architektur- und Implementierungskonzept

Die Implementierung besteht aus einer Reihe von abstrakten Klassen und Schnittstellen, welche für die Verwendung von spezifischen Architektur- und Quelltextanalysen in Form von Klassen implementiert werden. Eine Übersicht der Abstraktionen sowie die in dieser Arbeit vorgenommenen konkreten Implementierungen sind in Abbildung 6.2 dargestellt. Dabei gruppieren die zuvor eingeführten Pakete die einzelnen Teile des Kopplungsprogramms.

Die Klasse `Coupling` im Paket `edu.kit.kastel.sdq.coupling.patternbased` führt die unterschiedlichen Teile des Systems zusammen. Sie enthält die `main`-Methode des Programms, führt die Benutzerschnittstelle aus und übernimmt die Datenübergabe zwischen den einzelnen Teilen des Programms. Architektureigenschaften werden mit Hilfe eines `IArchitecturePropertyManager` aus einem Architekturmodell eingelesen. In der vorliegenden Arbeit wird diese Schnittstelle konkret durch den `C4CbseArchitecturePropertyManager` realisiert. Dieser liest Sicherheitseigenschaften, definiert durch den Ansatz Confidentiality4CBSE, ein. Das Paket `edu.kit.kastel.coupling.architecture.analysis` definiert abstrakte Klassen für die in Abschnitt 4.1.3 beschriebenen Metamodelle der Architektursicht. Diese abstrakten Klassen werden durch die Implementierung umgesetzt zu spezifischen Sicherheitseigenschaften eines Ansatzes (`C4CbseArchitectureProperty`). Diese können auch spezifische Informationen der verwendeten Architekturanalyse (z.B. einen eindeutigen Identifikator der Sicherheitseigenschaft) enthalten, wobei die Kopplungsanalyse zur Suche nach Verletzungen ausschließlich die Informationen der Abstraktionen

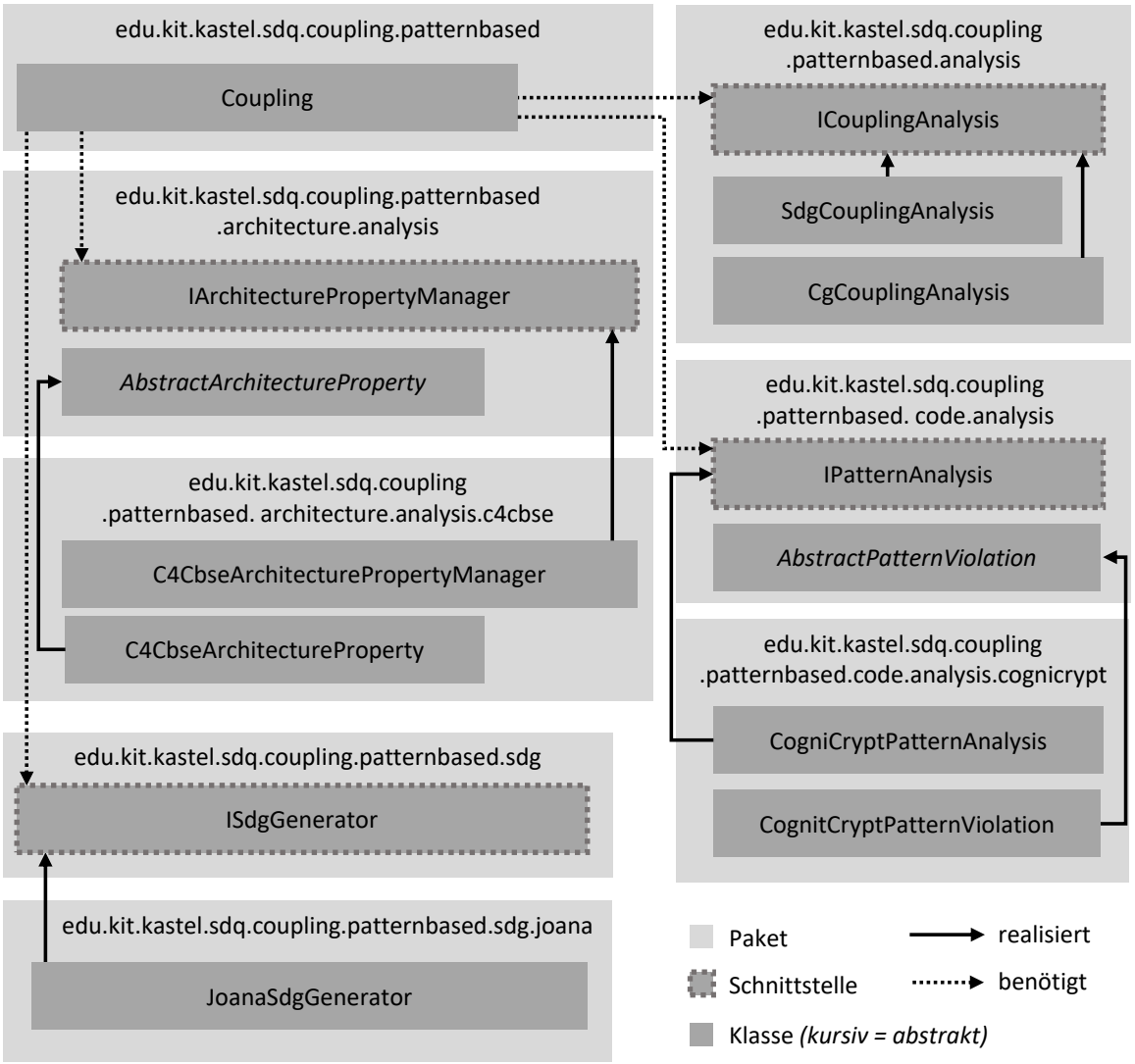

Abbildung 6.2: Übersicht der Architektur der Kopplungsanalyse mit Paketen, Schnittstellen, abstrakten Klassen und konkreten Implementierungen.

verwendet. Dadurch ist der konkrete Ansatz zur Definition von Architektureigenschaften austauschbar. Im Quelltext werden Musterfehler mit Hilfe einer `IPatternAnalysis` aus dem Paket `edu.kit.kastel.coupling.code.analysis` identifiziert. Im Rahmen dieser Arbeit wird diese Schnittstelle durch die `CogniCryptPattern Analysis` für CogniCrypt umgesetzt. Die Klasse `AbstractPatternViolation` definiert einen abstrakten Musterfehler. Diese Klasse wird für die spezifische Quelltextanalyse erweitert, wie in dieser Arbeit durch die Klasse `CogniCryptPatternViolation`. Äquivalent zur Verarbeitung der Architektureigenschaften verwendet die Kopplung für die Verarbeitung der Musterfehler nur die Abstraktionen. Dadurch ist die Kopplung ebenfalls unabhängig von den verwendeten Quelltextsicherheitsanalysen. Zur Realisierung der Kopplung werden außerdem der Systemabhängigkeits- beziehungsweise Aufrufgraph benötigt. Diese können mit Hilfe der Schnittstelle `ISdgGenerator` erzeugt werden. Da der Systemabhängigkeitsgraph auch den

Aufrufgraphen umfasst, wird der CG in der vorliegenden Implementierung aus dem SDG abgeleitet. In der vorliegenden Arbeit wird diese Schnittstelle zur Graphextraktion mit Hilfe von JOANA realisiert. Die Informationen, welche durch die zuvor beschriebenen Komponenten des Systems gewonnen werden, bilden die Grundlage für die Kopplungsanalyse. Der Ablauf der Kopplung wird gemäß der Schnittstelle `ICouplingAnalysis` implementiert. Hier stellen die `CgCouplingAnalysis` und die `SdgCouplingAnalysis` zwei Implementierungen dieser Schnittstelle dar. `CgCouplingAnalysis` definiert eine gekoppelte Analyse auf Basis eines Aufrufgraphen. `SdgCouplingAnalysis` realisiert eine gekoppelte Analyse mittels eines Systemabhängigkeitsgraphen. Die gekoppelten Analysen bestimmen die im Architekturmodell verletzten Sicherheitseigenschaften. Diese können dann mit Hilfe des `IArchitecturePropertyManager` aus dem Architekturmodell entfernt werden.

6.3 Einbindung von Confidentiality4CBSE

Im folgenden Abschnitt 6.3.1 wird beschrieben, wie Confidentiality4CBSE dazu verwendet wird Aufrufabhängigkeiten mit Sicherheitseigenschaften durch `LinkingResources` zu definieren. Der Abschnitt 6.3.2 beschreibt konzeptionell, wie das Architekturmodell für die Kopplung eingelesen wird.

6.3.1 Abbildung von LinkingResources auf Aufrufabhängigkeiten mit Sicherheitseigenschaften

Die in Abschnitt 4.1.1 beschriebenen Aufrufabhängigkeiten mit Sicherheitseigenschaften sollen durch den Kopplungsansatz auf Verletzung geprüft werden können. Mit Confidentiality4CBSE Modellen für die Zugriffsanalyse lassen sich Sicherheitseigenschaften allerdings nicht direkt auf Aufrufen zwischen zwei Komponenten definieren. Deshalb wird im Folgenden beschrieben, wie diese Modelle in das für die Kopplung geforderte Metamodell der Architektursicherheitsanalyse überführt werden.

