

Datenflussmodellierung für Vertraulichkeitsanalysen in Palladio

Bachelorarbeit von

Thomas Czogalik

an der Fakultät für Informatik
Institut für Programmstrukturen und Datenorganisation (IPD)

Erstgutachter:

Prof. Dr. Ralf H. Reussner

Zweitgutachter:

Jun.-Prof. Dr.-Ing. Anne Koziolek

Betreuernder Mitarbeiter:

M. Sc. Stephan Seifermann

Zweiter betreuender Mitarbeiter: Dipl.-Inform. Emre Taşpolatoğlu

01. Juni 2016 – 30. September 2016

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, 30.09.2016

.....
(Thomas Czogalik)

Zusammenfassung

Das Interesse, dass vertrauliche Daten in IT-Systemen geschützt sind, ist sehr hoch, da Datenlecks hohe Kosten verursachen können und die Reputation eines Unternehmens nachhaltig schädigen können. Ein solches System sicher zu entwerfen, ist allerdings schwierig, da die Bewertung, wie gut ein System abgesichert ist, häufig nur durch Experten möglich ist, weil Sicherheitsaspekte nicht in geeigneter Form dokumentiert sind und die Auswirkung einer Änderung auf das Gesamtsystem schwer festzustellen ist. Bei der Bestimmung von Vertraulichkeitseigenschaften können Datenflussanalysen helfen. Verbreitete Datenflussanalysen weisen aber einige Schwächen auf. Zum Einen werden Daten und Datenflüsse erst in der Implementierung analysiert. Fehler können somit erst in der Implementierung entdeckt werden und können zu aufwendigen Änderungen in der Implementierung und Architektur führen. Zum Anderen leiten diese Analysen Datenflüsse aus dem Kontrollfluss ab oder spezifizieren diese nur indirekt über Parameterübergaben. Sicherheitsanalysen werden dadurch komplexer und damit auch unverständlicher für deren Nutzer. In dieser Bachelorarbeit werden Daten und Datenflüsse als Objekte erster Klasse, auf Architekturebene eingeführt. Datenflüsse werden sowohl aus Nutzersicht, als auch aus technischer Sicht betrachtet und dokumentierbar. Durch Flexible Eigenschaften ist es möglich, vielfältige Analysen zu nutzen. Außerdem werden bestehende Architekturmodelle im Palladio-Komponenten-Modell (PCM) erweitert. Zur Validierung wurde ein bekanntes Fallstudiensystem modelliert, um Datenflüsse erweitert und die dadurch entstandenen Eingabedaten für eine Analyse mit existierenden Daten abgeglichen. Der Vergleich zeigt, dass der Modellierungsansatz für Vertraulichkeitsanalysen geeignet ist und zu sinnvollen Ergebnissen führt.

Inhaltsverzeichnis

Zusammenfassung	i
1. Einleitung	1
1.1. Ziel	2
1.2. Aufbau	2
2. Grundlagen	3
2.1. Modellgetriebene Software-Entwicklung	3
2.1.1. Modell	3
2.1.2. Meta-Modell	4
2.2. Rollen der komponentenbasierten Software-Entwicklung	4
2.3. Palladio-Komponentenmodell	5
2.3.1. Verwendete Teilmodelle	5
2.3.2. Service-Effect-Specification	6
3. Verwandte Arbeiten	9
3.1. Vertraulichkeitsbegriffsbildung	9
3.2. Datenflussmodellierung in Palladio	9
3.3. Datenflussdiagramm	10
3.4. Informationsflussanalyse	11
3.5. Erweiterung des PCM	13
4. Modellierung	15
4.1. Anforderungserhebung	15
4.2. Übersicht der Pakete	17
4.3. Beispielszenario	18
4.4. Software-Verteilungsexperte	19
4.5. Datenmodellierer	21
4.6. Komponentenentwickler	23
4.7. Domänenexperte	26
5. Validierung	29
5.1. Fallbeispiel	31
5.1.1. Beschreibung der PCM-Modelle	31
5.1.2. Vertraulichkeitsmodell	34
5.1.3. Datenflussmodellierung	37
5.1.4. Angreifermodell	40
5.2. Transformation und Datenflussanalyse	41

Inhaltsverzeichnis

5.3. Ergebnisse	43
6. Zusammenfassung und Ausblick	47
Literatur	49
A. Anhang	51

Abbildungsverzeichnis

1.1.	System, das vertrauliche Daten vom Benutzer verarbeitet	1
2.1.	Beziehung zwischen Original, Modell und Meta-Modell	4
2.2.	Beziehung zwischen den Rollen und den Submodellen des PCM	6
2.3.	Beispiel SEFF als endlicher Automat	7
2.4.	RDSEFF, der den Download von Audio-Dateien aus dem Palladio Media-Store modelliert	8
3.1.	Beispiel für ein Datenflussdiagramm	10
3.2.	Beispiel für Constant-Propagation	11
3.3.	Beispiel eines CFG, für das Beispiel aus Abbildung 3.2	12
4.1.	Übersicht der Pakete der Meta-Modell-Erweiterung	18
4.2.	Das Laden-Beispiel-Szenario	18
4.3.	Der Ablauf eines Basis-Kaufvorgangs im Laden	19
4.4.	Übersicht der Erweiterung für das Ressourcen-Umgebungs-Modell	20
4.5.	Übersicht der Eigenschaften für Resource-Container und Linking-Resource	20
4.6.	Resource-Container Beispiel des Laden-Szenarios	21
4.7.	Linking-Resource Beispiel des Laden-Szenarios	22
4.8.	Übersicht der Datenmodellierung	22
4.9.	Übersicht der DFSEFF Erweiterung	24
4.10.	Übersicht der Aktionen des DFSEFF	25
4.11.	Übersicht der Modellierung von Bindings	25
4.12.	Datenfluss des Dienstes <code>buy(Product product)</code> aus dem Laden-Szenario	26
4.13.	Modellierung der Erweiterung für das Nutzungsmodell	27
4.14.	Datenfluss, der vom Benutzer ausgeht	28
5.1.	Buchung eines Fluges	32
5.2.	Komponenten-Repository-Modell des Travelplanner	33
5.3.	Ressource-Umgebungs-Modell des Travelplanner	34
5.4.	Nutzungsmodell des Travelplanner	35
5.5.	Stereotypen, mit denen das PCM um Elemente, aus dem Vertraulichkeitsmodell, erweitert werden kann	36
5.6.	Erweiterung von <code>MobilePhone</code> und <code>4G+Internet</code> um einen Standort	37
5.7.	Erweiterung des Nutzungsmodells um Bindings	37
5.8.	DFSEFF des Aufrufs <code>BookingSelection.getFlightOffers</code>	38
5.9.	DFSEFF des Aufrufs <code>TravelAgency.getFlightOffers</code>	39
5.10.	DFSEFF des Aufrufs <code>BookingSelection.releaseCCD</code>	39
5.11.	DFSEFF des Aufrufs <code>Declassification.declassifyCCD</code>	39

Abbildungsverzeichnis

5.12. DFSEFF des Aufrufs BookingSelection.bookSelected	40
5.13. DFSEFF des Aufrufs Airline.bookFlight	40
5.14. Angreifermodell der Modellinstanz, nachdem sie transformiert wurde . .	41
5.15. Beispiel Datenfluss für die Transformation	42
A.1. Sequenzdiagramm zur Buchung eines Fluges nach [20].	51
A.2. Komponenten-Repositoy-Modell des Travelplanner nach [9]. Markierte Elemente wurden entfernt	52
A.3. Systemmodell des Travelplanner	53
A.4. Komponenten-Allokations-Modell des Travelplanner	53

Tabellenverzeichnis

5.1.	Ziele, Fragen und Metriken für die Validierung, nach der GQM-Methode	30
5.2.	Antworten zu den Fragen aus Tabelle 5.1	43
5.3.	Unterschiede der Zuordnung von Parameter in DataSets	45

Glossar

BVERFG Bundesverfassungsgericht.

CFG Kontrollflussgraph.

DFSEFF Datenfluss-Service-Effect-Specification.

EU-DSGVO Europäischen-Datenschutzgrundverordnung.

PCM Palladio-Komponentenmodell.

RDSEFF Resource-Demanding-Service-Effect-Specification.

SEFF Service-Effect-Specification.

1. Einleitung

Heutige IT-Systeme werden immer komplexer und verarbeiten immer mehr vertrauliche Daten. Dabei ist das Interesse, dass vertrauliche Daten geschützt sind, sehr hoch. Der Grund dafür ist, dass ein Datenleck hohe Kosten verursachen kann. Die von IBM und Ponemon Institute gesponserte Studie **Cost of Data Breach Study: Global Analysis** [15] veröffentlichte 2015 die Kosten, die ein gestohlener bzw. verlorener Datensatz verursacht. Bei den betrachteten Datensätzen handelt es sich um Kundendaten. In Deutschland betrug der Schaden dabei durchschnittlich 211 \$. 2014 lag der Schaden noch bei 195 \$. Durch den Anstieg der Kosten, entsteht ein finanzielles Interesse bei den Unternehmen, dass die Daten auf ihren Systemen geschützt sind.

Der Schutz der Daten ist eine große Herausforderung. Außerdem ist eine Bewertung, ob ein existierendes oder geplantes System ausreichend abgesichert ist, schwierig, da schwer zu ermitteln ist, wo relevante Daten in komplexen Systemen verarbeitet werden.

In Abbildung 1.1 ist ein System abgebildet. Dieses System besteht aus mehreren Komponenten, die auch an verschiedenen Orten liegen können. Außerdem gibt es einen Benutzer, der vertrauliche Daten in das System eingibt, z.B. seine Kreditkarteninformationen. Nun wird eine der Komponenten ausgelagert, z.B. an einen Drittanbieter. Dieser soll, bzw. darf an keine der vertraulichen Daten gelangen. Das heißt, dass die ausgelagerte Komponente nicht mehr an die vertraulichen Daten gelangen darf. Um das jedoch sicherzustellen, bzw. zu erkennen benötigt es einer Datenflussanalyse, die den Fluss von Daten nachvollzieht. Das Problem ist, dass verbreitete Datenflussanalysen, die Implementierung der

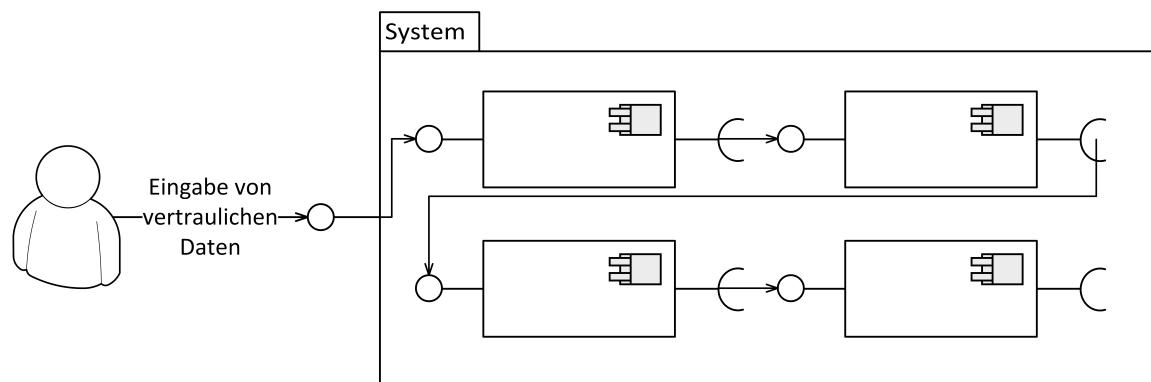


Abbildung 1.1.: System, das vertrauliche Daten vom Benutzer verarbeitet

Anwendung als Eingabe nutzen. Fehler können dadurch erst nach der Implementierung erkannt und nicht bereits vorzeitig beseitigt werden. Ist ein Fehler durch eine mangelhafte Architektur entstanden, muss ggf. sogar ein Großteil der Implementierung verworfen werden. Dies kann zu hohen Kosten führen, da nicht nur die Implementierung sondern

1. Einleitung

auch die Architektur repariert werden muss. Eine Datenflussanalyse auf Architekturebene hätte den Vorteil, dass Fehler bereits frühzeitig erkannt werden können, wie z.B. in dem Fall der IP-Telefone von Snom [7]. Dabei war es möglich, bei allen IP-Telefone des Herstellers, von außen eine neue Firmware aufzuspielen. Eine Datenflussanalyse hätte beim Entwurf helfen können zu überprüfen, wohin Nutzereingaben fließen. Ein weiteres Beispiel, ist der Fall eines Auto-Hacks, bei dem ein Jeep Cherokee aus der Ferne gehackt und ferngesteuert werden konnte [5]. Eine Datenflussanalyse hätte geholfen festzustellen, dass Komponenten beim Entwurf nicht sauber getrennt wurden.

Es ist schwierig, solche Analysen zu entwickeln und für den Nutzer, die dafür notwendigen Eingabedaten zu formulieren. Dies geht leichter über Datenflüsse. Ansätze dafür sind in Entwicklung, aber erlauben die Spezifikation von Datenflüssen bisher nur indirekt, z.B. über Parameterübergabe.

1.1. Ziel

Das Problem ist, dass es keine Datenflussdokumentation auf Architekturebene gibt. Daten können nur indirekt über Parameter identifiziert werden. Das führt dazu, dass Analysen nur eingeschränkt möglich sind, z.B. dadurch, dass Datenflüsse nicht innerhalb von Komponenten nachvollzogen werden können. Das ist soweit ein Problem, dass viele Sicherheitsanalysen schwer über den indirekten Datenfluss ausdrückbar sind.

Die Idee ist, Daten und Datenflüsse als Objekte erster Klasse auf Architekturebene einzuführen. Dazu soll eine Modellierung entwickelt werden, die es ermöglicht Daten und Datenflüsse auf Architekturebene auszudrücken. Die Modellierung soll dabei als Eingabe in Datenflussanalysen eingesetzt werden. Im Rahmen der Bachelorarbeit soll der Fokus zunächst auf Vertraulichkeitsanalysen in Palladio liegen. In nachgelagerten Arbeiten sollen weitere Analysen betrachtet werden.