Wie in Abschnitt Absatz 2.3.2 beschrieben, lassen sich für Confidentiality4CBSE Modelle `ResourceContainer` durch `LinkingResources` verbinden. Diese `ResourceContainer` enthalten eine Anzahl von Komponenten des modellierten Systems. Sind zwei Komponenten in unterschiedlichen `ResourceContainern`, welche über eine `LinkingResource` miteinander verbunden sind, so existiert für diese im Architekturmodell eine Aufrufabhängigkeit. Auf einer `LinkingResource` können zusätzlich Sicherheitseigenschaften definiert sein. Für die Implementierung wird die Sicherheitseigenschaft «`encrypted`» betrachtet. Es wird davon ausgegangen, dass diese Eigenschaft durch eine Verschlüsselung auf Applikationsebene realisiert werden muss. Demnach würden bei einer Kopplung ein relevanter Musterfehler im Sinne der Typkorrespondenz die «`encrypted`» Eigenschaft verletzen. In der vorliegenden Arbeit wird davon ausgegangen, dass alle Aufrufe zwischen den paarweisen Komponenten zweier `ResourceContainer` die Eigenschaften der verbindenden `LinkingResource` erfüllen müssen. Das heißt, eine Aufrufabhängigkeit mit Sicherheitseigenschaft existiert genau dann, wenn zwei Komponenten in unterschiedlichen `ResourceContainern` durch eine `LinkingResource` mit Sicherheitseigenschaft verbunden sind. Wird die aus dem Confidentiality4CBSE Modell abgeleitete Aufrufabhängigkeit mit Sicherheiseigenschaft verletzt, so

wird in der vorliegenden Arbeit die dazugehörige LinkingResource als verletzt angenommen. Es können somit auch Aufrufabhängigkeiten mit Sicherheitseigenschaften verloren gehen, die eigentlich nicht direkt durch eine Verletzung betroffen sind. Die Möglichkeit zur Festlegung von unencryptedData auf LinkingResources wird in dieser Arbeit nicht berücksichtigt.

Zur Veranschaulichung dient das folgende Beispiel Abbildung 6.3.

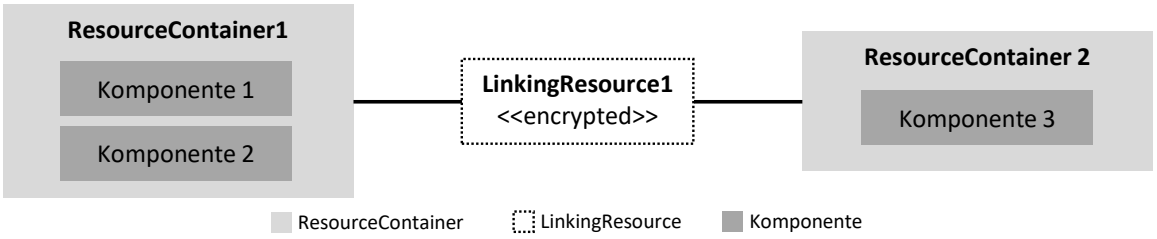

Abbildung 6.3: Beispiel einer LinkingResource mit Sicherheitseigenschaft.

Für die beiden ResourceContainer ResourceContainer1 und ResourceContainer2 ist eine LinkingResource festgelegt. Dieser wurde die Sicherheitseigenschaft «encrypted» zugewiesen. Der ResourceContainer1 enthält die Komponenten 1 und 2. Der ResourceContainer2 enthält Komponente 3. Aufgrund der LinkingResource ist für alle Aufrufe von Methoden der Komponente 3 durch Komponente 1 und 2 eine Aufrufabhängigkeit mit Sicherheitseigenschaft festgelegt. Da die LinkingResource eine ungerichtete Verbindung darstellt, gilt die Eigenschaft auch für alle Aufrufe von Methoden der Komponenten 1 und 2, durch die Komponente 3. Ein Musterfehler mit entsprechend verletzendem Typ in der Implementierung der Komponenten führt hier folglich also zu einer Verletzung der Sicherheitseigenschaft.

6.3.2 Einlesen der Sicherheitseigenschaften

Das Architekturmodell liegt in Form mehrerer XML-Dateien vor, welche automatisch eingelesen werden. Aus den Informationen der LinkingResourcen und ResourceContainern erzeugt der C4CbseArchitecturePropertyManager konkrete Instanzen von Sicherheitseigenschaften in Form von C4CbseArchitecturePropertys. Diese enthalten neben der Information über die aufrufende und aufgerufene Komponente den Sicherheitseigenschaftstyp sowie eine Referenz auf die definierende LinkingResource. Diese Referenz wird benötigt, um die verletzen Sicherheitseigenschaften anhand ihrer IDs eindeutig im Architekturmodell zu identifizieren und im Falle einer Verletzung zu entfernen.

6.4 Einbindung von CogniCrypt zur Bestimmung von Musterfehlern

In Abschnitt 6.4.1 wird zunächst konzeptionell beschrieben wie Musterfehler mittels CogniCrypt in der Implementierung bestimmt werden. Anschließend wird Abschnitt 6.4.2

erläutert, wie das Ergebnis von CogniCrypt auf das Quelltextsicherheitsanalyse Ergebnismetamodell aus Abschnitt 4.2.4 dieser Arbeit abgebildet wird.

6.4.1 Bestimmung der Musterfehler mittels CogniCrypt

CogniCrypt führt die Sicherheitsanalyse auf der kompilierten jar-Datei eines Programms, welche durch den Nutzer übergeben wird, aus. In der vorliegenden Arbeit wird der `HeadlessCryptoScanner` von CogniCrypt verwendet. Dieser wird im Kopplungsprogramm für die Quelltextanalyse direkt in die `CogniCryptPatternAnalysis` eingebunden. Als Regelsatz werden für diese Arbeit die vordefinierten Regeln für die Java Cryptographic Architecture, wie in Abschnitt 2.3.4 beschrieben, gewählt. Die Ausgabe der CogniCrypt-Analyse erzeugt eine JSON-Datei, die alle in der jar-Datei gefundenen Musterfehler enthält.

6.4.2 Abbildung auf das Ergebnismetamodell der Quelltextsicherheitsanalyse

Musterfehler, die durch CogniCrypt ausgegeben werden, folgen dem in dieser Arbeit beschriebenen Musterfehler Metamodell und enthalten alle notwendigen Informationen. Diese Informationen umfassen das Paket, die Klasse, Methode und Quelltextzeile, in der der Fehler auftritt und die Klasse und Methode die falsch verwendet wurden. Lediglich der Musterfehlertyp muss ergänzt werden. Dies erfolgt indem die Namen der verletzten Klasse auf einen entsprechenden Musterfehlertyp abbgebildet werden. Es wurde keine vollständige Abbildung aller definierten Regeln von CogniCrypt definiert, die Klasse zur Abbildung ist jedoch beliebig erweiterbar. In der vorliegenden Arbeit wurde festgelegt, dass Musterfehler, welche Aufrufe zur Klasse `Cipher` darstellen, den Musterfehlertyp `encrypted` besitzen. Mit den notwendigen Informationen wird für jeden Fehler ein `CogniCryptPatternViolation` Objekt erzeugt. Die so erzeugten Musterfehler werden dann für die Abbildung auf den Systemabhängigkeits- bzw. Aufrufgraphen benötigt.

6.5 Erzeugung des SDG und CG

Zur Extraktion des Systemabhängigkeitsgraphen wird JOANA verwendet. Die jar-Datei einer kompilierten Anwendung kann mittels `joana.ui.ifc.wala.cli.jar` über die Kommandozeile zu einem Systemabhängigkeitsgraphen verarbeitet werden. Für die Durchführung der SDG Generierung wird in der vorgenommenen Implementierung vorausgesetzt, dass die `main`-Methode eines zu analysierenden Programms durch die JOANA-Annotation `@EntryPoint(tag="coupling")` versehen wird. In der vorliegenden Arbeit wird Kommandozeilenanwendung für die Graphextraktion direkt aus dem Java Programm zur Kopplung ausgeführt. Der Systemabhängigkeitsgraph wird von JOANA in Form einer graphml-Datei ausgegeben. Diese wird mit Hilfe der JGraphT-Bibliothek [41] für die Kopplung eingelesen. Der eingelesene Graph enthält neben den Datenfluss- und Kontrollflussinformationen auch eine Reihe von nicht benötigten Kanten- und Knotentypen. Zur Vereinfachung des Graphen werden die nicht benötigten Kanten entfernt. Knoten werden zu Anweisungs- und Aufrufknoten vereinfacht. Jeder Knoten des Graphen enthält die notwendigen Informationen über die dazugehörige Klasse, Methode und Zeile, die dieser betrifft. So

können die Sicherheitseigenschaften aus dem Architekturmodell exakt den Knoten der entsprechenden Methodenaufrufe zugeordnet werden. Auch können so die Musterfehler auf die Knoten abgebildet werden, welche die Anweisungen repräsentieren, in denen die Musterfehler auftreten.

6.6 Realisierung der Kopplung

Die Sicherheitseigenschaften, Musterfehler und Graphen werden für die Kopplung zunächst eingelesen und auf die Abstraktionen aus Kapitel 4 für anwendbare Sicherheitsanalysen abgebildet. Im nächsten Schritt folgt das Abbilden der Sicherheitseigenschaften auf den Graphen.

Für die Zuweisung von Architektureigenschaften werden, wie in Abschnitt 5.3.2.2 beschrieben, die Methodenaufrufknoten des erzeugten Systemabhängigkeitsgraphen, verwendet. Im Falle der Verwendung von Confidentiality4CBSE sind das genau die Aufrufknoten, die einen Aufruf von einer Komponente zu einer anderen Komponente darstellen, wobei die beiden Komponenten über eine `LinkingResource` durch eine Sicherheitseigenschaft versehen wurden. Für die Zuordnung von Sicherheitseigenschaften wird zunächst jeder Methodenaufrufknoten betrachtet. Stellt dieser einen Aufruf einer Komponente zu einer anderen dar, die gemäß der in Abschnitt 6.3 beschriebenen Abbildung von `LinkingResources` eine Sicherheitseigenschaft besitzt, wird diese dem Knoten zugewiesen. Die Aufrufknoten-Anweisungen des Systemabhängigkeitsgraphen sind auch genau die Teilmenge von Knoten, welche die Kanten des Aufrufgraphen beschreiben. Folglich können diese auch für die Variante auf Basis des Aufrufgraphen verwendet werden.

Knoten des Systemabhängigkeitsgraphen, welche durch JOANA erzeugt wurden, enthalten Informationen über die Anweisung des Quelltextes, auf die sie sich beziehen, inklusive der Zeilennummer. Folglich können Musterfehler eindeutig auf die Anweisungsknoten abgebildet werden. Hierfür wird für alle Anweisungsknoten im Graphen geprüft, ob diese genau die Anweisung darstellen, welche durch einen Musterfehler als fehlerhaft erkannt wurden. Ist dies der Fall, wird der Musterfehler dem entsprechenden Anweisungsknoten zugewiesen. Für die Variante des Aufrufgraphen wird überprüft, ob ein Fehler im Ausgangsknoten einer Aufrufkante mit Sicherheitseigenschaft vorliegt um diese bei entsprechender Typkorrespondenz direkt als verletzt zu identifizieren.

Zur Pfadsuche von Datenabhängigkeiten zwischen Anweisungen des Systemabhängigkeitsgraphen werden die Datenstruktur und Parameterstruktur Kanten von JOANA traversiert, da diese die Datenflusskanten gemäß des Metamodells in Abschnitt 5.3.2 darstellen. Wird ein Pfad von einem Musterfehler zu einer Sicherheitseigenschaft gefunden, so gilt die Eigenschaft als verletzt. Für die Pfadsuche wird `DijkstraShortestPath` aus der JGraphT-Bibliothek verwendet. Es wird nur geprüft, ob ein Pfad existiert. Die Anzahl der möglichen Pfade von einem Fehler zu einer Sicherheitseigenschaft spielt demnach keine Rolle.

Die zugewiesenen Sicherheitseigenschaften besitzen eine eindeutige Referenz auf die Stelle im Architekturmodell, auf die sie sich beziehen. Im Falle der vorliegenden Implementierung mit Confidentiality4CBSE ist dies die `ID` der `LinkingResource` aus dem die Sicherheitseigenschaft abgeleitet wurde. Zum Löschen einer verletzten Eigenschaft

wird diese Information verwendet, um die betroffene Eigenschaft zu identifizieren und anschließend aus der XML-Datei zu entfernen.

7 Evaluation

Die Kopplung von Sicherheitseigenschaften auf Architektursicht und Musterfehlern auf Quelltextsicht wurde mit zwei Analysen exemplarisch umgesetzt. Der konzipierte Kopplungsansatz soll nun evaluiert werden. Zu diesem Zweck werden in Abschnitt 7.1 zunächst die Forschungsfragen der Evaluation beschrieben. Abschnitt 7.2 erläutert das für die Evaluation verwendete System für die Fallstudie aus Architektur- und Quelltextsicht. In Abschnitt 7.3 werden die Evaluationsszenarien beschrieben. Abschnitt 7.4 präsentiert die Ergebnisse der Evaluation. Abschließend werden die Ergebnisse in Abschnitt 7.5 diskutiert.

7.1 Forschungsfragen

Durch die Evaluation sollen Forschungsfragen beantwortet werden. Deshalb werden die Ziele der Evaluation beschrieben und daraus Forschungsfragen abgeleitet. Für die verschiedenen Forschungsfragen werden außerdem die Metriken beschrieben, mit denen diese beantwortet werden können.

Das übergeordnete Ziel der vorliegenden Arbeit ist, dass die Ergebnisse der Quelltextanalyse in das Architektursicherheitsmodell überführt werden können. Grundsätzlich stellt dieses Ziel also die Machbarkeit der Kopplung dar. Weiterhin sollen zusätzliche Schwachstellen in der Architektur aufgedeckt werden, wenn Sicherheitseigenschaften durch die gefundenen Musterfehler verletzt sind. Daraus folgt:

- Ziel 1 (*Z1*): Eine Kopplung der Sicherheitsanalysen soll machbar sein
  - Frage 1 (*Z1.F1*): Können die gefundenen Sicherheitsverstöße in Form von Musterfehlern im Quelltext in das Architekturmodell überführt werden?
    - Metrik 1: Anzahl der auf die Graphen abgebildeten Sicherheitseigenschaften (erwartet vs. tatsächlich)
    - Metrik 2: Anzahl der auf die Graphen abgebildeten Musterfehler (erwartet vs. tatsächlich)
    - Metrik 3: Anzahl und Pfade der gefundenen Verletzungen von Sicherheitseigenschaften durch Musterfehler
  - Frage 2 (*Z1.F2*): Können durch die Kopplung von Musterfehlern mehr Schwachstellen durch die Architektursicherheitsanalyse aufgedeckt werden, als ohne die Kopplung?
    - Metrik 1: Anzahl der aufgedeckten Schwachstellen der Architekturanalyse vor und nach der Kopplung

Das Ziel des Ansatzes auf Basis des vollständigen Systemabhängigkeitsgraphen ist die präzise Analyse des Datenflusses zur Entfernung von verletzten Sicherheitseigenschaften. Es ist möglich, dass der Aufrufgraph Eigenschaften löscht, die im Sinne des Datenflusses nicht verletzt sind. Weiterhin werden die Sicherheitseigenschaften definiert auf `LinkingResources` abgeleitet zu den spezifischeren Sicherheitseigenschaften auf Aufrufabhängigkeiten. Es genügt allerdings die Verletzung einer einzigen abgeleiteten Sicherheitseigenschaft damit die Eigenschaft der `LinkingResource` entfernt wird. Dabei kann sich die `LinkingResource` aber zusätzlich auf Aufrufabhängigkeiten von Komponenten beziehen, die gar nicht betroffen sind. Demnach kann es auch in diesem Sinne für beide Variationen des Ansatzes zu einer Überapproximation der verletzten Eigenschaften kommen. Daraus folgt demnach:

- Ziel 2 (*Z2*): Ein Kopplungsansatz soll die verletzen Sicherheitseigenschaften möglichst präzise aus dem Architektursicherheitsmodell entfernen.
  - Frage 1 (*Z2.F1*): Werden durch den Kopplungsansatz auf Basis des Aufrufgraphen mehr Fehler gefunden, als es im Sinne des Datenflusses tatsächlich gibt?
    - Metrik 1: Anzahl der aufgedeckten Schwachstellen der Architekturanalyse vor und nach der Kopplung
  - Frage 2 (*Z2.F2*): Werden bei der Rückführung von verletzen Sicherheitseigenschaften auf LinkingResources, abgeleitete Sicherheitseigenschaft entfernt, die nicht verletzt sind?
    - Metrik 1: Abgeleitete Sicherheitseigenschaften aus dem Architektursicherheitsmodell vor und nach der Kopplung

Die beschriebenen Forschungsfragen werden im Folgenden durch ein exemplarisches System für die Evaluation und eine Reihe von Szenarien überprüft.

7.2 Beschreibung des Systems für die Evaluation

In der vorliegenden Arbeit wird ein Testsystem zur Evaluierung des implementierten Kopplungsansatzes umgesetzt. Abschnitt 7.2.1 beschreibt zunächst die Funktion und Implementierung des Systems. Der Abschnitt 7.2.2 präsentiert das Architektursicherheitsmodell des Systems und die daraus abgeleiteten Aufrufabhängigkeiten mit Sicherheitseigenschaften.

7.2.1 Funktionsbeschreibung

In der vorliegenden Arbeit wird das existierende `TravelPlanner` Beispiel [42] als Testsystem verwendet. Dieses wurde auf Basis der Umsetzung und Anpassungen in der Arbeit von Häring [7] für die Evaluationszwecke dieser Arbeit weiter angepasst. Das Testsystem stellt eine Smartphone Anwendung dar, wobei mit Hilfe einer `TravelPlanner` Komponente Flüge über ein Smartphone gebucht werden können. Für die Buchung kommuniziert der `TravelPlanner` mit der Komponente `TravelAgency`, welche die Buchung abwickelt. Angeboten werden die Flüge von der Komponente `Airline`. Für die Buchung ist es notwendig,

dass die Kreditkarteninformationen des Kunden von der Komponente `CreditCardCenter` in Erfahrung gebracht werden. Eingaben in den `TravelPlanner` erfolgen durch den Nutzer mittels der Komponente `UserInterface`. Die Komponente `AirlineLogger` kann von der `Airline` verwendet werden, um Logging-Informationen während der Ausführung zu protokollieren. Der Quelltext des Testsystems von Häring [7] wurde so angepasst, dass sich die übergebenen Daten von Methodenaufrufen zwischen den Komponenten mittels der Klasse `javax.crypto.Cipher` verschlüsseln lassen.