Der Vorteil dabei ist, dass weitere Qualitätsattribute in Palladio analysierbar werden. Außerdem ist der Dokumentationsaufwand geringer. Alternativ müsste ein komplett neues Modell erstellt werden. Mit dem hier gewählten Ansatz können aber Palladio-Modelle wiederverwendet werden.

1.2. Aufbau

Im Folgenden wird der Aufbau der Arbeit beschrieben. In Kapitel 2 werden Grundlagen erläutert, die für die weitere Arbeit benötigt werden. In Kapitel 3 wird ein Überblick über themenverwandte Arbeiten gegeben. Sie werden kurz zusammengefasst und diskutiert, in wie weit sie für die Bachelorarbeit relevant sind, bzw. wie sich die Bachelorarbeit abgrenzt. Kapitel 4 beschreibt eine Modellierung, mit der es möglich ist Daten und Datenflüsse in Palladio zu modellieren. Diese Modellierung wird in Kapitel 5 mithilfe eines Fallbeispiels und einer Vertraulichkeitsanalyse für Palladio validiert. Schließlich wird in Kapitel 6 die Arbeit zusammengefasst und ein Ausblick für zukünftige Arbeiten gegeben.

2. Grundlagen

Die Bachelorarbeit setzt konzeptionell im Wesentlichen auf drei existierenden Grundlagen auf. Dabei soll zunächst eine Erweiterung der Architekturbeschreibungssprache Palladio [2] erstellt werden, die das Paradigma der komponentenbasierten Entwicklung [8] und ihr Rollenverständnis zugrunde legt. Der Gesamtansatz lässt sich in den Bereich der modellgetriebenen Entwicklung [19] einordnen. In den folgenden Abschnitten werden diese Konzepte kurz erläutert.

2.1. Modellgetriebene Software-Entwicklung

In der modellgetriebenen Software-Entwicklung [19] werden Teile des Software-Systems innerhalb von Modellen unter Nutzung von Abstraktionen beschrieben. Modelle sind Entwicklungsartefakte und Systembestandteile oder erlauben das automatische Ableiten von Systemartefakten. Das Ziel der modellgetriebenen Softwareentwicklung ist, die Verbesserung der Softwarequalität und der Wiederverwendbarkeit. Außerdem soll eine Erhöhung der Entwicklungseffizienz, zum Beispiel durch Quellcodeerzeugung, aus den Modellen erreicht werden.

2.1.1. Modell

Nach Herbert Stachowiak [17] zeichnet sich ein Modell durch die folgenden drei Merkmale aus.

- **Abbildung** - Bei einem Modell handelt es sich um eine Abbildung oder Repräsentation von einem künstlichen oder natürlichen Original. Das Original kann selbst wieder ein Modell sein.
- **Verkürzung** - Es werden nicht alle Attribute des Originals erfasst, sondern nur diejenigen, die relevant sind.
- **Pragmatismus** - Modelle erfüllen ihre Ersatzfunktion für bestimmte Subjekte, innerhalb einer bestimmten Zeit und unter Einschränkung bestimmter gedanklicher oder tätlicher Operationen. Beispielsweise benötigt eine Analyse je nach gewünschter Genauigkeit, unterschiedlich genaue Modelle. Der Pragmatismus schreibt vor, dass man hier nicht unnötig genau arbeitet.

Ein Beispiel der Beziehung zwischen Original und Modell ist in Abbildung 2.1 dargestellt. Es handelt sich dabei um eine Audio-Datei, die das Original darstellt. Das dazugehörige Modell enthält ein Element mit dem Namen *song:Song*. Es besitzt die Attribute: Titel: *Stairway to Heaven*, Sänger: *Led Zeppelin* und Jahr: 1971. Der Einsatzzweck dieser Modellierung ist z.B. eine Musikdatenbank.

2. Grundlagen

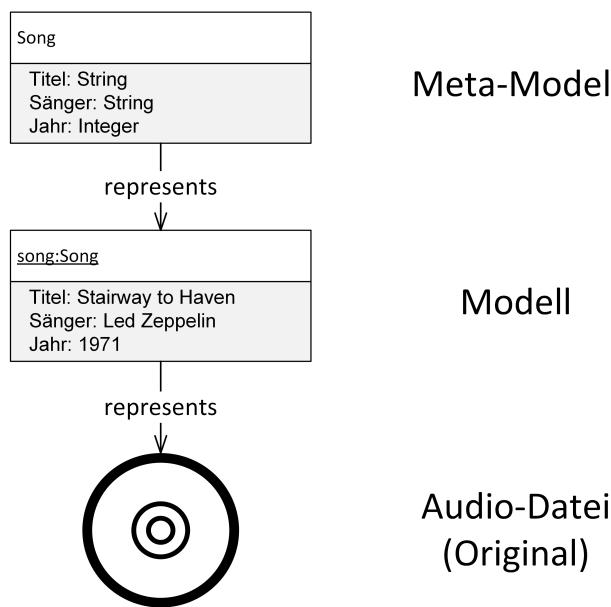


Abbildung 2.1.: Beziehung zwischen Original, Modell und Meta-Modell

2.1.2. Meta-Modell

Ein Meta-Modell beschreibt die Struktur eines Modells. In dem Musik-Beispiel aus Abbildung 2.1, enthält das Meta-Modell eine Klasse **Song** mit den Attributen **Titel**, **Sänger**, vom Typ String und **Jahr** vom Typ Integer. Ein Meta-Modell muss folgende Bereiche abdecken:

- **Abstrakte Syntax:** Sie beschreibt die Konstrukte, aus denen die Modelle bestehen, sowie deren Eigenschaften und Beziehungen.
- **Konkrete Syntax:** Diese beschreibt die Darstellung der Konstrukte, Eigenschaften und Beziehungen, die in der abstrakten Syntax spezifiziert sind.
- **Statische Semantik:** Mithilfe dieser werden Modellierungsregeln und Einschränkungen ausgedrückt, die mit der abstrakten Syntax nicht möglich sind.
- **Dynamische Semantik:** Sie drückt die Bedeutung der Konstrukte aus und wird oft nicht formal, sondern durch natürlich sprachlichen Text spezifiziert.

2.2. Rollen der komponentenbasierten Software-Entwicklung

In der komponentenbasierten Softwareentwicklung werden Software-Architektur-Modelle verwendet. Dabei gibt es vier verschiedene Rollen [8], die an unterschiedlichen Teilen der Architektur arbeiten und dort ihr spezifisches Wissen einbringen.

- **Systemarchitekt:** Die Architektur und der Zusammenhang der einzelnen Komponenten miteinander werden vom Systemarchitekten entworfen. Er ist auch dafür zuständig, weitere Anweisungen an die anderen Rollen weiterzugeben.
- **Domänenxperte:** Das Wissen darüber, wie der Benutzer mit dem System interagiert und welche Parameter im Kontrollfluss verwendet werden, stammt vom Domänenxperten.
- **Komponentenentwickler:** Für das Implementieren und Spezifizieren der einzelnen Komponenten, ist der Komponentenentwickler verantwortlich.
- **Software-Verteilungsexperte:** Die Zusammenstellung der Systemumgebung der Software wird vom Softwareverteilungsexperten übernommen. Außerdem weist der Software-Verteilungsexperte den Ressourcen Komponenten zu.

2.3. Palladio-Komponentenmodell

Das Palladio-Komponentenmodell (PCM) [2] ist eine Architecture-Description-Language (ADL). Sie wird verwendet, um komponentenbasierte Software zu beschreiben. Neben der Beschreibung sind auch Qualitätsanalysen, wie bezüglich der Performanz oder Zuverlässigkeit möglich. Im Folgenden werden Teile des PCM erklärt, die für die Bachelorarbeit wichtig sind.

2.3.1. Verwendete Teilmodelle

Das PCM unterscheidet ebenfalls in vier verschiedene Rollen. Diese entsprechen den Rollen aus Abschnitt 2.2. Der einzige Unterschied ist dabei, dass im PCM der Systemarchitekt Software-Architekt heißt. Die Aufgaben sind jedoch dieselben. Jede dieser Rollen ist außerdem für ein bestimmtes Submodell zuständig. Die Summe aller Teilmodelle ergibt dann die vollständige Architekturbeschreibung. In Abbildung 2.2 sind die Beziehungen zwischen den Rollen und den Submodellen dargestellt.

Für das **Komponenten-Repository-Modell** (Component-Repository-Model) ist der Komponentenentwickler zuständig. Dieses Modell enthält die einzelnen Komponenten und Schnittstellen. Der Software-Architekt ist für das **Systemmodell** (System-Model) zuständig, das die Zusammensetzung der Komponenten charakterisiert. Der Softwareverteilungsexperte ist für gleich zwei Modelle verantwortlich. Das erste Modell ist das **Ausführungs-Umgebungs-Modell** (Execution-Environment-Model). Es beschreibt die benutzte Hardware und das Netzwerk. Das zweite Modell beschreibt wie die einzelnen Komponenten auf der Hardware verteilt werden. Dieses Modell ist das **Komponenten-Allokations-Modell** (Component-Allocation-Model). Schließlich ist der Domänenxperte für das **Nutzungsmodell** (Usage-Model) zuständig. Dieses beschreibt die Interaktion des Benutzers mit dem System.

Diese Modelle werden im Rahmen der Bachelorarbeit entsprechend um die Möglichkeit zur Modellierung von Datenflüsse und ggf. weiteren für die Vertraulichkeitsanalyse notwendigen Eigenschaften erweitert.

2. Grundlagen

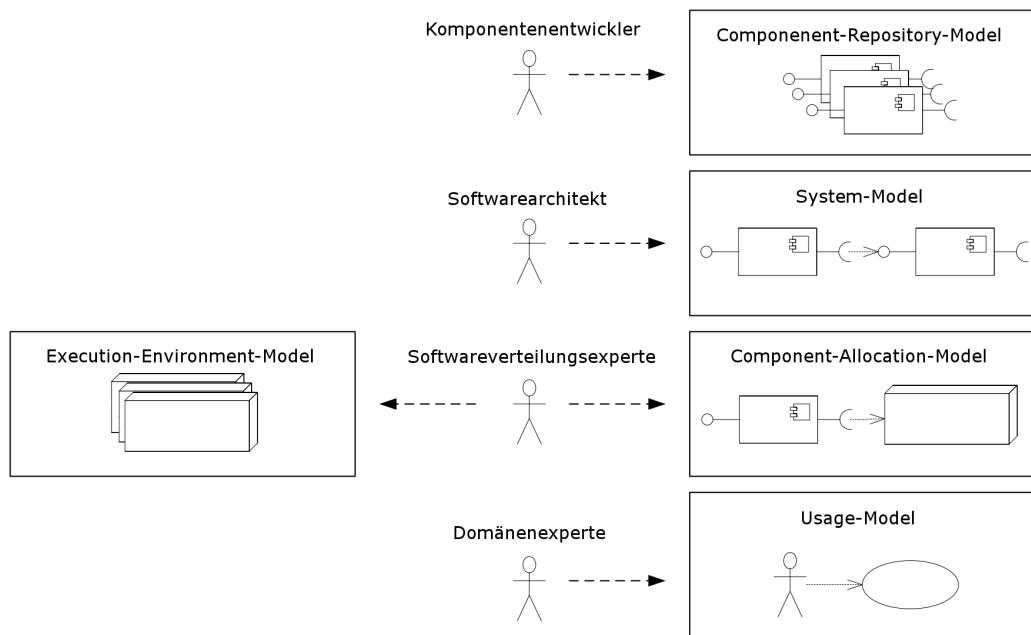


Abbildung 2.2.: Beziehung zwischen den Rollen und den Submodellen des PCM

2.3.2. Service-Effect-Specification

Mithilfe der Service-Effect-Specification (SEFF) [16], kann das Verhalten innerhalb der Komponenten, auf abstrakten und für bestimmte Qualitätsattribute zugeschnittenen Spezifizierungs-/Detailgrade, beschrieben werden. Durch das Spezifizieren von SEFFs kann die Software auf Architekturebene analysiert werden. Dabei gibt es drei verschiedene Abstraktionslevel, die aufeinander aufbauen:

- **Signature-List-Based-Interface:** Schnittstellen dienen zur Kommunikation zwischen den Komponenten. Auf dem Signature-List-Based-Interface bauen die anderen Abstraktionslevel auf. Es besteht aus Parametern und Rückgabewerten.
- **Protocol-Modeling-Interface:** Mithilfe dieser Schnittstelle lassen sich Aufrufsequenzen definieren.
- **Quality-Of-Service-Modelling Interface:** Hier können verschiedene Qualitätsattribute definiert werden.

Für Qualitätsanalysen, kann der SEFF in einen endlichen Automaten transformiert werden. Dabei wird ein Aufruf zu einer Komponente als Zustandsübergang modelliert. In Abbildung 2.3 ist ein SEFF als endlicher Automat dargestellt. Das Beispiel stammt aus [16]. Der SEFF modelliert dabei den Aufruf `TagWatermarking.download`. Innerhalb des Aufrufs wird `IDownload.download`, in einer anderen Komponente aufgerufen.