7.2.2 Architektursicherheitsmodell des Systems

Im Folgenden wird das Architektursicherheitsmodell des Testsystems vorgestellt. In Abschnitt 7.2.2.1 wird die `ResourceEnvironment` des Architekturmodells beschrieben. In Abschnitt 7.2.2.2 wird der für diese Arbeit ergänzte `Adversary` erklärt. Dieser wurde hinzugefügt, da das Ausgangsmodell bisher keine Angreifer für die im Architektursicherheitsmodell festgelegten `LinkingResources` enthält. Abschnitt 7.2.2.3 beschreibt außerdem die aus dem Architektursicherheitsmodell abgeleiteten Aufrufabhängigkeiten mit Sicherheitseigenschaften.

7.2.2.1 ResourceEnvironment des Architektursicherheitsmodells

Die `ResourceEnviroment` definiert die `ResourceContainer` und `LinkingResources` des Architektursicherheitsmodells. Die Komponenten der `ResourceContainer` werden durch eine Allokation ihrem entsprechenden `ResourceContainer` zugewiesen. Die Abbildung 7.1 zeigt das `ResourceEnvironment` des Architektusicherheitsmodells für das für die Evaluation verwendete Testsystem.

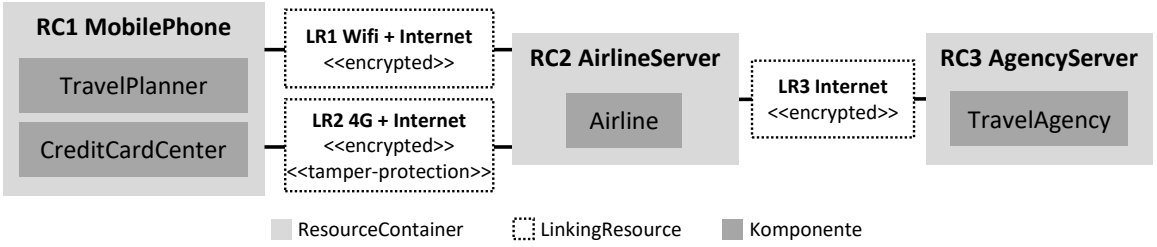

Abbildung 7.1: `ResourceEnvironment` des Architekturmodells für das für die Evaluation verwendete Testsystems.

Der `RC1 MobilePhone` kommuniziert mit dem `RC2 AirlineServer` via Internet entweder über die `LinkingResource` `LR1 Wifi + Internet` oder die `LinkingResource` `LR2 Internet + 4G`. Dabei wird auf Architektursicht angenommen, dass für die beiden `LinkingResources` die Sicherheitseigenschaft «`encrypted`» gilt. Weiterhin hat die `LinkingResource` `LR1 "Wifi + Internet"` die «`tamper-protection`» Eigenschaft. Das heißt, dass Angreifer ohne entsprechende Fähigkeit die «`tamper-protection`» zu umgehen auf die `LinkingResource` keinen Zugriff erlangen können. Der `RC3 AgencyServer` kommuniziert in dem gewählten System für die Evaluation mit dem `RC2 AirlineServer` über das Internet. Für die dazugehörige `LinkingResource LR3` gilt ebenfalls die «`encrypted`» Sicherheitseigenschaft.

7.2.2.2 Adversary des Architektursicherheitsmodell

Im Architektursicherheitsmodell der Arbeit von Häring existiert kein Adversary, welcher auf die in der `ResourceEnviroment` festgelegten `LinkingResources` Zugriff hat. Dieser wurde deshalb im Architektursicherheitsmodell ergänzt. In der vorliegenden Arbeit wird davon ausgegangen, dass der Angreifer Zugriff auf die WLAN Verbindung erlangen kann. Weiterhin hat der Angreifer sich in die Verbindung zwischen dem `AirlineServer` und der `TravelAgency` eingeschleust. Der Angreifer besitzt auf Grund der für `LR2` definierten `«tamper-protection»` keinen Zugriff auf die `LinkingResource`. Dadurch soll ein Wegfallen der `«encrypted»` Eigenschaften für die `LR2` keine Auswirkungen auf die Sicherheit des Systems haben und keine zusätzlichen Fehler verursachen.

7.2.2.3 Abgeleitete Sicherheitseigenschaften aus den `LinkingResource`s

Aus dem Confidentiality4CBSE Architektursicherheitsmodell werden die Aufrufabhängigkeiten mit Sicherheitseigenschaften abgeleitet. Die Sicherheitseigenschaften werden auf den Graphen abgebildet. Zum besseren Verständnis der Evaluation zeigt die folgende Tabelle 7.1, welche Aufrufabhängigkeiten mit Sicherheitseigenschaften aus den `LinkingResources` mit Sicherheitseigenschaften spezifisch für das implementierte System der Evaluation vor der Kopplung abgeleitet werden können. Zur Referenzierung in der Evaluationsbeschreibung wurden die abgeleiteten Sicherheitseigenschaften mit eindeutigen `ID`s versehen. Es werden nur die Sicherheitseigenschaften aufgeführt, die auch in Form von tatsächlichen Aufrufen im Quelltext existieren.

ID	Quelle	Eigenschaft	Aufrufer	Aufgerufen
SE1	LR1	`«encrypted»`	TravelPlanner.declassifiedCCD	Airline.bookFlightOffer
SE2	LR2	`«encrypted»`	TravelPlanner.declassifiedCCD	Airline.bookFlightOffer
SE3	LR3	`«encrypted»`	Airline.bookFlightOffer	TravelAgency.payCommission
SE4	LR3	`«encrypted»`	TravelAgency.getFlightOffers	Airline.getFlightOffers

Tabelle 7.1: Abgeleitet Sicherheitseigenschaften auf Aufrufabhängigkeiten basierend auf den `LinkingResources` des Systems für die Evaluation.

7.3 Evaluationsszenarien

Um die zuvor beschriebenen Forschungsfragen zu beantworten, werden eine Reihe von Evaluationsszenarien beschrieben. Hierfür werden in Abschnitt 7.3.1 die relevanten Fehlerklassen eingeführt, aus denen sich Änderungen für die Metriken der dazugehörigen Forschungsfragen ergeben. In Abschnitt 7.3.2 wird beschrieben wie die Fehlerklassen durch Musterfehler im Quelltext realisiert werden und wie erwartet wird, wie sich die Musterfehler im TravelPlanner Evaluationssystem bei der Kopplung auswirken. Abschnitt 7.3.3 fasst die Evaluationsszenarien zusammen.

7.3.1 Fehlerklassen

Für die verschiedenen Szenarien werden Musterfehler so integriert, dass verschiedene Fehlerklassen abgedeckt sind. Die Fehlerklassen beziehen sich auf die unterschiedlichen Forschungsfragen, die den Zielen der Evaluation untergeordnet sind. Die folgenden Fehlerklassen zur Ableitung von Metriken werden durch die Evaluationsszenarien umgesetzt.

- **FK1:** Beschreibt die Verletzung von Sicherheitseigenschaften durch Musterfehler mit Auswirkung auf die Ergebnisse der Architekturanalyse. Durch diese Fehlerklasse werden die Metriken zur Diskussion von *Z1.F1*, *Z1.F2*, *Z2.F1* und *Z2.F2* beeinflusst.
- **FK2:** Beschreibt die Verletzung von Sicherheitseigenschaften durch Musterfehler ohne Auswirkung auf die Ergebnisse der Architekturanalyse. Durch diese Fehlerklasse werden die Metriken zur Diskussion von *Z1.F2* beeinflusst.
- **FK3:** Beschreibt die Verletzung von Sicherheitseigenschaften auf Basis des Kontrollflusses aber nicht des Datenflusses. Durch diese Fehlerklasse werden die Metriken zur Diskussion von *Z2.F1* beeinflusst.
- **FK4:** Beschreibt die Verletzung von Sicherheitseigenschaften durch Musterfehler, wodurch die Sicherheitseigenschaft einer `LinkingResource` entfernt wird und deshalb auch eine abgeleitet Sicherheitseigenschaft auf Aufrufabhängigkeit entfernt wird, die eigentlich nicht verletzt ist. Durch diese Fehlerklasse werden die Metriken zur Diskussion von *Z2.F2* beeinflusst.
- **FK5:** Beschreibt Musterfehler, welche keine Auswirkungen auf Sicherheitseigenschaften hat. Durch diese Fehlerklasse werden Metriken zur Diskussion von *Z1.F1*, *Z1.F2* und *Z2.F1* beeinflusst.