Der Resource-Demanding-Service-Effect-Specification (RDSEFF) ist eine Erweiterung zum SEFF. Die Performanzvorhersage für Komponenten basiert in Palladio, unter anderem, auf der Notation des RDSEFF. In Abbildung 2.4 ist eine Modellierung eines RDSEFF abgebildet. Dabei wurde der Aufruf `Download` einer `MediaManagement`-Komponente aus

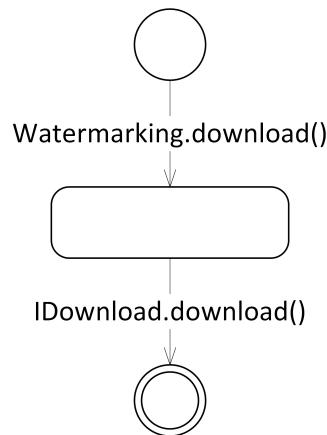


Abbildung 2.3.: Beispiel SEFF als endlicher Automat

dem Palladio Media-Store modelliert. Das Beispiel stammt aus [16]. Die graphische Darstellung ähnelt dabei der eines UML-Aktivitätsdiagramms, hat aber mit den UML-Elementen nichts gemeinsam. Der Kontrollfluss beginnt mit dem Ausführen der **InternalAction JNDI RMI Overhead**. Eine InternalAction repräsentiert dabei einen Aufruf zu Code, der sich innerhalb der Komponente befindet. Außerdem können damit Ressourcen angefordert werden. Im Beispiel werden CPU-Einheiten angefordert. Im Anschluss wird ein ExternalCall ausgeführt, der **Download** in einer anderen Komponente aufruft, um angeforderte Audio-Dateien zu erhalten. Ein ExternalCall ruft eine Funktion in einer anderen Komponente auf. Schließlich ruft ein weiterer ExternalCall Zip auf, wenn mehr als eine Datei angefordert wurde. Danach endet der Kontrollfluss.

Im Rahmen der Bachelorarbeit soll eine weitere SEFF-Spezialisierung entstehen, die den Datenfluss innerhalb der Komponenten beschreibt.

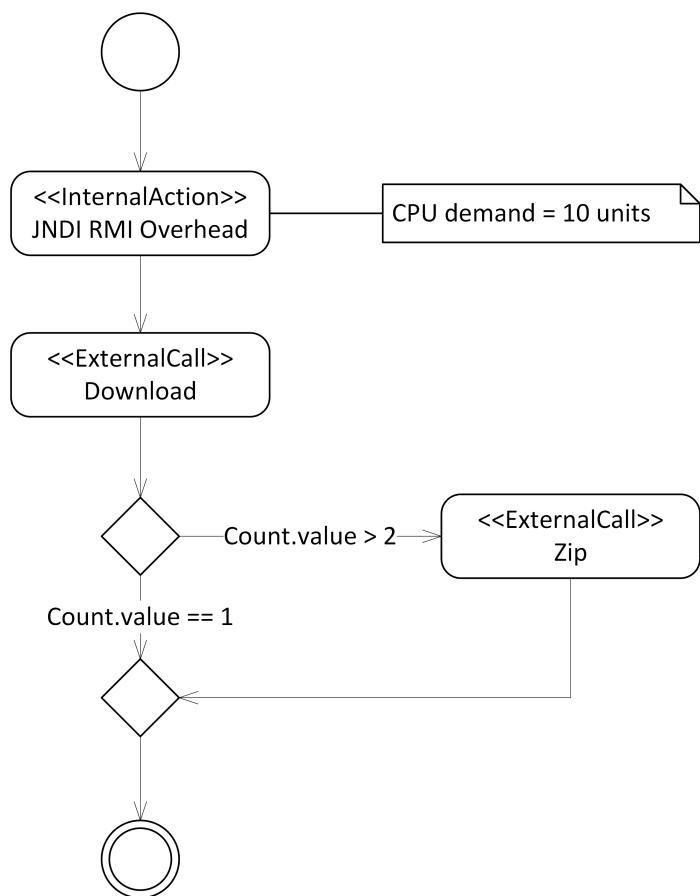


Abbildung 2.4.: RDSEFF, der den Download von Audio-Dateien aus dem Palladio Media-Store modelliert

3. Verwandte Arbeiten

Im Folgenden wird der aktuelle Forschungsstand bzgl. Datenflussanalysen auf Architekturebene beschrieben. Dabei werden Arbeiten vorgestellt, die als Grundlage dieser Bachelorarbeit dienen, bzw. von denen sich diese Arbeit abgrenzt.

3.1. Vertraulichkeitsbegriffsbildung

Zunächst muss ein Vertraulichkeitsbegriff gefunden werden, der sich auf Architekturebene überprüfen lässt. In der Literatur gibt es verschiedene Definitionen für Vertraulichkeit. In der Europäischen-Datenschutzgrundverordnung (EU-DSGVO) heißt es, dass Vertraulichkeit die Eigenschaft ist, dass Unbefugte keinen Zugang zu den Daten haben und weder die Daten noch die Geräte, mit denen diese verarbeitet werden, benutzen können (vgl. Art. 26ff EU-DSGVO). Eine andere Definition liefert das Bundesverfassungsgericht (BVerfG), in einem Urteil. Dort versteht man unter Vertraulichkeit den Schutz vorhandener Daten, gegen das Ausspähen (vgl. BVerfG 120, 274). In dem Buch **Secure Systems Development with UML** [6] findet sich eine weitere Definition. Sie besagt, dass Daten nur von legitimen Parteien gelesen werden dürfen. Eine weitere Definition findet sich in dem Buch **IT-Sicherheit: Konzepte-Verfahren-Protokolle** von Claudia Eckert [Eckert2013]. Dort heißt es, dass ein System Vertraulichkeit bewahrt, wenn es keine unautorisierte Informationsgewinnung ermöglicht. Diese Definition bezieht, im Vergleich zu den anderen, auch die Verarbeitung von Daten durch einen Angreifer ein und beschränkt sich nicht nur auf den Zugang zu den Daten.

Da in dieser Bachelorarbeit auch die Verarbeitung der Daten innerhalb von Komponenten betrachtet wird, wird im weiteren Verlauf der Arbeit der Vertraulichkeitsbegriff nach Eckert verwendet.

3.2. Datenflussmodellierung in Palladio

Für Palladio gibt es bereits Erweiterungen, die sich mit der Datenflussmodellierung befassen. Die Erweiterung **PASE** [14] erlaubt in PCM-Modellen den Datenfluss zu modellieren und zu analysieren. In **PASE** geht es um kryptographische Analysen. Diese basieren auf implementierungsnahe modellierten Daten, im Sinne von Variablen und Parametern. **Palladio.TX** [13] modelliert und analysiert transaktionale Informationssysteme und spezifiziert dazu Daten, sowie deren Speicherung in Datenbanktabellen. Schließlich stellt die Arbeit **Model-Driven Specification and Analysis of Confidentiality in Component-Based Systems** [9] eine weitere Palladio-Erweiterung bereit. Die Erweiterung spezifiziert, an der jeweiligen Schnittstelle, das gewünschte Verhalten der Komponente. Anschließend wird analysiert, ob diese Spezifikation zu dem gewünschten Ergebnis führt.

Während alle Arbeiten für ihren spezifischen Einsatzzweck wertvolle Ergebnisse liefern, kann keine der Erweiterungen feststellen, wie Daten innerhalb von Komponenten verarbeitet werden. Deshalb eignen sich die Erweiterungen für diese Bachelorarbeit nur bedingt, um auf ihnen aufzubauen. **PASE** hat das Problem, dass die Daten sehr fein modelliert werden müssen. **Palladio.TX** hat das Problem, dass die Datenmodellierung speziell für Datenbanken ausgelegt ist und deshalb für diese Arbeit nicht verwendet werden kann. Die Erweiterung aus der Arbeit **Model-Driven Specification and Analysis of Confidentiality in Component-Based Systems** bietet eine funktionierende Vertraulichkeitsanalyse für Palladio an. Deshalb soll mithilfe dieser Analyse die in Kapitel 4 entwickelte Modellierung von Daten und Datenflüssen, in 5 validiert werden.

3.3. Datenflussdiagramm

Um in dieser Arbeit Daten und Datenflüsse graphisch darstellen zu können, muss eine Notation dafür festgelegt werden. Zur Darstellung von Datenflüssen werden Datenflussdiagramme verwendet. Im Folgenden soll die Notation von Tom DeMarco [3] betrachtet werden. Bei dieser Notation gibt es vier Elemente. Das erste ist ein Datenspeicher. Dieser wird durch zwei parallele Linien dargestellt. Zwischen diesen Linien steht der Speichername des Datenspeichers. Das zweite Element stellt den Datenfluss dar. Der Datenfluss wird als gerichtete Kante dargestellt. Auf der Kante wird der Name notiert. Das dritte Element stellt einen Prozess oder Funktion dar. Diese werden als Kreis, mit einem Namen dargestellt. Schließlich gibt es noch eine Schnittstelle zu Außenwelt. Sie wird als Rechteck mit Namen dargestellt. Dabei unterscheidet man zwischen Schnittstellen aus denen Daten in das System fließen (Datenquelle) und durch die Daten aus dem System fließen (Datensenke). In Abbildung 3.1 ist ein Beispiel für ein mögliches Datenflussdiagramm abgebildet. Dabei fließen Daten, vom Namen **Input**, aus dem Datenspeicher **Database** in die Funktion **System**. Anschließend fließen Daten, mit dem Namen **Output** zur Schnittstelle **Customer**.

Die hier beschriebene Notation ist zu grobgranular für Datenflussanalysen, da Daten nur informell mit Namen spezifiziert werden und die Semantik der Verarbeitungsknoten nicht angegeben ist. Deshalb orientiert sich die Darstellung von Daten und Datenflüssen im Folgenden an einer Notation, die ähnlich zu der des RDSEFF aus Abbildung 2.4 ist.

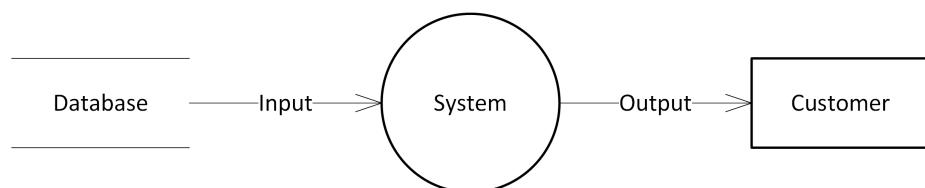


Abbildung 3.1.: Beispiel für ein Datenflussdiagramm

3.4. Informationsflussanalyse

Die Modelle, die im Verlauf der Bachelorarbeit entstehen, sollen später als Eingabe für Datenflussanalysen dienen. Deshalb sind Arbeiten relevant, die sich mit Datenflussanalysen beschäftigen. Daraus können Anforderungen abgeleitet werden, die für die Modellierung benötigt werden. Im Folgenden sollen solche Arbeiten betrachtet werden.

Eine dieser Grundlagenarbeiten ist das Buch **Principles of Program Analysis** [1]. In dieser Arbeit werden Datenflussanalysen auf Implementierungsebene beschrieben. Das heißt, dass um die Analyse ausführen zu können, eine Implementierung des Systems vorhanden sein muss. Die Datenflussanalyse wird mithilfe von Constant-Propagation, formuliert. Damit ist gemeint, dass konstante Ausdrücke zur Kompilierzeit ausgewertet werden. Ein Beispiel dafür ist in Abbildung 3.2 abgebildet. Dort werden die Kontrollkonstrukte statisch, zur Kompilierzeit, ausgewertet.

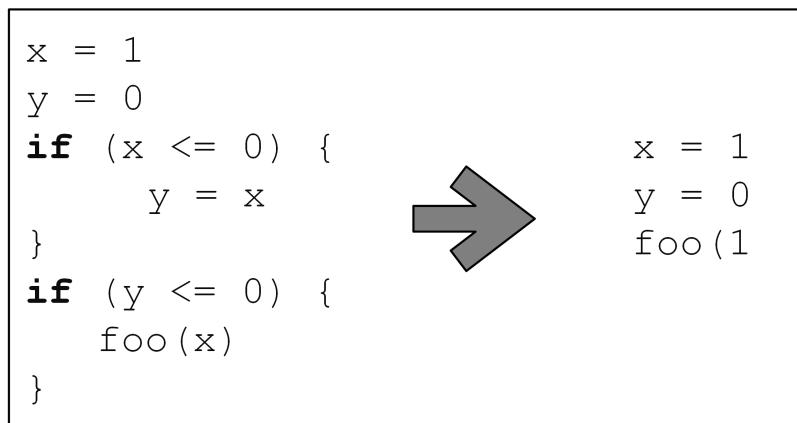


Abbildung 3.2.: Beispiel für Constant-Propagation

Der Ausgangspunkt für die Analyse ist dabei der Kontrollflussgraph (CFG). Ein CFG besteht aus:

- Einer Menge von Knoten, die Instruktionen darstellen.
- Einem Wurzelknoten, mit dem die Ausführung beginnt.
- Einer Menge von gerichteter Kanten, die mögliche Programmabläufe darstellen.

In Abbildung 3.3 ist der CFG für das Beispiel aus Abbildung 3.2 abgebildet. Dabei werden in die Knoten eine oder mehrere Anweisungen des Codes eingetragen. Mit diesem CFG kann im Anschluss eine Datenflussanalyse durchgeführt werden.

Diese Analyse arbeitet jedoch nicht auf Datenflüssen, sondern leitet diese aus dem Kontrollfluss ab. Außerdem setzt die Analyse eine fertige Implementierung voraus. In dieser Bachelorarbeit sollen Datenflüsse auf Architekturebene betrachtet werden. Dabei soll die Analyse direkt auf diesen Datenflüssen arbeiten und diese nicht aus dem Kontrollfluss ableiten. Deshalb soll im Weiteren nicht näher auf diese Arbeit eingegangen werden.

3. Verwandte Arbeiten

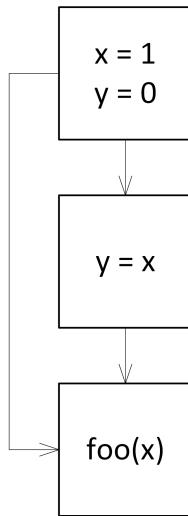


Abbildung 3.3.: Beispiel eines CFG, für das Beispiel aus Abbildung 3.2

In der Arbeit von Jürjens et al. [6] wurde **UMLsec** vorgestellt. Dabei handelt es sich um eine Erweiterung für UML. **UMLsec** definiert 21 Stereotypen, die UML um Sicherheits-eigenschaften erweitern. Ein Beispiel für einen Stereotypen, ist **Encrypted**. Mit diesem Stereotypen kann eine verschlüsselte Verbindung modelliert werden. Eine Sicherheits-analyse kann im Anschluss mithilfe eines definierten Angreifers prüfen, ob das System sicher ist. Auch hier werden Datenflüsse nicht direkt verwendet, sondern aus dem Kon-trollfluss abgeleitet. Da die Erweiterung für UML konzipiert ist und im Vordergrund der Arbeit die Sicherheit eines Systems und nicht der Datenfluss steht, wird sie im weiteren Verlauf der Bachelorarbeit nicht weiter betrachtet.