7.3.2 Umsetzung der Fehlerklassen durch Musterfehler

Die beschriebenen Fehlerklassen sollen durch Musterfehler im Quelltext realisiert werden. Deshalb sind in Tabelle 7.2 die Musterfehler, welche für die Evaluation eingebaut wurden, aufgeführt. Alle beschriebenen Fehler verletzen die «`encrypted`» Eigenschaft im Sinne der Typkorrespondenz. Die Musterfehler werden durch CogniCrypt gefunden, da hier die `Cipher` Klasse an den entsprechenden Stellen im Quelltext falsch zur Verschlüsselung verwendet wird. Die unterschiedlichen Fehler wurden so eingebaut, dass diese die in Tabelle 7.1 beschriebenen Sicherheitseigenschaften entweder im Sinne des Kontrollflusses, des Datenflusses oder beidem verletzen. Außerdem wurden Musterfehler eingebaut, die keine Sicherheitseigenschaften verletzen. Durch die aufgeführten Musterfehler werden die in der Tabelle 7.2 beschriebenen Fehlerklassen adressiert.
Die Auswirkungen dieser Fehler wird durch die Evaluation überprüft.

7.3.3 Beschreibung der Szenarien

Aus den beschriebenen Sicherheitseigenschaften, den Fehlerklassen, und den eingeführten Musterfehlern setzen sich die Evaluationsszenarien zusammen. Diese werden im Folgenden

ID	Quelltext Stelle	Verletzt (CG)	Verletzt (SDG)	FK
MF1	TravelPlanner.declassifiedCCD	SE1, SE2	SE1, SE2	FK1, FK2
MF2	TravelPlanner.declassifiedCCD	SE1, SE2	SE1, SE2	FK1, FK2
MF3	Airline.bookFlightOffer	SE3	-	FK3, FK4
MF4	Airline.bookFlightOffer	SE3	-	FK3, FK4
MF5	Airline.processBooking	-	-	FK5
MF6	Airline.processBooking	-	-	FK5

Tabelle 7.2: Beschreibt die Musterfehler (MF), die in den Quelltext eingebaut wurden und welche Sicherheitseigenschaften (SE) diese verletzen sollen. Bezieht sich sowohl auf den Datenfluss als auch den Kontrollfluss. Durch die Musterfehler werden die zuvor beschriebenen Fehlerklassen (FK) realisiert.

zusammengefasst. Dadurch wird der Zusammenhang zwischen Sicherheitseigenschaften der Architektur, Musterfehlern im Quelltext, den erwarteten Auswirkungen bei der Rückkopplung und den Forschungsfragen hergestellt.

- **Szenario 1 - Fehler verletzt «encrypted» Eigenschaft und es kommt zu weiteren Fehlern in der Architektursicherheitsanalyse:** Es wurden die Musterfehler *MF1* und *MF2* im Quelltext eingebaut, welche die *SE1* verletzen. Es kommt somit bei der Rückkopplung zur Löschung der «encrypted» Eigenschaft der `LR1`. Aufgrund der fehlenden «tamper-protection» wird erwartet, dass weitere Schwachstellen durch die Architektursicherheitsanalyse aufgedeckt werden. Dadurch wird *FK1* adressiert und die Forschungsfragen *Z1.F1, Z1.F2, Z2.F1* und *Z2.F2* können evaluiert werden.

- **Szenario 2 - Fehler verletzt «encrypted» Eigenschaft, es kommt aber nicht zu weiteren Fehlern in der Architektursicherheitsanalyse:** Die Musterfehler *MF1* und *MF2* im Quelltext verletzen die *SE2*. Dies führt dazu, dass die «encrypted» Eigenschaft von `LR2` gelöscht wird. Durch die «tamper-protection» sollen aber keine zusätzlichen Schwachstellen in der Architektursicherheitsanalyse gefunden werden. Dadurch wird *FK2* adressiert und die Forschungsfrage *Z1.F2* kann evaluiert werden.

- **Szenario 3 - Fehler verletzt «encrypted» Eigenschaft auf Basis des Kontrollflusses mit CG aber nicht auf Basis des Datenflusses mit SDG** Der Musterfehler *MF3* verletzt *SE3* auf Basis des Kontrollflusses, jedoch nicht auf Basis des Datenflusses. Bei einer Verletzung der *SE3* wird die «encrypted» Eigenschaft der `LR3` entfernt. Es werden dadurch mehr Fehler durch die Architektursicherheitsanalyse nach der gekoppelten Analyse auf Basis des CG erwartet. Dadurch wird *FK3* adressiert und die Forschungsfrage *Z2.F1* kann evaluiert werden.

- **Szenario 4 - «encrypted» Eigenschaft wird entfernt und es werden Sicherheitseigenschaften von Aufrufabhängigkeiten entfernt, die nicht betroffen sind** Der Musterfehler *MF3* führt bei der Kopplung auf Basis des CG dazu, dass

die «encrypted» Eigenschaft der LR3 entfernt wird. Durch die fehlende «encrypted» Eigenschaft der LR3 wird auch die *SE4* entfernt. Dadurch wird *FK4* adressiert und die Forschungsfrage *Z2.F2* kann evaluiert werden.

- **Szenario 5 - Fehler mit Typ, der die «encrypted» Eigenschaft verletzt, aber keine Auswirkungen auf die Eigenschaften der Architektur hat:** Die Musterfehler *MF5* und *MF6* verletzen zwar Eigenschaften mit dem Typ «encrypted» im Sinne der Typkorrespondenz, aber es gibt keine relevanten Verbindungen zu Sicherheitseigenschaften. Dadurch wird *FK5* adressiert und die Forschungsfragen *Z1.F1, Z1.F2* und *Z2.F1* können evaluiert werden.

7.4 Ergebnisse der Evaluation

Auf Basis der zuvor beschriebenen Evaluationsszenarien und des Systems für die Evaluation wurde eine Kopplung der Analysen durchgeführt. Im Folgenden werden nun die Ergebnisse dieser Kopplung präsentiert. Abschnitt 7.4.1 beschreibt die Ergebnisse der Durchführung der Kopplung. Abschnitt 7.4.2 beschreibt die Anzahl der gefundenen Fehler der Architektursicherheitsanalyse vor und nach der Kopplung.

7.4.1 Kopplungsausführung

Tabelle 7.3 gibt einen Überblick über die durchgeführte Kopplung der Analysen. Aus den drei LinkingResources der ResourceEnvironment des Testsystems ergeben sich auf Basis der Komponenten in den ResourceContainern die zuvor aufgeführten Sicherheitseigenschaften. Diese werden vollständig auf die erwarteten Kanten des Aufrufgraphen beziehungsweise auf die Methodenaufruf-Anweisungsknoten des Systemabhängigkeitsgraphen abgebildet. Die zuvor beschriebenen Musterfehler wurden ebenfalls vollständig abgebildet, also sowohl auf die Methodenknoten des Aufrufgraphen als auch die Anweisungsknoten des Systemabhängigkeitsgraphen. Es wurde außerdem die Anzahl der Musterfehler, welche gemäß der Kopplungvarianten auf Sicherheitseigenschaften abgebildet wurden, betrachtet. Die Kopplung auf Basis des CG findet die durch Musterfehler verursachten Verletzungen von Sicherheitseigenschaften. Die Kopplung auf Basis des SDG bildet die Musterfehler ebenfalls auf die Sicherheitseigenschaften ab, die durch diese verletzt werden. Dadurch werden die Szenarien 1 bis 5 umgesetzt und die verschiedenen Fehlerklassen 1 bis 5 treten ein. Für die Rückkopplung werden die zu den Sicherheitseigenschaften dazugehörigen «encrypted» Eigenschaften der entsprechenden LinkingResources entfernt. Die Ergebnisse der Analyse auf dem Architektursicherheitsmodell nach der Rückkopplung werden im nächsten Abschnitt beschrieben.

7.4.2 Schwachstellen in der Architektursicherheitsanalyse nach Durchführung der Kopplung

Um die Auswirkung der Kopplung der Analysen auf das Ergebnis der Architektursicherheitsanalyse zu betrachten, wurde diese sowohl vor als auch nach der Kopplung durchge-

Abbildung	CG (erw.)	CG (tatsäch.)	SDG (erw.)	SDG (tatsäch.)
Sicherheitseigenschaft auf Graph	SE1, SE2, SE3, SE4	SE1, SE2, SE3, SE4	SE1, SE2, SE3, SE4	SE1, SE2, SE3, SE4
Musterfehler auf Graph	MF1, MF2, MF3, MF4, MF5, MF6	MF1, MF2, MF3, MF4, MF5, MF6	MF1, MF2, MF3, MF4, MF5, MF6	MF1, MF2, MF3, MF4, MF5, MF6
Gefundene Verletzung von Musterfehler zu Sicherheitseigenschaft	MF1 → SE1, MF2 → SE1, MF1 → SE2, MF2 → SE2, MF3 → SE3, MF4 → SE3	MF1 → SE1, MF2 → SE1, MF1 → SE2, MF2 → SE2, MF3 → SE3, MF4 → SE3	MF1 → SE1, MF2 → SE1, MF1 → SE2, MF2 → SE2	MF1 → SE1, MF2 → SE1, MF1 → SE2, MF2 → SE2

Tabelle 7.3: Vergleich der erwarteten und tatsächlichen Abbildung von Sicherheitseigenschaften, Musterfehlern, und Musterfehlern auf Sicherheitseigenschaften. (erw. = erwartet; tatsäch. = tatsächlich)

führt. Die folgende Tabelle 7.4 zeigt, wie sich die Anzahl der vorhandenen `LinkingResources` mit «`encrypted`» Eigenschaften verändert hat. Die «`encrypted`» Eigenschaft geht verloren, da die aus der `LinkingResource` abgeleiteten Sicherheitseigenschaften eine Verletzung aufweisen. Außerdem wird die Anzahl der gefundenen Schwachstellen insgesamt, sowie die Anzahl der Schwachstellen in Verbindung mit den unterschiedlichen `LinkingResources` beschrieben. Die Modellierung des Architektursicherheitsmodells des TravelPlanner enthält auch ohne Kopplung bereits eine Anzahl von Schwachstellen, welche die Analyse identifiziert. Daher werden vor der Kopplung bereits Schwachstellen, unabhängig von den `LinkingResources`, gefunden.