Eine Arbeit, die sich mit Datenflüsse auf Architekturebene befasst, ist die bereits in Abschnitt 3.2 vorgestellte Arbeit **Model-Driven Specification and Analysis of Confidentiality in Component-Based Systems** [9]. Sie erweitert das PCM um Datenflüsse und eine Vertraulichkeitsanalyse. Die Arbeit basiert auf dem Poster **Specification and Verification of Confidentiality in Component-Based Systems** [10]. Datenflüsse wer-den aber nicht innerhalb der Komponenten betrachtet. Außerdem werden Datenflüsse aus dem Kontrollfluss abgeleitet. Stattdessen werden Anforderungen für das Verhalten der Datenflüsse spezifiziert und anschließend überprüft. Die Prüfung der Komponente erfolgt zur Implementierungszeit auf Basis des Quelltextes. Die Arbeit beschreibt außer-dem eine Vertraulichkeitsanalyse für Palladio. Diese Analyse prüft die Vertraulichkeit in komponentenbasierten Systemen. Für die Analyse müssen Eingabe und Ausgabe eines Systems spezifiziert werden, sowie Zugriffsspezifikation für Hardware und Kommunikationsverbindungen. Mithilfe dieser Informationen und einem Angreifer Modell, wird eine Architektur- und Code-Analyse durchgeführt. Nach Durchführung der Analyse werden Designfehler, Datenlecks und Verstöße gegen die Spezifikation angezeigt. Die Vertrau-lichkeitsanalyse hält sich ebenfalls an die Definition von Vertraulichkeit nach Eckert [4].

In allen vorgestellten Arbeiten werden Eigenschaften für Daten festgemacht. Jedoch werden die Datenflüsse aus Kontrollflüssen abgeleitet. In dieser Bachelorarbeit sollen Da-ten und Datenflüsse als Objekte erster Klasse eingeführt werden. Für die Validierung soll

jedoch die Vertraulichkeitsanalyse aus [11] verwendet werden, da diese bereits eine funktionierende Vertraulichkeitsanalyse für Palladio anbietet.

3.5. Erweiterung des PCM

Die Erweiterung des PCMs erfolgt durch die Erweiterung seines Meta-Modells. Dazu wurde von Strittmatter et al. [21] ein Ansatz entwickelt, bei dem die zu modellierenden Informationen in Gruppen eingeteilt werden. Das Meta-Modell wird in Schichten eingeteilt, die jeweils eine Gruppe abbilden. Schichten referenzieren sich gegenseitig. Dies führt zu einer modularen, flexiblen und erweiterbaren Struktur, die die Koexistenz von verschiedenen Qualitätsattributen und -analysen ermöglicht. Das Meta-Modell ist in die folgenden vier Schichten unterteilt:

- Die **Paradigm**-Schicht ist die Basisschicht. Sie legt den Grundstein für die Modellierungssprache, indem sie die Struktur, aber keine Semantik bereitstellt.
- Die Semantik für die abstrakte Struktur der Sprache liefert die **Domain**-Schicht.
- In der **Quality**-Schicht werden Qualitätseigenschaften für bestimmte Bereiche hinzugefügt.
- Schließlich bietet die **Analysis**-Schicht Module für die Eingabe, Ausgabe und den internen Zustand. Außerdem gibt es Konfigurationsoptionen für Analysen.

Diese Arbeit wird für die Modellierung in Kapitel 4 verwendet.

4. Modellierung

Während der Bachelorarbeit wurde ein Meta-Modell zur Darstellung von Daten und Datenflüssen, in Palladio, erstellt. Dabei wurde nach dem Prozess von Völter und Stahl vorgegangen [18]. Dieser beschreibt das Vorgehen für die Entwicklung eines Meta-Modells. Der Prozess besteht aus mehreren Schritten. Zunächst findet eine Domänenanalyse statt. Sie besteht aus einer Anforderungserhebung (Abschnitt 4.1) und anschließender Meta-Modell-Bildung (ab Abschnitt 4.2). Nach der Domänenanalyse folgt die Erstellung eines Referenzmodells (Abschnitt 5.1). Dabei handelt es sich um eine Instanz des Meta-Modells, aus dem ersten Schritt. Als nächstes wird das Programmiermodell dokumentiert. Das Programmiermodell erklärt, wie eine Anwendung implementiert werden kann, die auf dem Meta-Model basiert. In der Bachelorarbeit fällt das Programmiermodell weg, da kein Quelltext erzeugt wird und auch sonst das Ergebnis der Modellierung nicht mehr weiter, in einer anderen Form, bearbeitet werden muss. Im nächsten Schritt wird eine Transformation (Abschnitt 5.2) durchgeführt. Dieser Schritt dient dazu, ein gegebenes Anwendungsmodell automatisch in eine Implementierung oder einen Rahmen zu überführen. Schließlich soll zu dem Meta-Modell noch ein Editor erstellt werden. Ein Editor stand nicht im Fokus der Bachelorarbeit und wurde deshalb nicht realisiert. Allgemein kann das Modell im automatisch erzeugten Editor der Eclipse IDE editiert werden.

Auf die Anforderungserhebung, sowie das Meta-Modell wird in den folgenden Abschnitten genauer eingegangen. Das Referenzmodell und die Transformation werden in der Validierung in Kapitel 5 erläutert.

4.1. Anforderungserhebung

Bei der Anforderungserhebung wurde das Meta-Modell aus der Arbeit **Model-Driven Specification and Analysis of Confidentiality in Component-Based Systems** [9] untersucht. Dabei wurde nach Elementen gesucht, die zum Ableiten des vertraulichkeitsrelevanten Verhaltens einer Komponente benötigt werden, jedoch noch nicht vorhanden sind und welche man generisch in Datenflüssen ausdrücken kann. Dazu wurde die Vertraulichkeitsanalyse aus Abschnitt 3.4 untersucht. Für die Analyse müssen Eingabe und Ausgabe des zu untersuchenden Systems spezifiziert werden. Außerdem können Zugriffsspezifikationen für Hardware und Kommunikationsverbindungen spezifiziert werden. Um dies spezifizieren zu können, wurde ein Annotationsmodell eingeführt. Bei einem Annotationsmodell handelt es sich um ein strukturiertes Modell von Annotationen. Die Annotationen können an Elemente im Komponenten-Repository- und Ressourcen-Umgebungs-Modell angehängt werden, um diese zu erweitern und zusätzliche Informationen zu liefern. Das Annotationsmodell wurde für die Bachelorarbeit nachmodelliert. Dabei wurde darauf geachtet, dass das Modell in Zukunft erweiterbar ist, z.B. zur Nutzung

4. Modellierung

für neue oder andere Qualitätsanalysen. Mithilfe von Oberklassen wurde eine Schnittstelle geschaffen, die das ermöglicht. Außerdem sollte eine Gruppierung der Elemente möglich sein, z.B. je untersuchtem Qualitätsattribut. Dieses Verhalten wurde mithilfe von Containern sichergestellt. Das Modell wird in Abschnitt 4.4 beschrieben.

In der Arbeit [9] wird mithilfe von **DataSets** die Eingabe und Ausgabe des Systems spezifiziert. Ein **DataSet** ist dabei eine Gruppierung von Daten, die in dem System verarbeitet werden. Eine Datenflussanalyse innerhalb von Komponenten ist nicht vorgesehen. Für die Komponente werden gewünschte Ergebnisse für die Eingabe und Ausgabe spezifiziert. Im Anschluss untersucht die Analyse, wie sich einzelne Parameter gegenseitig beeinflussen. Mithilfe der Modellierung dieser Bachelorarbeit soll aber auch der Datenfluss innerhalb von Komponenten nachvollzogen werden. Es sollen außerdem nicht direkt Parameter, sondern Daten betrachtet werden. Das heißt, dass es möglich sein muss angeben zu können welche Daten sich innerhalb von Komponenten beeinflussen. Außerdem muss es möglich sein Daten Parametern zuordnen zu können. Dazu müssen zunächst Daten modelliert werden. Dabei handelt es sich nicht um individuelle Daten sondern um Datenklassen. Die Modellierung ist in Abschnitt 4.5 beschrieben. Der Datenfluss der Daten innerhalb von Komponenten, soll mithilfe eines Datenfluss-Service-Effect-Specification (DFSEFF) modelliert werden. Bei diesem DFSEFF soll nur die Eingabe spezifiziert werden. Das heißt, dass nur die **DataSets** der Eingabe spezifiziert werden. Die Ausgabe wird aus der Modellierung des DFSEFF abgeleitet. Aktionen die ein DFSEFF unterstützt sind Aufrufe an externe Dienste, das Kombinieren von mehreren Datenklassen zu einer neuen und das Erstellen einer neuen Datenklasse. Zu diesen Aktionen gehört auch die Fähigkeit Daten und Parameter zu verknüpfen. In Abschnitt 4.6 wird der DFSEFF beschrieben. Außerdem soll die Eingabe eines Benutzers des Systems, mithilfe von Datenflüssen unterschbar sein. Auch hier soll nur die Eingabe spezifiziert werden. Wohin die Daten fließen, wird auch hier aus der Modellierung abgeleitet. Eine Beschreibung dieses Modells befindet sich in Abschnitt 4.7.

Die Meta-Modell-Bildung orientiert sich an der Arbeit von Strittmatter et. al. [21]. Dabei wurde die **Quality**-Schicht erweitert, da dort Qualitätseigenschaften spezifiziert werden können. Außerdem wurde die **Domain**-Schicht um weitere Attribute und Qualitäts-eigenschaften erweitert. Die **Analysis**-Schicht wurde für die Validierung prototypisch erweitert. Allerdings ist die Analyse kein Kernbeitrag dieser Bachelorarbeit. Die Arbeit zur Meta-Modell-Erweiterung wurde in Kapitel 3 beschrieben.

Als nächstes wurde untersucht, welche bestehenden Palladio-Elemente wiederverwenden werden können und welche neu modelliert werden müssen. Dabei kann die Wiederverwendung auf Meta-Modell- und Modell-Ebene statt finden. Bei ersterem, wird bei der Erstellung des Datenfluss-Meta-Modells auf bestehenden Meta-Modell-Elementen des PCM aufgebaut. Beim zweiten, kann der Architekt seine bestehenden PCM-Modelle um Datenflüsse erweitern. Die Vorteile bei der Wiederverwendung von PCM-Meta-Modell-Elementen sind, dass der Meta-Modellierer weniger Arbeit hat, da er auf bestehenden Elementen aufbaut und diese nicht nochmal modellieren muss. Der Benutzer hat den Vorteil, dass auch er bereits erstellte Elemente nicht nochmal modellieren muss und seine Modelle nur um Datenflüsse erweitern kann. Außerdem muss er keine neuen Elemente lernen. Nachteile sind jedoch, dass die Semantik der existierenden Elemente nicht zur angepeilten Semantik der neuen Elemente passen muss. Das äußert sich z.B. dadurch, dass Referen-

zen zu Elementen existieren, die für den Anwendungsfall irrelevant sind. Eine mögliche Referenz ist der RDSEFF. Dies kann zu einem Problem führen, da in dieser Bachelorarbeit, unter anderem ein SEFF modelliert werden soll, mit dem Datenflüsse modellierbar sind. Bei der Wiederverwendung der Elemente, die den RDSEFF referenzieren, wird der DFSEFF abhängig vom RDSEFF. Für den Benutzer entsteht dadurch auch eine unklare Semantik, da dieser nicht weiß, welche Eigenschaften er angeben muss, um sinnvolle Ergebnisse zu erhalten. Außerdem gibt es Elemente die zwar unabhängig vom RDSEFF sind, aber Attribute enthalten, die für den Anwendungsfall unbrauchbar sind. Auch hier entsteht für den Benutzer eine unklare Semantik. Außerdem können Änderungen keine Auswirkung haben, da die angegebenen Eigenschaften nicht verwendet werden. Eine Wiederverwendung wird deshalb nur durchgeführt, wenn die Semantik übernommen oder sinnvoll erweitert werden kann, z.B. wenn alle Referenzen in beiden Kontexten sinnvoll sind.

Die Ergebnisse der Meta-Modell-Bildung werden in den folgenden Abschnitten präsentiert.

4.2. Übersicht der Pakete

Das Meta-Modell besteht aus den in Abbildung 4.1 gezeigten Paketen. Die Pakete orientieren sich an den in Palladio verwendeten Sichten aus Abschnitt 2.3.

Das Paket Container-Stereotypes erweitert das Ressourcen-Umgebungs-Modell. Für die Erweiterung ist der Softwareverteilungsexperte zuständig. Es enthält Eigenschaften, die an einen Resource-Container oder eine Linking-Resource in Palladio angehängt werden können. Dabei handelt es sich bei einer Linking-Resource um einen Verbindungs-kanal. Bei einem Resource-Container handelt es sich um eine Hardware-Ressource. Die Elemente sind in Abschnitt 4.4 genauer beschrieben.

Das Paket DFSEFF erweitert das Komponenten-Repository-Modell. Es ist für die Modellierung eines SEFFs zuständig, mit dem es möglich ist, den Datenfluss innerhalb einer Komponente zu modellieren. Die Informationen, die für die Modellierung benötigt werden, werden vom Komponentenentwickler bereitgestellt. In Abschnitt 4.6 wird die Erweiterung genauer beschrieben.

Das Paket Data erweitert (momentan) das Repository-Modell und das Nutzungsmodell. Die Informationen werden dabei vom Komponentenentwickler und dem Domänenexperten bereitgestellt. In dem Paket werden Daten modelliert, die in einem System verwendet werden. Dabei handelt es sich nicht um individuelle Daten, sondern um Datenklassen. Eine genaue Beschreibung befindet sich in Abschnitt 4.5.

Schließlich erweitert das Paket Usage das Nutzungsmodell. Alle Elemente die für die Modellierung des Datenflusses für das Nutzungsmodell in Palladio benötigt werden, befinden sich darin. Der Domänenexperte ist für die Modellierung und Erweiterung zuständig. Die Erweiterung wird in Abschnitt 4.7 genauer beschrieben.

4. Modellierung

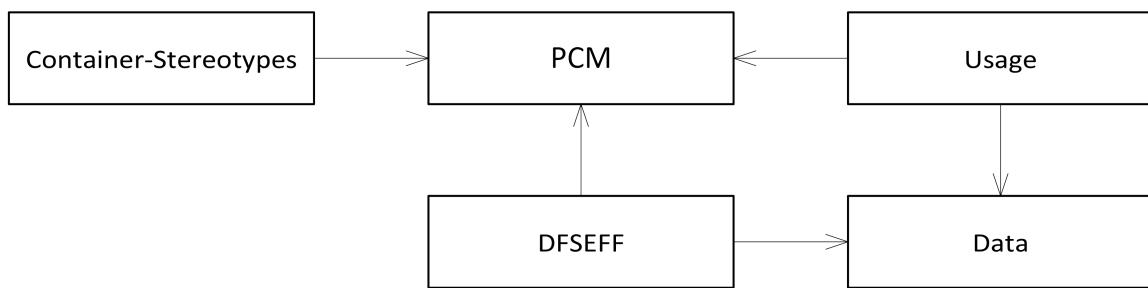


Abbildung 4.1.: Übersicht der Pakete der Meta-Modell-Erweiterung

4.3. Beispieldaten

Um die im Folgenden beschriebenen Modellierung besser zu veranschaulichen, soll an dieser Stelle ein Szenario beschrieben werden, in das die einzelnen Pakete und Elemente eingeordnet werden können. In Abbildung 4.2 ist das Szenario abgebildet. In diesem Szenario gibt es einen Laden, der ein Warenlager hat. In diesem Warenlager befinden sich Produkte. Der Laden besteht nur aus dem Besitzer und nur dieser hat Zugang zu dem Warenlager. Wenn ein Kunde in den Laden kommt richtet er seine Kaufbegehrungen an den Ladenbesitzer und dieser verkauft ihm je nach Verfügbarkeit das gewünschte Produkt. Für diesen Kaufvorgang benötigt der Ladenbesitzer die Kreditkarte des Kunden. Der Geldbetrag wird mithilfe eines Kreditkartenlesegeräts bei der Bank des Kunden eingefordert. Das Kreditkartenlesegerät ist über das Internet mit der Bank verbunden.

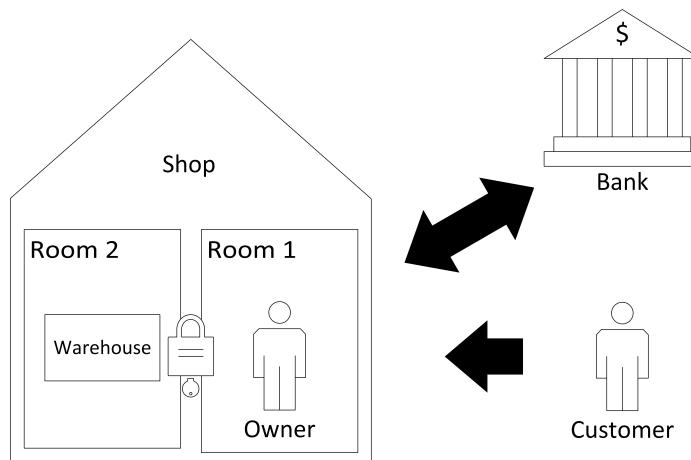


Abbildung 4.2.: Das Laden-Beispiel-Szenario

Der Kaufvorgang ist in Abbildung 4.3 abgebildet. Der Kunde signalisiert dem Besitzer des Ladens, welches Produkt er haben möchte. Der Besitzer prüft nun im Warenlager, ob dieses Produkt vorhanden ist. Ist das der Fall, holt er es heraus und gibt es dem Kunden. Daraufhin bezahlt dieser mit seiner Kreditkarte. Dies ist nur der Basis-Kaufvorgang. Es gibt noch weitere Abläufe, die hier nicht beschrieben werden, wie z.B., wenn das Produkt nicht vorhanden ist.

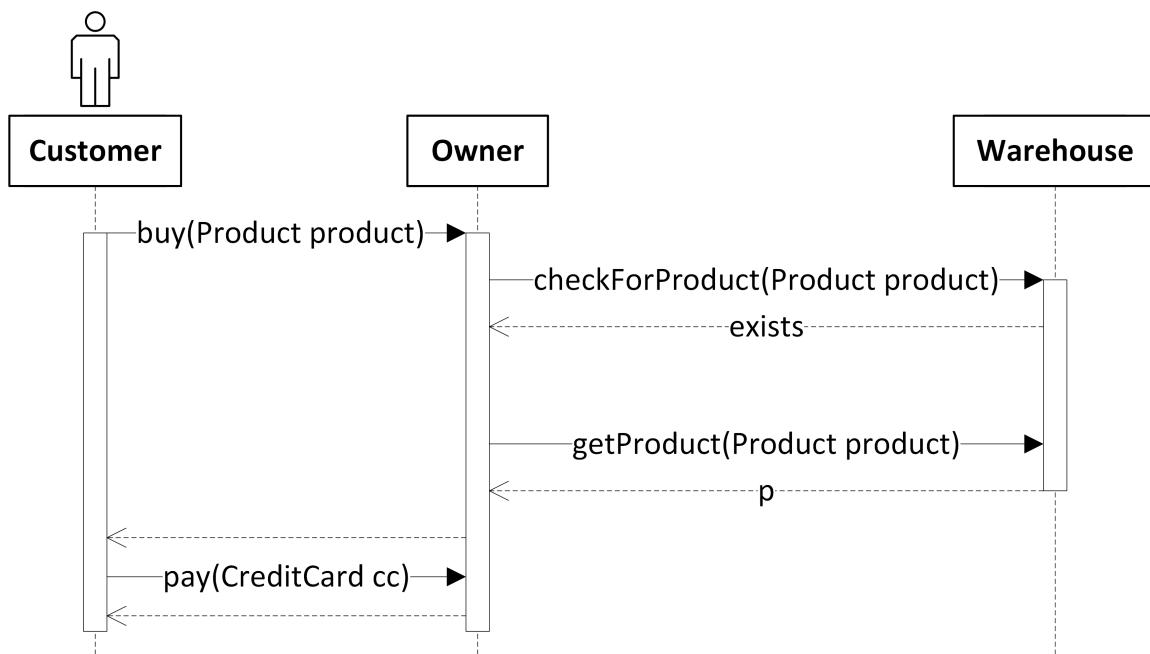


Abbildung 4.3.: Der Ablauf eines Basis-Kaufvorgangs im Laden

4.4. Software-Verteilungsexperte

Im folgenden wird die Erweiterung des Ressourcen-Umgebungs-Modells vorgestellt. Mit Hilfe dieser Erweiterung lassen sich die Linking-Resource oder der Resource-Container um verschiedene Eigenschaften erweitern. Die hier beschriebenen Eigenschaften stammen aus [9], um Vertraulichkeit auf Hardware-Ebene zu unterstützen. Die Eigenschaften werden vom Softwareverteilungsexperten erstellt und referenzieren Resource-Container und Linking-Resources im Execution-Environment-Modell. Eine Darstellung des Modells ist in Abbildung 4.4 abgebildet.

Damit in Zukunft weitere Eigenschaften hinzugefügt werden können, haben diese eine gemeinsame Oberklasse. Dadurch ist Erweiterbarkeit möglich. Dabei haben die Eigenschaften des Resource-Containers und der Linking-Resources jeweils eine eigene Oberklasse. Außerdem erweitern die Eigenschaften nicht direkt den Resource-Container bzw. die Linking-Resource, sondern werden in einem Container-Element gesammelt und dieser verweist auf den jeweiligen Resource-Container, bzw. die jeweilige Linking-Resource. Dies hat außerdem den Vorteil, dass mehrere Container auf einen Resource-Container oder eine Linking-Resource verweisen können und somit Eigenschaften gruppiert werden können. Nicht benötigte Gruppen können dann bei Bedarf ein- oder ausgeblendet werden.

Die Eigenschaften, die einen Resource-Container erweitern können sind in Abbildung 4.5 abgebildet. Zu diesen Eigenschaften gehört die Runtime-Eigenschaft. Diese Eigenschaft sagt aus, ob auf dem Resource-Container noch weitere Software genutzt wird oder ob alles modelliert wurde. Um das zu spezifizieren, gibt es zwei Optionen. Diese werden in der Aufzählung Runtime modelliert. Wenn weitere Software auf dem Resource-Container genutzt wird, kann das mit *shared* modelliert werden. Wenn aber keine weitere Softwa-

4. Modellierung

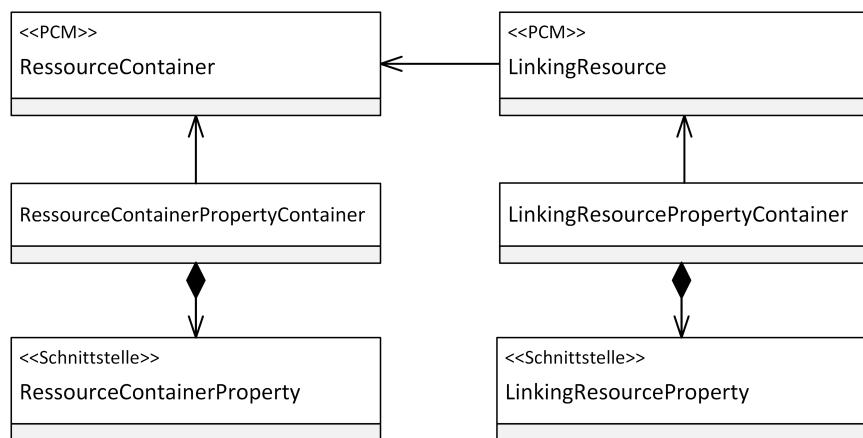


Abbildung 4.4.: Übersicht der Erweiterung für das Ressourcen-Umgebungs-Modell

re auf dem Resource-Container genutzt wird, kann dies mit *exclusive* modelliert werden. Möchte man angeben wo sich ein Resource-Container befindet, erweitert man diesen mit der Location-Eigenschaft. Möchte man den Resource-Container mit einem Sicherheitsmechanismus schützen, erweitert man diesen mit der Eigenschaft TamperProtection. Dadurch lässt sich ein Sicherheitsmechanismus spezifizieren, der den Resource-Container vor Angriffen schützen soll.

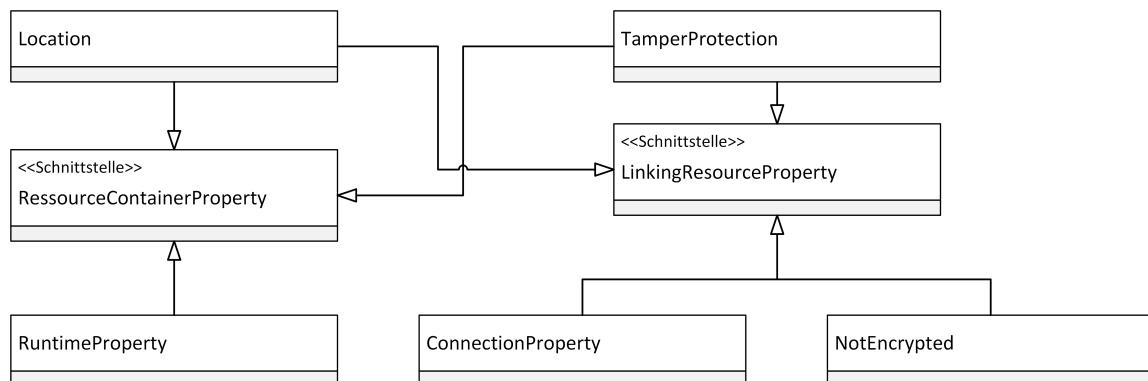


Abbildung 4.5.: Übersicht der Eigenschaften für Resource-Container und Linking-Resource

In Abbildung 4.6 ist eine mögliche Modellierung eines Resource-Containers des Beispiels aus Abschnitt 4.3 dargestellt. Dabei ist das Kreditkartenlesegerät (*CCReader*) als Resource-Container modelliert. Auf den Resource-Container verweist ein *ResourceContainerPropertyContainer*. Dieser sammelt alle Eigenschaften, die an das Kreditkartenlesegerät angehängt werden sollen. Dazu gehört die *Runtime*-Eigenschaft. Sie ist hier *exclusive*, da keine weitere Software auf dem Kreditkartenlesegerät genutzt wird. Eine weitere Eigenschaft, die angehängt wird, ist *TamperProtection*. Das Kreditkartenlesegerät wird mit einem Passwort vor unbefugten Zugang geschützt. Schließlich wird mit der *Location*-Eigenschaft festgelegt, dass das Kreditkartenlesegerät sich im zweiten Raum des Ladens befindet.