Die Kopplung auf Basis des CG entfernt die «`encrypted`»-Eigenschaft der drei vorhandenen `LinkingResources`, da *SE1, SE2* und *SE3* verletzt sind. Die Kopplung auf Basis des SDG entfernt für zwei `LinkingResources` die «`encrypted`»-Eigenschaft, da *SE1* und *SE2* verletzt sind. Für die `LinkingResources` ohne «`tamper-protection`» kommt es zu weiteren Fehlern in der Architektursicherheitsanalyse (`LR1` und `LR3`).

7.5 Diskussion

Die in Abschnitt 7.1 beschriebenen Forschungsfragen werden in den Abschnitten 7.5.1 bis 7.5.4 diskutiert. Dadurch soll vermittelt werden, wie die erzielten Ergebnisse einzuordnen sind und welche Einschränkungen diese haben. Außerdem wird die Validität der durchgeführten Evaluation in Abschnitt 7.5.5 diskutiert.

	Vor Kopplung	Nach Kopplung (CG)	Nach Kopplung (SDG)
LR mit «encrypted»	LR1, LR2, LR3	-	LR3
Anzahl Verletzung Architekturanalyse (ges.)	130	142	138
Anzahl Verletzung Architekturanalyse LR1	0	8	8
Anzahl Verletzung Architekturanalyse LR2	0	0	0
Anzahl Verletzung Architekturanalyse LR3	0	4	0

Tabelle 7.4: LinkingResources mit «encrypted» Eigenschaft sowie Anzahl der gefundenen Fehler der Architektursicherheitsanalyse vor und nach der Kopplung.

7.5.1 Diskussion der Forschungsfrage Z1.F1

Wie Tabelle 7.3 zeigt, wurden für die beiden Varianten des Ansatzes die Sicherheitseigenschaften und Musterfehler erfolgreich auf die erwarteten Kanten beziehungsweise Knoten gemäß Tabelle 7.1 und Tabelle 7.2 des jeweiligen Graphen abgebildet. Die Abbildung der Eigenschaften und Fehler auf den Graphen ermöglicht die Kopplung. Ebenso wurden die eingebauten Verletzungen der Sicherheitseigenschaften durch Musterfehler, wie gemäß der Beschreibung in Tabelle 7.2 und den erzielten Ergebnissen in Tabelle 7.3 durch die Kopplung umgesetzt. Die gefundenen Verletzungen sorgen dafür, dass die dazugehörigen Sicherheitseigenschaften der LinkingResources wie in Tabelle 7.4 entfernt werden. Außerdem führen die Musterfehler *MF5* und *MF6*, wie erwartet, zu keinen Verletzungen von Sicherheitseigenschaften. Demnach kann geschlossen werden, dass mit dem in dieser Arbeit entwickelten Kopplungsansatz gefundene Musterfehler durch die Löschung von verletzten Sicherheitseigenschaften erfolgreich und korrekt in das Architekturmodell überführt werden können.

In der vorliegenden Arbeit wurde ausschließlich untersucht, ob Musterfehler mit sicheren Aufrufabhängigkeiten, definiert auf Architektursicht, gekoppelt werden können. Eine Kopplung mit anderen Sicherheitseigenschaften wurde bisher nicht betrachtet.

7.5.2 Diskussion der Forschungsfrage Z1.F2

Die in Tabelle 7.3 und Tabelle 7.4 präsentierten Ergebnisse zeigen, dass durch die Anwendung des Kopplungsansatzes und das damit verbundene Wegfallen von Sicherheitseigenschaften zusätzliche Fehler durch die Architektursicherheitsanalyse aufgedeckt werden.

Mit der Kopplung werden durch die CG Variante 12 Verletzungen mehr aufgedeckt. Durch die SDG Variante werden 8 Verletzungen mehr aufgedeckt. Demnach führen beiden Varianten dazu, dass mehr Schwachstellen aufgedeckt werden als ohne eine Kopplung. Unter den Umständen, dass ein `LinkingResource` durch andere Sicherheitsmaßnahmen geschützt ist (z.B. «`tamper-protection`»), kann das Wegfallen einer Sicherheitseigenschaft jedoch auch keine Auswirkung im Sinne von zusätzlichen erzeugten Verletzungen haben. In den Ergebnissen in Tabelle 7.4 ist dies ersichtlich. Hier verursacht das Wegfallen der `LR2` keine zusätzlichen Fehler, da diese durch die «`tamper-protection`» zusätzlich geschützt ist. Die Kopplung und Löschung von Sicherheitseigenschaften führen also nicht ausschließlich dazu, dass zusätzliche Fehler aufgedeckt werden können. Dies ist jedoch abhängig von der Modellierung einer Anwendung. Weiterhin führen Musterfehler wie MF5, die keine Sicherheitseigenschaften verletzen und somit nicht gekoppelt werden, nicht zu zusätzlichen Schwachstellen, die durch die Architektursicherheitsanalyse aufgedeckt werden.

7.5.3 Diskussion der Forschungsfrage Z2.F1

Die Kopplung mittels CG identifiziert wie in Tabelle 7.3 und Tabelle 7.4 dargestellt, drei Sicherheitseigenschaften auf der Basis von sechs Musterfehlern als verletzt. Als Folge dessen werden die «`encrypted`» Eigenschaften von allen drei `Linkingesources` aus dem Architektursicherheitsmodell gelöscht. Die Kopplung mittels SDG identifiziert hingegen nur zwei verletzte Sicherheitseigenschaften auf der Basis der selben sechs Musterfehlen. Infolgedessen werden aus dem Architekturmodell die «`encrypted`» Eigenschaften von zwei `Linkingesources` entfernt. Da durch die CG Variante mehr «`encrypted`» Eigenschaften gelöscht werden als mit der SDG Variante, findet die Architektursicherheitsanalyse hier mehr Fehler im modifizierten Architektursicherheitsmodell. Die durch den CG Ansatz zusätzlich als verletzt identifizierten Sicherheitseigenschaften beziehen sich auf einen Aufruf, der zwar ausgeführt wird, über den jedoch keine fehlerhaft verschlüsselten Daten übergeben werden. Es findet durch die Kopplung mittels CG demnach eine Überapproximation von verletzten Sicherheitseigenschaften statt. Wie erwartet bildet die Kopplung auf Basis des CG mehr Musterfehler auf Sicherheitseigenschaften ab, als die auf Basis des SDG.

7.5.4 Diskussion der Forschungsfrage Z2.F2

Die beiden Musterfehler *MF3* und *MF4* führen auf der Basis des CG zu einer Verletzung der Sicherheitseigenschaft *SE3*. Infolgedessen wird bei der Rückkopplung die «`encrypted`» Eigenschaft der dazugehörigen `LinkingResource 3` entfernt. Die Entfernung dieser Eigenschaft führt dazu, dass auch die daraus abgeleitete Aufrufabhängigkeit mit Sicherheitseigenschaft SE4 nicht mehr im Architektursicherheitsmodell existiert. Folglich kommt es hierbei in der Rückkopplung zu einer Überapproximation. Dieser Effekt würde äquivalent auch bei der Rückkopplung mit der SDG Variante zutreffen. Folglich kann eine Rückführung der verletzten Sicherheitseigenschaften mehr Aufrufabhängigkeiten mit Sicherheitseigenschaften entfernen, als tatsächlich verletzt sind. Dieses Problem ist spezifisch für die Umsetzung des Kopplungsansatz auf der Basis von Confidentiality4CBSE.