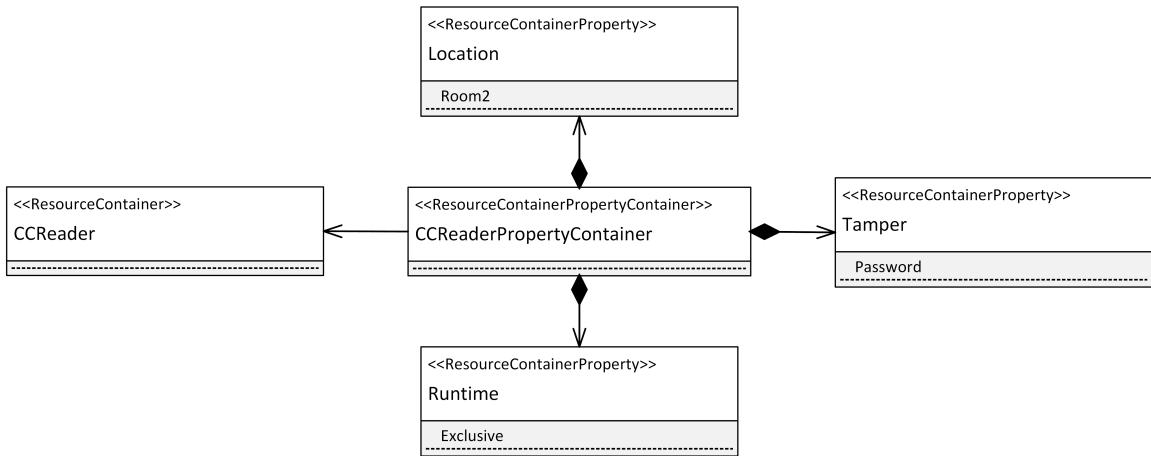


Abbildung 4.6.: Resource-Container Beispiel des Laden-Szenarios

Auch die Linking-Resource kann mit Eigenschaften erweitert werden. Diese sind auch in Abbildung 4.5 abgebildet. Eine Eigenschaft, die Linking-Resource erweitern kann, ist `Connection`. Diese Eigenschaft gibt an, ob es weitere Verbindungen auf der Linking-Resource gibt, die aber nicht modelliert wurden. Die Optionen die dabei angegeben werden können sind in der Aufzählung `Connection` modelliert. Mit `complete` wird signalisiert, dass es keine weiteren Verbindungen gibt. Wenn es weitere Verbindungen gibt, die nicht modelliert wurden, kann dies mit `existing` modelliert werden. Schließlich kann ein Resource-Container Ports enthalten, die nicht verwendet werden, aber eine Verbindung an diesem möglich wäre. Dies kann mit `possible` modelliert werden. Die Eigenschaft `NotEncrypted` gibt an, ob etwas auf der Linking-Resource nicht verschlüsselt wird, z.B. Protokolle oder Datengröße. Die beiden Eigenschaften `Location` und `TamperProtection` sind analog zum Resource-Container.

In Abbildung 4.7 ist eine mögliche Modellierung einer Linking-Resource des Beispiels aus Abschnitt 4.3 abgebildet. Hier ist die Internetverbindung zwischen dem Kreditkartenlesegerät des Ladens und der Bank des Kundens als Linking-Resource modelliert. Auf die Linking-Resource verweist ein `LinkingResourcePropertyContainer`. Dieser sammelt alle Eigenschaften, die an die Linking-Resource angehängt werden sollen. Dazu gehört die `Location`-Eigenschaft. Sie spezifiziert als Ort, für die Linking-Resource, das Internet. Geschützt ist die Linking-Resource mit einem Passwort. Dies ist in der Eigenschaft `TamperProtection` spezifiziert. Außerdem wird spezifiziert, dass das Protokoll, über das kommuniziert wird, nicht verschlüsselt wird. Mithilfe der Eigenschaft `NotEncrypted` wird dies modelliert. Schließlich wird mit der `Connection`-Eigenschaft angegeben, dass es keine weiteren Verbindungen, außer den modellierten gibt (`complete`).

4.5. Datenmodellierer

In diesem Abschnitt, wird die Modellierung von Daten beschrieben, die in einem System verarbeitet werden. Dabei handelt es sich nicht um individuelle Daten, sondern um Datenklassen. Das können z.B. Daten sein, die für einen Kauf benötigt werden, wie Na-

4. Modellierung

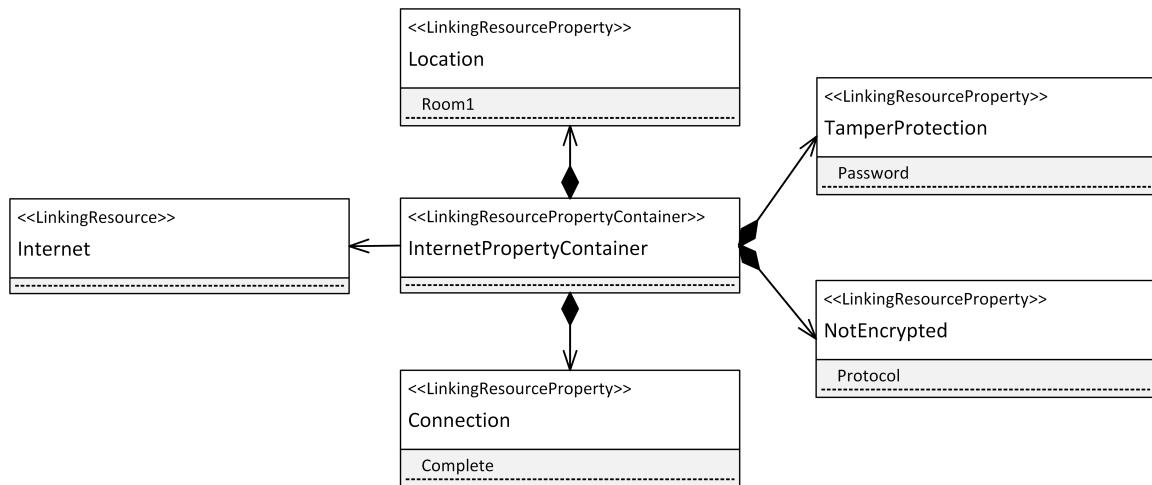


Abbildung 4.7.: Linking-Resource Beispiel des Laden-Szenarios

me oder Adresse. Das bedeutet, dass der Benutzer die Datenklasse Namen in den Dienst eingibt und nicht einen speziellen Namen. Die Daten werden vom Komponentenentwickler sowie dem Domänenexperten im Ressourcen-Umgebungs-Modell und Nutzungsmodell eingesetzt. In Abbildung 4.8 ist diese Modellierung abgebildet.

Die Klasse Data repräsentiert dabei eine Datenklasse. An Data können Eigenschaften (DataProperty) angehängt werden. Mithilfe eines DataPropertyContainers werden die Eigenschaften gesammelt und an eine Datenklasse gehängt. Die Eigenschaften können je nach untersuchtem Qualitätsattribut unterschiedlich sein. Sie können daher variabel gewählt werden. Für Datenflussanalysen können so Sicherheitseigenschaften spezifiziert werden. Darauf aufbauend ist eine Überprüfung der Einhaltung dieser Eigenschaften möglich. Außerdem befindet sich in dem Paket das Element DataSet. Ein DataSet ist eine Gruppierung von Datenklassen. Daten können so verschiedene Gruppierungen zugeordnet werden. Im Anschluss kann, in Hinblick auf eine Datenflussanalyse, festgelegt werden, welche Person Zugriff auf welche DataSets haben soll. Während der Datenflussanalyse kann schließlich geprüft werden, ob jeder nur Zugriff auf die DataSets hat, auf die er zugreifen darf.

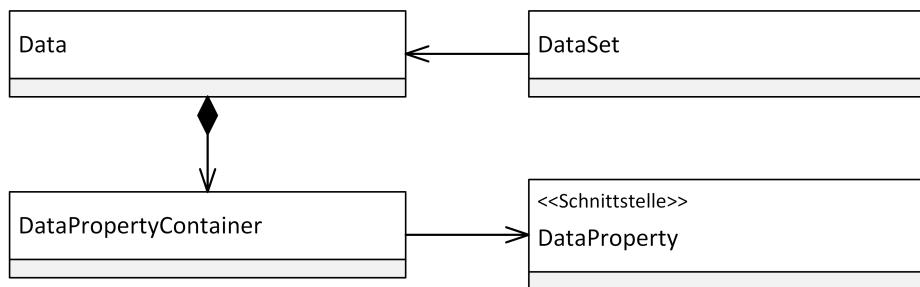


Abbildung 4.8.: Übersicht der Datenmodellierung

In dem Beispielszenario aus Abschnitt 4.3 wären *Kreditkartendaten* eine Datenklasse und *Kunde*, *Besitzer* und *Warenlager* als DataSet modelliert. Dabei sind die *Kreditkar-*

tendaten nur im `DataSet Kunde`. Das heißt, dass nur Personen, die dem `DataSet Kunde` zugewiesen sind, Zugriff auf die Daten haben. Bei einer Datenflussanalyse kann nun geprüft werden, ob die *Kreditkartendaten* bei einer Person oder einem System ankommen, die eigentlich keinen Zugriff auf diese haben sollte.

4.6. Komponentenentwickler

In diesem Abschnitt wird die Modellierung für einen DFSEFF beschrieben. Mit diesem DFSEFF kann der Komponentenentwickler einen Datenfluss innerhalb einer Komponente, in Form eines SEFF, modellieren. Dabei soll es möglich sein Daten zu verfeinern, indem man diese mit Parametern verknüpft. Außerdem soll der Datenfluss über mehrere Datenverarbeitungsschritte modelliert werden können und schließlich ein globaler Datenfluss aus dem Nutzungs-, Komponenten-Repository-, Ressourcen-Umgebungs-, Komponenten-Allokations- und System-Modell abgeleitet werden. Vor der Meta-Modell-Bildung muss überlegt werden, ob Elemente aus dem PCM wiederverwendet werden können. Das war in den Abschnitten davor nicht nötig, da es nichts Vergleichbares in Palladio gab. In Palladio existiert aber bereits ein SEFF. Der RDSEFF, der eine SEFF-Spezialisierung für Performanzvorhersagen darstellt.

Als ein Element, das wiederverwendet werden kann, wurde die `Signatur` identifiziert. Eine `Signatur` modelliert die Signatur einer Methode. Dazu zählen Parameter, der Rückgabetyp und Fehlerspezifikationen. Der Vorteil dieser Wiederverwendung ist, dass der Komponentenentwickler seine bereits modellierten Signaturen wiederverwenden kann. Außerdem geht der Zusammenhang zwischen Kontroll- und Datenfluss nicht verloren. Ein weiteres Element, das wiederverwendet werden kann, ist `Parameter`. `Parameter` modelliert einen Parameter einer `Signatur`. `Parameter` besteht aus einem Namen und einem Datentyp. Die Vorteile sind ähnlich zu denen der `Signatur`. `Parameter` ist nicht vom RDSEFF abhängig. Außerdem geht der Zusammenhang zwischen Kontroll- und Datenfluss nicht verloren. Alternativ wäre hier ein neues `Parameter` Element möglich, dass direkt Daten und Parameter verknüpft. Jedoch hat bei dieser Modellierung der Benutzer wenig Mehrwert und muss stattdessen evtl. bereits modellierte Parameter neu modellieren.

Elemente, die hingegen nicht wiederverwendet werden können, sind `AbstractAction`, ihre Unterklassen und `ResourceDemandingBehaviour`. `AbstractAction` modelliert dabei einen Aufruf zu einem internen oder externen Dienst. Außerdem hat `AbstractAction` eine Referenz zu `ResourceDemandingBehaviour`. Dieses Element modelliert das Verhalten einer Komponente als Sequenz von `AbstractActions`. Durch die Wiederverwendung der Elemente würden auch eine Vielzahl anderer Elemente im DFSEFF verwendbar werden, die aber an dieser Stelle keinen Sinn machen. Ein Beispiel wäre hier die `ReleaseAction`, die eine Unterklasse von `AbstractAction` ist. Die `ReleaseAction` gibt Ressourcen im RDSEFF wieder frei. Da Ressourcen für die untersuchte Analyse nicht benötigt werden, ist eine Wiederverwendung an dieser Stelle nicht sinnvoll.

Eine Übersicht des Meta-Modells ist in Abbildung 4.9 abgebildet. Das Root-Element, dieser Erweiterung, ist das Element `DataFlowSEFF`. Aufgrund einer Containment-Beziehung zwischen den Elementen `BasicComponent` und `ServiceEffectSpecification` kann

4. Modellierung

das Element `ServiceEffectSpecification` nicht wiederverwendet werden. `ServiceEffectSpecification` ist eine abstrakte Klasse und modelliert dabei einen SEFF. Spezielle SEFFs sollten von dieser Klasse erben. Stattdessen referenziert das Element `DataFlowSEFF` die Elemente `BasicComponent` und `Signature`. Eine `BasicComponent` modelliert in PCM eine Komponente. Komponenten können dabei abstrakt oder konkret sein. Abstrakte Komponenten spezifizieren nur ihre Schnittstellen und werden vom Software-Architekten als Platzhalter verwendet. Sie werden im weiteren Verlauf der Bachelorarbeit nicht näher betrachtet. Konkrete Komponenten sind entweder Basis Komponenten (`BasicComponent`) oder setzen sich aus mehreren zusammen. Eine `BasicComponent` spezifiziert dabei, für jede Operation die sie bereitstellt, ein Verhalten. Die zusammengesetzten Komponenten werden im Weiteren nicht näher betrachtet.

Außerdem enthält `DataFlowSEFF` das Element `DataFlowBehavior`. `DataFlowBehavior` modelliert das Verhalten einer Komponente als Sequenz, die aus mehreren `DataFlowAbstractActions` besteht. Der Ablauf eines DFSEFF ist ähnlich eines UML-Aktivitätsdia-

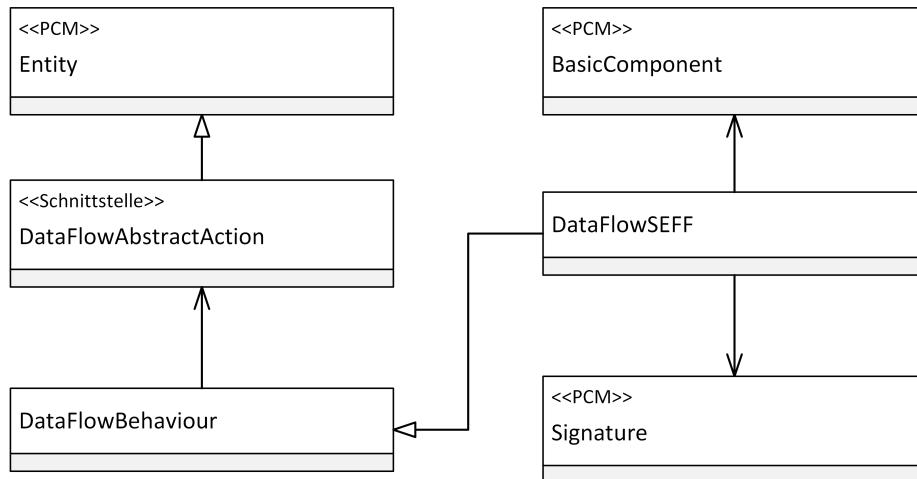


Abbildung 4.9.: Übersicht der DFSEFF Erweiterung

gramm und des kontrollflussorientierten SEFFs. Die verschiedenen Elemente mit denen der Datenfluss beschrieben wird, sind in Abbildung 4.10 abgebildet. Ein DFSEFF besteht aus einer `StartAction` und einer `StopAction`, die den Anfang und das Ende des Datenflusses kennzeichnen. Zwischen diesen beiden Elementen können ein Reihe anderer `DataFlowAbstractActions` eingegliedert werden. Dazu gehört die `DataFlowExternalAction` und die `InternalDataSourceAction`. Beide Elemente sind, wie ihre Oberklasse, abstrakt und erlauben es um weitere Aktionen erweitert zu werden. Die `DataFlowExternalAction` modelliert einen Aufruf eines Dienstes, der in einer Schnittstelle spezifiziert wurde. Eine `InternalDataSourceAction` gibt an, woher Daten stammen. In der jetzigen Modellierung unterteilt sie sich in `DataJoin` und `CreateData`. Bei `DataJoin` werden mehrere Daten zu einem neuen Datum vereinigt. `CreateData` hingegen erstellt ein neues Datum. Indem jeweils das Vorgänger und Nachfolger Feld der einzelnen `DataFlowAbstractAction`-Elemente spezifiziert wird, kann ein Datenfluss modelliert werden.

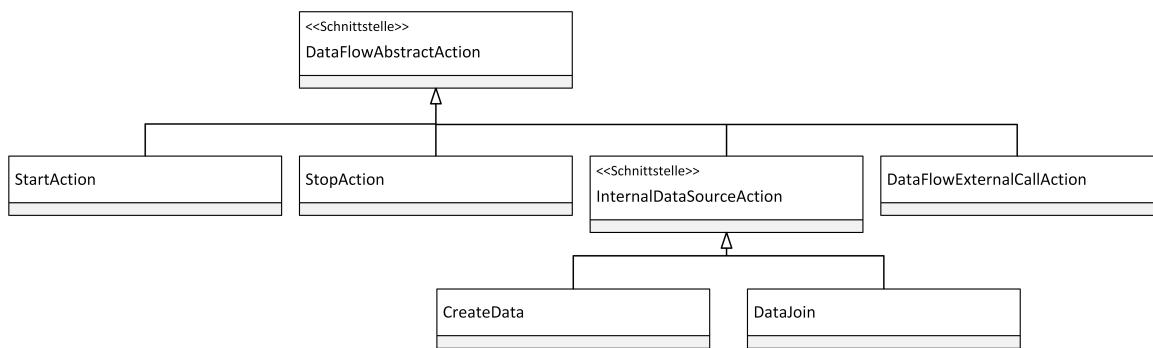


Abbildung 4.10.: Übersicht der Aktionen des DFSEFF

Das wohl wichtigste Konzept des DFSEFF ist die Möglichkeit Daten und Parameter zu verknüpfen. Dadurch können Daten durch das System verfolgt werden und schließlich in DataSets eingeordnet werden. Die Modellierung dazu ist in Abbildung 4.11 abgebildet. Mithilfe des Elements VariableBinding können DataFlowExternalActions um eine Verknüpfung von Daten und Parameter erweitert werden. Diese Verknüpfung wird als Binding modelliert. Ein Binding besteht aus einem Parameter, der aus einer Signatur stammt, die die DataFlowExternalAction aufruft und einer Datenklasse. Bindings werden in BindingContainer gesammelt und gruppiert.

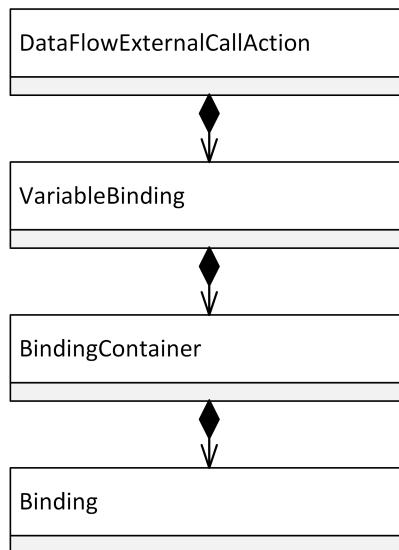


Abbildung 4.11.: Übersicht der Modellierung von Bindings

In Abbildung 4.12 wurde der Datenfluss des Kaufvorgangs aus Abschnitt 4.3 als DFSEFF modelliert. In dieser Modellierung gibt es die Datenklasse *ProductData*. Der Datenfluss fängt mit einer DataFlowExternalAction an, die den Dienst `checkForProduct(Product product)` aufruft. Sie enthält ein Binding, das den Parameter *product* mit der Datenklasse *ProductData* verknüpft. Als nächstes wird eine weitere DataFlowExternalAction ausgeführt. Sie ruft den Dienst `getProduct(Product product)` auf. Diese Aktion enthält ebenfalls ein Binding. Dieses verknüpft den Parameter *product* mit der Datenklasse *Product-*

Data. Schließlich wird mit `CreateData` ein neues Datum erstellt. Das neue Datum heißt *Product*. Mit dieser letzten Aktion endet der Datenfluss.

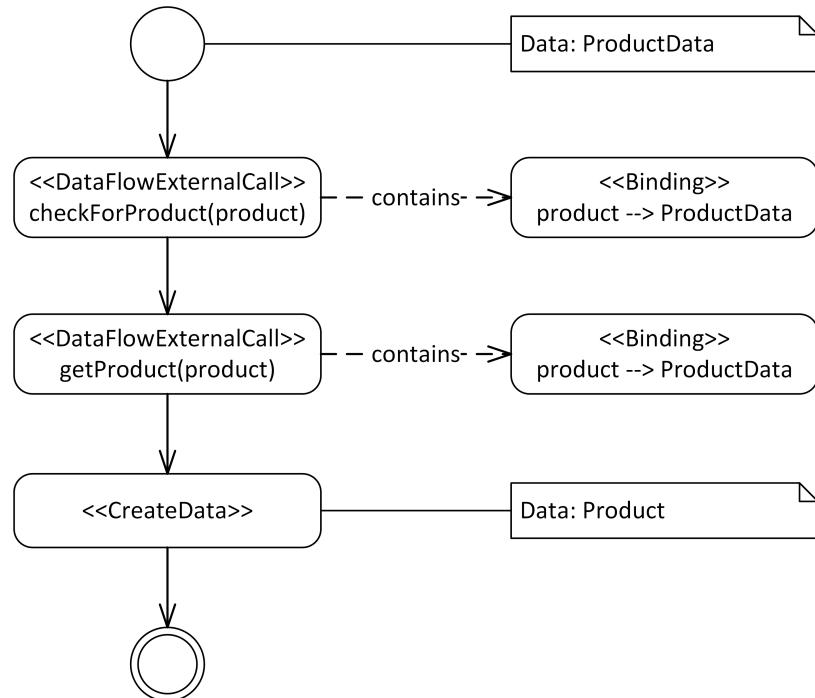


Abbildung 4.12.: Datenfluss des Dienstes `buy(Product product)` aus dem Laden-Szenario

4.7. Domänenexperte

Diese Erweiterung ermöglicht es, dem Domänenexperten, den Datenfluss im Nutzungsmodell zu modellieren. Die Erweiterung ist ähnlich aufgebaut wie die Erweiterung des Komponentenentwickler aus Abschnitt 4.6. Für die Modellierung wurden hier zwei Ansätze identifiziert. Zunächst wurde festgestellt, dass das Element `AbstractUserAction`, aus dem PCM, nicht RDSEFF spezifisch ist und deshalb wiederverwendet werden kann. `AbstractUserAction` ist vergleichbar zu der `AbstractAction` und modelliert einen Aufruf zu einem Dienst, der von einem Benutzer getätigkt wird. Der erste Ansatz wäre einen `DataFlowEntryLevelSystemCall`, ähnlich dem `EntryLevelSystemCall` aus Palladio, zu definieren. Der `EntryLevelSystemCall` modelliert einen Aufruf zu einem Dienst, der von einem PCM-System bereitgestellt wird. Dieser `DataFlowEntryLevelSystemCall` hätte `AbstractUserAction` als Basisklasse und würde zusätzlich Bindings enthalten. Dieser Ansatz hätte jedoch den Nachteil, dass der Domänenexperte bereits modellierte `EntryLevelSystemCalls` nochmal modellieren müsste, um den Datenfluss zu berücksichtigen. Der zweite Ansatz, der realisiert wurde, ist in Abbildung 4.13 abgebildet. Dabei wird das Nutzungsmodell um das Elemente `UsageVariableBinding` erweitert. `UsageVariableBinding` besitzt als Basisklasse das Element `VariableBinding` aus Abschnitt 4.6. `UsageVariableBinding` hat eine Referenz auf einen `EntryLevelSystemCall` aus dem PCM. Diese Referenz ist möglich, da der `EntryLevelSystemCall` nicht abhängig vom RDSEFF ist. Somit

wurden auch in diesem Ansatz Elemente wiederverwendet. Außerdem hat der Benutzer weniger Modellierungsaufwand, da er bereits modellierte `EntryLevelSystemCalls` wiederverwenden kann. Signaturen können auch hier wiederverwendet werden, denn die Daten müssen auch vom Benutzer in das System gelangen. Das Root-Element dieser Erweiterung ist das Element `UsageModelContainer`. Es enthält die `UsageVariableBindings`, die für die Modellierung des Datenflusses im Nutzungsmodell verwendet werden.

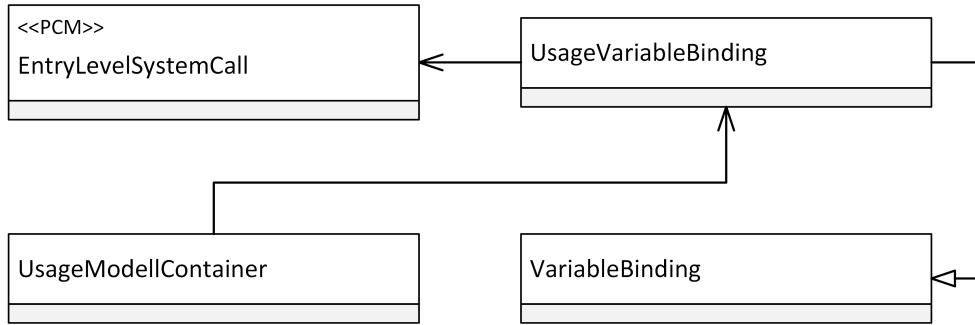


Abbildung 4.13.: Modellierung der Erweiterung für das Nutzungsmodell

In Abbildung 4.14 ist der Datenfluss im Nutzungsmodell, anhand des Beispiels aus Abschnitt 4.3, modelliert. In der Modellierung gibt es die Datenklassen `ProductData` und `CreditCardData`. Der Datenfluss beginnt mit einem `EntryLevelSystemCall`, der den Dienst `buy(Product product)` aufruft. Außerdem wird er von einem `UsageBinding`, dass den Parameter `product` mit der Datenklasse `ProductData` verknüpft, referenziert. Die Daten fließen zum Nächsten `EntryLevelSystemCall`, welcher wiederum den Dienst `pay(CreditCard creditCard)` aufruft. Auch dieser `EntryLevelSystemCall` wird von einem `UsageBinding`, dass den Parameter `creditCard` mit der Datenklasse `CreditCardData` verknüpft, referenziert. Danach endet der Datenfluss.

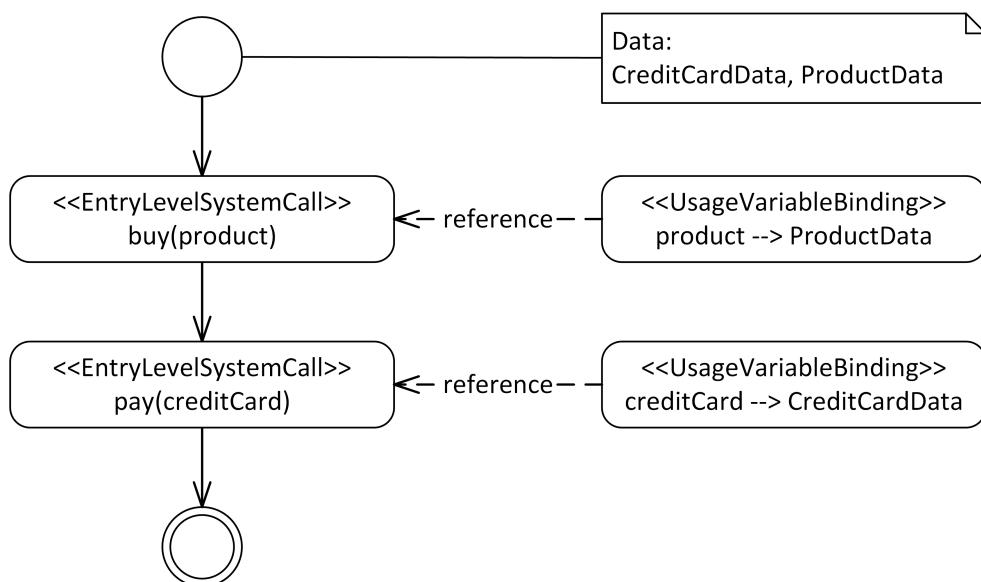


Abbildung 4.14.: Datenfluss, der vom Benutzer ausgeht

5. Validierung

In diesem Kapitel werden die restlichen Ergebnisse aus dem Prozess nach Völter und Stahl [18] präsentiert. Dabei wird die Modellierung aus Kapitel 4 validiert. Durch Prüfung der Modelleigenschaften nach Stachowiak, die in Unterabschnitt 2.1.1 beschrieben wurden, soll der Nachweis über die Einsatzignung der entwickelten Methodik aus Kapitel 4 erbracht werden. Die Eigenschaften Abbildung und Verkürzung sind gegeben, da es sich um eine Abbildung von Datenflüssen handelt und nur für die Analyse relevante Attribute erfasst werden. Schließlich muss die Eigenschaft Pragmatismus geprüft werden. Pragmatismus ist gegeben, wenn die Modelle für ihren späteren Einsatzzweck geeignet sind. In dieser Bachelorarbeit sind Analysen der Anwendungszweck. Im Rahmen der Validierung wird die Eignung der Modellierung für Vertraulichkeitsanalysen nach Kramer et. al. [9] nachgewiesen. Ob die Eigenschaft Pragmatismus erfüllt ist, soll mithilfe der Goal-Question-Metric-Methode (GQM-Methode) [1] bewertet werden. Die GQM-Methode spezifiziert Ziele für das zu validierende Konzept. Daraufhin werden zu den Zielen Fragen spezifiziert, mit denen die Erfüllung des Ziels überprüft werden soll. Schließlich werden Metriken festgelegt, durch die die Fragen beantwortet werden sollen.