7.5.5 Gefahren für die Validität der Evaluation

Die interne Validität bezieht sich auf mögliche Fehler in der Evaluation welche dann auftreten, wenn ein zu untersuchender Faktor durch einen dritten, unbewusst vernachlässigten Faktor beeinflusst wird [43]. Im Sinne der internen Validität ist die Studie in dem Maß eingeschränkt, dass das System für Evaluation durch die Autorin selbst definiert wurde. Die Umsetzung dieses Systems schließt demnach das Wissen der Autorin mit ein. Dadurch besteht ein Risiko, dass bei der Umsetzung des Systems für die Evaluation implizite Annahmen getroffen wurden, die nicht formell für den Kopplungsansatz beschrieben wurden. Dadurch könnten Randfälle entstehen, die zu einem unerwarteten Kopplungsergebnis führen, aber die mit der durchgeführten Evaluation nicht aufgedeckt wurden. Weiterhin könnte eine andere Herangehensweise zur Realisierung der Kopplung mehr oder weniger Schwachstellen aufdecken, wie als in dieser Arbeit beschrieben. Die interne Validität könnte verbessert werden, indem das System für die Evaluation von einem unabhängigen Dritten festgelegt wird. Die externe Validität sagt aus, inwiefern sich die erzielten Ergebnisse generalisieren lassen [43]. Gemäß der externen Validität ist die durchgeführte Evaluation in dem Sinne eingeschränkt, dass der Kopplungsansatz nur anhand eines einzigen Systems mit dazugehörigem Architekturmodell und Quelltext überprüft wurde. Weiterhin wurden die präsentierte Varianten zur Kopplung nur für ein Paar von Quelltext- und Architekturanalyse realisiert. Demnach kann nicht ohne Einschränkungen darauf geschlossen werden, dass die präsentierten Eigenschaften der anwendbaren Analysen und des Kopplungsansatzes problemlos auf diese übertragen werden können. Die externe Validität der Evaluation könnte verbessert werden, indem weitere Testsysteme evaluiert werden und die Kopplung für andere Analysen erfolgreich realisiert werden würde.

8 Fazit und Ausblick

In der vorliegenden Arbeit wurden zwei Varianten zur Kopplung von Sicherheitseigenschaften auf Aufrufabhängigkeiten in der Architektursicht und Musterfehlern auf Quelltextsicht vorgestellt. Das entwickelt Konzept zur Kopplung wurde anhand eines Fallbeispiels evaluiert. Durch die realisierte Kopplung können Musterfehler, welche angenommene Sicherheitseigenschaften auf Aufrufabhängigkeiten der Architektursicht verletzen, in das Architektursicherheitsmodell zurückgeführt werden. Durch das Entfernen von verletzten Sicherheitseigenschaften kann durch die Kopplung mit der Architekturanalyse untersucht werden, wie sich die Verletzung der Eigenschaften auswirkt. So können erfolgreich zusätzliche Schwachstellen aufgedeckt werden. Dadurch werden die Informationen der Architektur- und Quelltextsicht miteinander kombiniert. Die Architekturanalyse liefert dadurch außerdem genauere Analyseergebnisse in Betracht auf das tatsächlich implementierte System einer Anwendung.

Im Detail wurden auf der Architektursicht Aufrufabhängigkeiten mit Sicherheitseigenschaften betrachtet. Dies sind genau solche Sicherheitseigenschaften, die sich auf den Aufruf einer Methode der einen Komponente durch eine andere beziehen. Es wurden die für die Kopplung vorausgesetzten Merkmale der Metamodelle der Architektursicherheitsanalyse beschrieben. Dies umfasst insbesondere das Architekturstrukturmetamodell und das Architektursicherheitsmetamodell. Ebenso wurden die Voraussetzungen der Metamodelle der Quelltextsicherheitsanalyse für die Kopplung beschrieben. Dazu gehören das Quelltext- und Quelltextanalyse Ergebnismetamodell. Es wurden auf Basis der definierten Voraussetzungen der beiden Sichten zwei Varianten zur Kopplung präsentiert. Die erste Variante nutzt die Informationen des Methoden-Kontrollflusses aus dem Quelltext, um zu bestimmen, ob eine Sicherheitseigenschaft durch einen Musterfehler verletzt wird. Dies wird mit Hilfe des Aufrufgraphen realisiert. Die zweite Variante verwendet hingegen den exakten Datenfluss aus dem Quelltext eines Programms, um eine Verletzung von Sicherheitseigenschaften durch Musterfehler zu ermitteln. Dieser Ansatz wird auf Basis des Systemabhängigkeitsgraphen realisiert. Für einen Ansatz zur Definition von Sicherheitseigenschaften, der auf die Kopplung angewendet werden soll, ist es notwendig, die Definition von Sicherheitseigenschaften auf die beschriebenen Metamodelle abzubilden. In der vorliegenden Arbeit wurde dies für den Ansatz Confidentiality4CBSE und die auf `LinkingResources` definierten Sicherheitseigenschaften vorgenommen. Ebenso wurde als konkreter Ansatz für die Identifikation von Musterfehlern im Quelltext CogniCrypt eingebunden. Die Analyseergebnisse von CogniCrypt entsprechenden den Metamodellen von anwendbaren Quelltextsicherheitsanalysen für die Kopplung.

Eine Evaluation der beiden Kopplungsvarianten wurde anhand eines beispielhaften Systems durchgeführt. In diesem System gibt es eine Reihe von Sicherheitseigenschaften welche auf Aufrufabhängigkeiten definiert sind. Es wurden dann systematisch Musterfehler eingebaut, um eine Reihe von möglichen Fehlerklassen und dazugehörige Forschungsfra-

gen zu adressieren. So konnte gezeigt werden, dass beide Kopplungsvarianten in der Lage sind Musterfehler mit Sicherheitseigenschaften zu koppeln. Das Entfernen von verletzten Sicherheitseigenschaften führt, je nach Eigenschaften des betrachteten Systems, zu weiteren Fehlern, die durch die Architekturanalyse aufgedeckt werden. Die Verwendung des Aufrufgraphen kann dazu führen, dass Sicherheitseigenschaften als verletzt identifiziert werden, obwohl keine musterfehlerbehafteten Daten über eine Aufrufabhängigkeit mit Sicherheitseigenschaft ausgetauscht werden. Demnach neigt der Ansatz auf Basis des Aufrufgraphen zu einer Überapproximation bei der Identifikation von verletzten Sicherheitseigenschaften und Ansatz auf Basis des Systemabhängigkeitsgraphen ist in dieser Hinsicht exakter.

Die größte Einschränkung der erzielten Evaluationsergebnisse ist, dass diese bisher nur exemplarisch anhand eines einzigen Testsystems evaluiert wurden. Zukünftige Arbeiten könnten deshalb zur Steigerung der Validität weitere Systeme für die Evaluation betrachten. Außerdem könnten weitere Ansätze zur Definition von Sicherheitseigenschaften und Identifikation von Musterfehlern eingebunden werden. Eine Einschränkung der präsentierten Kopplung ist, dass die Typen von Musterfehlern, welche Sicherheitseigenschaften eines bestimmten Typs verletzen, explizit formuliert werden müssen. Momentan beschränkt sich die Definition dieser Korrespondenz auf die «`encrypted`» Eigenschaft auf Architektursicht und Musterfehler im Zusammenhang mit der `Cipher`-Klasse auf Quelltextsicht. Weiterhin beschränkt sich der Kopplungsansatz nur auf Aufrufabhängigkeiten mit Sicherheitseigenschaften. Es wird hingegen nicht geprüft, ob Rückgabewerte von Aufrufabhängigkeiten die Sicherheitseigenschaften verletzen, oder ob es in der aufgerufenen Komponenten, etwa bei der Entschlüsselung, zu einer nachträglichen Verletzung von Sicherheitseigenschaften kommt. Die Umsetzung solcher Eigenschaften könnte sich auf Basis des Ansatzes mit Graphen erarbeiten lassen und so der bisherige Ansatz erweitert werden.

Literatur

[1] Identity Theft Resource Center. *U.S. data breaches and exposed records 2020*. en. 2021. URL: https://www.statista.com/statistics/273550/data-breaches-recorded-in-the-united-states-by-number-of-breaches-and-records-exposed/ (besucht am 05. 06. 2022).

[2] Robin Gandhi u. a. „Dimensions of Cyber-Attacks: Cultural, Social, Economic, and Political“. In: *IEEE Technology and Society Magazine* 30.1 (2011), S. 28–38. ISSN: 1937-416X. DOI: 10.1109/MTS.2011.940293.

[3] Istehad Chowdhury und Mohammad Zulkernine. „Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities?“ In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. SAC '10. New York, NY, USA: Association for Computing Machinery, 2010, S. 1963–1969. ISBN: 9781605586397. DOI: 10.1145/1774088.1774504. URL: https://doi.org/10.1145/1774088.1774504 (besucht am 05. 06. 2022).

[4] Amir Ahmadian u. a. *Model-based privacy and security analysis with CARiSMA*. Aug. 2017. DOI: 10.1145/3106237.3122823.

[5] Jürgen Graf, Martin Hecker und Martin Mohr. *Using JOANA for information flow control in Java programs - A practical guide*. en. Gesellschaft für Informatik e.V., 2013. ISBN: 9783885796091. URL: http://dl.gi.de/handle/20.500.12116/17361 (besucht am 06. 06. 2022).

[6] Stefan Krüger. „CogniCrypt – The Secure Integration of Cryptographic Software (Dissertation)“. Diss. Okt. 2020. URL: https://www.bodden.de/pubs/phdKrueger.pdf.

[7] Johannes Häring. *Enabling the Information Transfer between Architecture and Source Code for Security Analysis*. de. 2021. DOI: 10.5445/IR/1000142571. URL: https://publikationen.bibliothek.kit.edu/1000142571 (besucht am 25. 05. 2022).

[8] G. McGraw. „Software security“. In: *IEEE Security & Privacy* 2.2 (März 2004), S. 80–83. ISSN: 1558-4046. DOI: 10.1109/MSECP.2004.1281254.

[9] Nabil M. Mohammed u. a. „Exploring software security approaches in software development lifecycle: A systematic mapping study“. en. In: *Computer Standards & Interfaces* 50 (Feb. 2017), S. 107–115. ISSN: 0920-5489. DOI: 10.1016/j.csi.2016.10.001. URL: https://www.sciencedirect.com/science/article/pii/S0920548916301155 (besucht am 06. 06. 2022).

[10] Johannes Geismann, Bastian Haverkamp und Eric Bodden. „Ensuring threat-model assumptions by using static code analyses“. en. In: (), S. 10. URL: http://ceur-ws.org/Vol-2978/mde4sa-paper1.pdf.

[11] B. Selic. „The pragmatics of model-driven development“. In: *IEEE Software* 20.5 (Sep. 2003), S. 19–25. ISSN: 1937-4194. DOI: 10.1109/MS.2003.1231146.

[12] Aníbal Iung u. a. „Systematic mapping study on domain-specific language development tools“. en. In: *Empirical Software Engineering* 25.5 (Sep. 2020), S. 4205–4249. ISSN: 1573-7616. DOI: 10.1007/s10664-020-09872-1. URL: https://doi.org/10.1007/s10664-020-09872-1 (besucht am 17. 06. 2022).

[13] Herbert Stachowiak. *Allgemeine Modelltheorie*. de. Springer, 1973. ISBN: 9783211811061.

[14] Nenad Medvidovic u. a. „Modeling software architectures in the Unified Modeling Language“. In: *ACM Transactions on Software Engineering and Methodology* 11.1 (Jan. 2002), S. 2–57. ISSN: 1049-331X. DOI: 10.1145/504087.504088. URL: https://doi.org/10.1145/504087.504088 (besucht am 17. 06. 2022).

[15] Ralf Reussner u. a. *The Palladio Component Model*. de. 2011. DOI: 10.5445/IR/1000022503. URL: https://publikationen.bibliothek.kit.edu/1000022503 (besucht am 04. 10. 2022).

[16] Max E. Kramer u. a. *Model-Driven Specification and Analysis of Confidentiality in Component-Based Systems*. de. 2017. DOI: 10.5445/IR/1000076957. URL: https://publikationen.bibliothek.kit.edu/1000076957 (besucht am 03. 10. 2022).

[17] *Find Security Bugs*. URL: https://find-sec-bugs.github.io/ (besucht am 02. 06. 2022).

[18] Jan Jürjens. „UMLsec: Extending UML for Secure Systems Development“. en. In: *UML 2002 — The Unified Modeling Language*. Hrsg. von Jean-Marc Jézéquel, Heinrich Hussmann und Stephen Cook. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2002, S. 412–425. ISBN: 978-3-540-45800-5. DOI: 10.1007/3-540-45800-X_32.

[19] Torsten Lodderstedt, David Basin und Jürgen Doser. „SecureUML: A UML-Based Modeling Language for Model-Driven Security“. In: Springer, 2002, S. 426–441.

[20] *PreReqSec*. original-date: 2017-12-18T12:55:06Z. Sep. 2019. URL: https://github.com/kit-sdq/PreReqSec (besucht am 25. 05. 2022).

[21] B. Chess und G. McGraw. „Static analysis for security“. In: *IEEE Security & Privacy* 2.6 (Nov. 2004), S. 76–79. ISSN: 1558-4046. DOI: 10.1109/MSP.2004.111.

[22] Stephan Lipp, Sebastian Banescu und Alexander Pretschner. „An Empirical Study on the Effectiveness of Static C Code Analyzers for Vulnerability Detection“. en. In: *South Korea* (2022), S. 12. URL: https://mediatum.ub.tum.de/doc/1659728/1659728.pdf.

[23] *checkstyle – Checkstyle 10.3*. URL: https://checkstyle.sourceforge.io/ (besucht am 09. 06. 2022).

[24] *Source Code Analysis Tools | OWASP Foundation*. en. URL: https://owasp.org/www-community/Source_Code_Analysis_Tools (besucht am 25. 05. 2022).

[25] Yuanyuan Pan. „Interactive Application Security Testing“. In: *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)*. Aug. 2019, S. 558–561. DOI: 10.1109/ICSGEA.2019.00131.

[26] Fabian Yamaguchi. „Pattern-Based Vulnerability Discovery“. eng. In: (Nov. 2015). DOI: 10.53846/goediss-5356. URL: https://ediss.uni-goettingen.de/handle/11858/00-1735-0000-0023-9682-0 (besucht am 22. 06. 2022).

[27] Stefan Kruger u. a. „CrySL: An Extensible Approach to Validating the Correct Usage of Cryptographic APIs“. en. In: *IEEE Transactions on Software Engineering* 47.11 (Nov. 2021), S. 2382–2400. ISSN: 0098-5589, 1939-3520, 2326-3881. DOI: 10.1109/TSE.2019.2948910. URL: https://ieeexplore.ieee.org/document/8880510/ (besucht am 25. 05. 2022).

[28] *Crypto-API-Rules*. original-date: 2017-05-21T14:48:59Z. Apr. 2022. URL: https://github.com/CROSSINGTUD/Crypto-API-Rules (besucht am 27. 05. 2022).

[29] Eclipse Foundation. *The CrySL Language | CogniCrypt*. en. URL: https://www.eclipse.org/cognicrypt/documentation/crysl/ (besucht am 09. 06. 2022).

[30] W. T. Tutte und William Thomas Tutte. *Graph Theory*. en. Google-Books-ID: uTGhooU37h4C. Cambridge University Press, Jan. 2001. ISBN: 9780521794893.

[31] Karl J. Ottenstein und Linda M. Ottenstein. „The program dependence graph in a software development environment“. In: *Proceedings of the first ACM SIGSOFT/SIGPLAN software engineering symposium on Practical software development environments*. SDE 1. New York, NY, USA: Association for Computing Machinery, Apr. 1984, S. 177–184. ISBN: 9780897911313. DOI: 10.1145/800020.808263. URL: https://doi.org/10.1145/800020.808263 (besucht am 03. 10. 2022).

[32] N. Walkinshaw, M. Roper und M. Wood. „The Java system dependence graph“. In: *Proceedings Third IEEE International Workshop on Source Code Analysis and Manipulation*. Sep. 2003, S. 55–64. DOI: 10.1109/SCAM.2003.1238031.

[33] Ranjan Kumar, Subhrakanta Panda und Durga Prasad Mohapatra. „Analysis of Java programs using Joana and Java SDG API“. In: *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. Aug. 2015, S. 2402–2408. DOI: 10.1109/ICACCI.2015.7275978.

[34] Michael Langhammer u. a. „Automated Extraction of Rich Software Models from Limited System Information“. In: *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. Apr. 2016, S. 99–108. DOI: 10.1109/WICSA.2016.35.

[35] Francesc Mateo Tudela u. a. „On Combining Static, Dynamic and Interactive Analysis Security Testing Tools to Improve OWASP Top Ten Security Vulnerability Detection in Web Applications“. en. In: *Applied Sciences* 10.24 (Jan. 2020). Number: 24 Publisher: Multidisciplinary Digital Publishing Institute, S. 9119. DOI: 10.3390/app10249119. URL: https://www.mdpi.com/2076-3417/10/24/9119 (besucht am 07. 05. 2021).

[36] Christopher Gerking u. a. „Domain-Specific Model Checking for Cyber-Physical Systems“. en. In: (), S. 10.

[37] Sven Peldszus u. a. „Secure Data-Flow Compliance Checks between Models and Code Based on Automated Mappings“. In: *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*. Sep. 2019, S. 23–33. DOI: 10.1109/MODELS.2019.00-18.

[38] Marwan Abi-Antoun, Daniel Wang und Peter Torr. „Checking threat modeling data flow diagrams for implementation conformance and security“. In: *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ASE '07. New York, NY, USA: Association for Computing Machinery, Nov. 2007, S. 393–396. ISBN: 9781595938824. DOI: 10.1145/1321631.1321692. URL: https://doi.org/10.1145/1321631.1321692 (besucht am 04. 08. 2022).

[39] Katja Tuma u. a. *Checking security compliance between models and code | SpringerLink*. Feb. 2022. URL: https://link.springer.com/article/10.1007/s10270-022-00991-5 (besucht am 30. 06. 2022).

[40] Dorothy E. Denning. „A lattice model of secure information flow“. In: *Communications of the ACM* 19.5 (1976), S. 236–243. ISSN: 0001-0782. DOI: 10.1145/360051.360056. URL: https://doi.org/10.1145/360051.360056 (besucht am 19. 06. 2022).

[41] Dimitrios Michail u. a. „JGraphT—A Java Library for Graph Data Structures and Algorithms“. en. In: *ACM Transactions on Mathematical Software* 46.2 (Juni 2020), S. 1–29. ISSN: 0098-3500, 1557-7295. DOI: 10.1145/3381449. URL: https://dl.acm.org/doi/10.1145/3381449 (besucht am 06. 10. 2022).

[42] *Modeling the Travel Planner Application with IFlow*. URL: https://kiv.isse.de/projects/iflow/TravelPlannerSite/index.html (besucht am 31. 10. 2022).

[43] Per Runeson u. a. *Case Study Research in Software Engineering: Guidelines and Examples*. en-us. März 2012. URL: https://www.wiley.com/en-us/Case+Study+Research+in+Software+Engineering%3A+Guidelines+and+Examples-p-9781118181003.