Die Ziele, Fragen und Metriken, für die Validierung dieser Bachelorarbeit können aus Tabelle 5.1 entnommen werden. Dabei sollen zwei Ziele erreicht werden. Das Erste ist, dass eine Verhaltensspezifikation auf Datenflussebene in Palladio ermöglicht wird (G1). Das zweite Ziel ist, dass es möglich sein soll Datenflüsse für Qualitätsvorhersagen in Palladio zu benutzen (G2). Die folgenden Fragen sollen helfen zu überprüfen, ob die Ziele erreicht wurden.

Das erste Ziel (G1) soll mit zwei Fragen überprüft werden. Die erste Frage prüft, ob sich Daten und Datenflüsse mit der Modellierung aus Kapitel 4 modellieren lassen (Q1). Dazu soll ein Fallbeispiel, das in Abschnitt 5.1 beschrieben wird, mit Daten und Datenflüssen erweitert werden. Dieser Vorgang wird in Unterabschnitt 5.1.3 beschrieben. Mithilfe einer Metrik soll geprüft werden, wie viele Daten und Datenflüsse nicht modelliert werden können (M1). Die zweite Frage bezieht sich darauf, wie gut die Datenflussmodellierung in das PCM eingebunden ist (Q2). Die Frage soll mit zwei Metriken beantwortet werden. Die Erste prüft, wie viele bereits modellierte Informationen nochmal modelliert werden müssen, damit ein Datenfluss modelliert werden kann (M2). Die Zweite prüft wie viele Elemente neu modelliert werden müssen, damit ein Datenfluss modelliert werden kann (M3).

Auch das zweite Ziel (G2) soll mit zwei Fragen überprüft werden. Die erste Frage bezieht sich darauf, ob Daten in DataSets eingeordnet werden können (Q3), da dies das Analyseziel der verwendeten Vertraulichkeitsanalyse, nach Kramer et. al. [9], ist. Aus dieser Zuordnung kann schließlich Vertraulichkeit abgeleitet werden. Die dazugehörige Metrik prüft, wie viele Daten nicht in DataSets eingeordnet werden können (M4). Die Daten werden mit einer Datenflussanalyse und Transformation in DataSets eingeordnet. Diese

5. Validierung

Ziel 1 (G1): Verhaltensspezifikation auf Datenflussebene in Palladio ermöglichen.

Frage	Metrik
<i>Q1:</i> Lassen sich Daten und Datenflüsse für das Travelplanner-Fallbeispiel modellieren?	<i>M1:</i> Wie viele Daten oder Datenflüsse lassen sich für das Fallbeispiel nicht modellieren?
<i>Q2:</i> Wie gut funktioniert die Einbindung der Datenflussmodellierung in das PCM?	<i>M2:</i> Wie viele Informationen müssen nochmal modelliert werden? <i>M3:</i> Wie viele Elemente müssen neu modelliert werden, um den Datenfluss modellieren zu können?

Ziel 2 (G2): Nutzung der Datenflüsse für Qualitätsvorhersagen in Palladio.

Frage	Metrik
<i>Q3:</i> Werden Daten in DataSets eingeordnet?	<i>M4:</i> Wie viele Daten können nicht eingeordnet werden?
<i>Q4:</i> Ist die Modellierung für Datenflussanalysen nutzbar?	<i>M5:</i> Entsteht nach Analyse und Transformation ein valides Modell? <i>M6:</i> Kann die Vertraulichkeitsanalyse ausgeführt werden?

Tabelle 5.1.: Ziele, Fragen und Metriken für die Validierung, nach der GQM-Methode

wird in Abschnitt 5.2 beschrieben. Die zweite Frage prüft, ob die Modellierung für Datenflussanalysen nutzbar ist (Q4). Im Rahmen dieser Bachelorarbeit soll dies mithilfe der Vertraulichkeitsanalyse nach Kramer et. al. [9] überprüft werden. Dazu soll geprüft werden, ob die Datenflussanalyse und Transformation, aus Abschnitt 5.2, ein valides Modell erstellt (M5). Außerdem soll geprüft werden, ob die Vertraulichkeitsanalyse mit der Modellierung dieser Bachelorarbeit lauffähig ist (M6).

Das Vorgehen der Validierung ist zunächst das Travelplanner-Fallbeispiel aus [20] mit der Modellierung aus Kapitel 4 um Daten und Datenflüsse zu erweitern. Das Fallbeispiel wird auch in der Arbeit von Kramer et. al. [9] für die Vertraulichkeitsanalyse verwendet. Somit gibt es einen Gold-Standard, mit dem das Ergebnis verglichen werden kann. Die mittels Datenflüssen spezifizierte Modellinstanz soll anschließend in eine Modellinstanz transformiert werden, mit der die Vertraulichkeitsanalyse nach Kramer et. al. [9] möglich ist. Dazu muss eine Modelltransformation geschrieben werden. Elemente, die nicht abgebildet werden können, müssen ggf. auf Meta-Modellierungsebene ergänzt werden. Das Ergebnis ist eine Modellinstanz, in der alle Parameter in DataSets eingeordnet sind. Vor der Transformation werden lediglich einzelne Datenzuordnungen spezifiziert. Die Parameterzuordnung wird während der Transformation abgeleitet. Im Anschluss wird die Modellinstanz der Transformation und die Modellinstanz aus der Arbeit von Kramer et. al. [9] verglichen und ggf. die Vertraulichkeitsanalyse durchgeführt. Schließlich werden die GQM-Fragen beantwortet.

5.1. Fallbeispiel

Im Folgenden wird das erstellte Referenzmodell beschrieben. Es ist das Ergebnis aus dem zweiten Schritt des Prozesses nach Völter und Stahl. Es soll als Fallbeispiel für die Vertraulichkeitsanalyse dienen.

Das benutzte Beispiel basiert auf [20]. Es handelt sich dabei um ein Reiseplanungssystem, das aus einer Reiseagentur (TravelAgency), Fluglinie (Airline), Reiseplaner (TravelPlanner) und einem Kreditkartenzentrum (CreditCardCenter) besteht. Das System bietet Dienste zum Buchen eines Flugs und Bestätigen der Kreditkarteninformation, für die verschiedenen Gruppen Fluglinie (Airline), Reiseagentur (TravelAgency) und Kunde (User) an. Das Vorgehen des Systems wird im Folgenden beschrieben. Zunächst verbindet sich der Reiseplaner mit der Reiseagentur. Daraufhin verbindet sich die Reiseagentur mit einer Airline, welche passende Flüge liefert. Die Flüge werden an die Reiseagentur weitergegeben. Der Benutzer kann einen Flug auswählen und mit seinen Kreditdaten direkt bei der Fluglinie buchen. Im Anschluss zahlt die Airline der Reiseagentur eine Provision. Abbildung A.1 zeigt dieses Verhalten. Die Lebenslinien repräsentieren den Benutzer, den Reiseplaner, das Kreditkartenzentrum und die Reiseagentur. Die Systeme kommunizieren über Nachrichten miteinander. Als Erweiterung zu klassischen UML-Sequenzdiagrammen sind zu den Nachrichten in geschweiften Klammern die Sicherheitsdomänen angehängt, z.B. User, TravelAgency, Airline.

Für die Bachelorarbeit wurde das Fallbeispiel modifiziert, damit es zur PCM-Methodik kompatibel ist. In der bereits modellierten Fallstudie wurde gefordert, dass der Benutzer Schnittstellen implementiert. Dieses Verhalten ist in PCM nicht vorgesehen. Wird das Verhalten innerhalb von Komponenten und durch das Nutzungsmodell nicht untersucht, stellt das kein Problem dar. Da aber in dieser Bachelorarbeit das Verhalten innerhalb von Komponenten und des Nutzungsmodells betrachtet wird, müssen Änderungen an der Modellierung vorgenommen werden, damit sie konform zur PCM-Methodik ist. Damit ist gemeint, dass der Benutzer keine Schnittstelle bereitstellt, sondern die angebotenen Systemschnittstellen lediglich nutzt. Kontrollflüsse starten immer von außen. Außerdem fehlten Parameter und Rückgabewerte. Somit war das System nicht lauffähig, weil benötigte Komponenten, z.B der Benutzer, nicht spezifiziert waren.

Das angepasste Sequenzdiagramm ist in Abbildung 5.1 abgebildet. Dabei wurden alle Aufrufe zum Benutzer entfernt. Außerdem wurde das Anfordern der Kreditkarteninformationen in mehrere Aufrufe aufgeteilt. Im folgendem Abschnitt werden das Komponenten-Repository-Modell, das Ressourcen-Umgebungs-Modell und das Nutzungsmodell beschrieben.

5.1.1. Beschreibung der PCM-Modelle

In Abbildung 5.2 findet sich die graphische Darstellung des Komponenten-Repository-Modells. Aus den Klassen Travelplanner, CreditCardCenter, TravelAgency und Airline wurde jeweils eine BasicComponent. Die Komponenten kommunizieren über Schnittstellen miteinander. Dabei ist der Travelplanner die zentrale Komponente. Sie kommuniziert über die Schnittstelle FlightOffers mit der TravelAgency und der Airline, um Flüge zu erhalten. Mithilfe der Schnittstelle Booking kann der Travelplanner mit der Airline kom-

5. Validierung

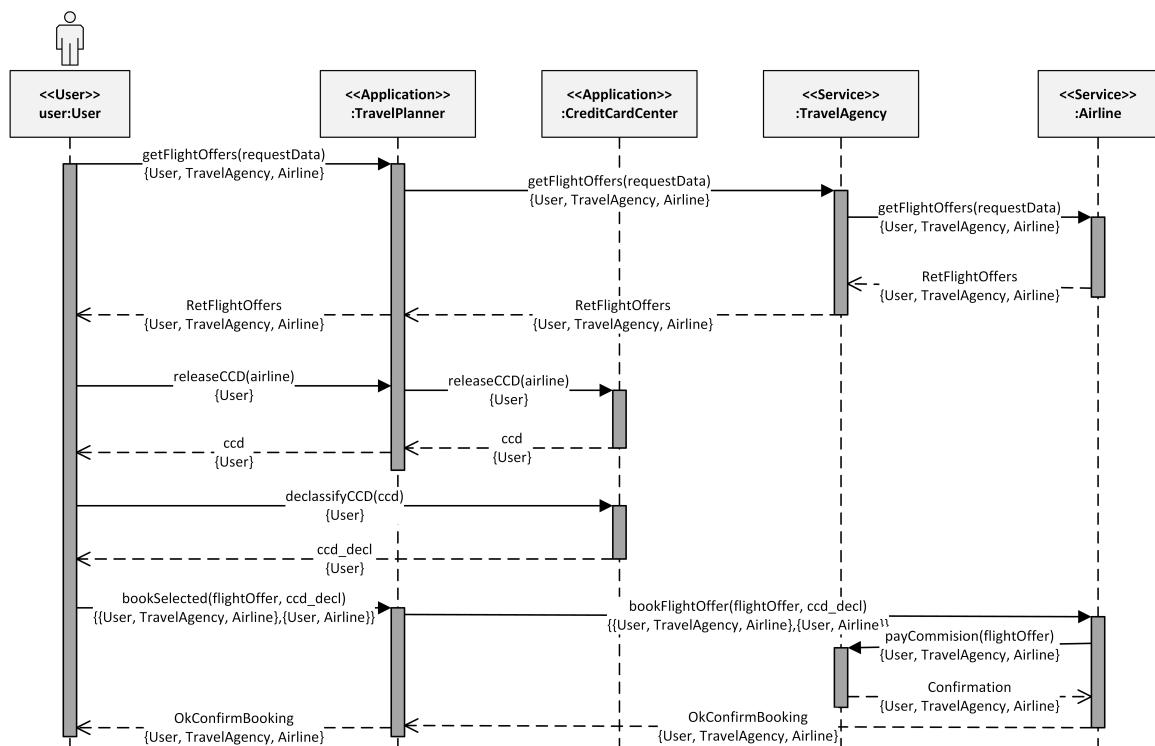


Abbildung 5.1.: Buchung eines Fluges

munizieren, um einen Flug zu buchen. Über die Schnittstelle `Declassification` kann mit der `CreditCardCenter`-Komponente kommuniziert werden. Mithilfe dieser Schnittstelle kann eine Freigabe der Kreditkartendaten erhalten werden.

Da das Modell nicht vollständig war und nicht zur PCM-Methodik gepasst hat, musste es angepasst und vervollständigt werden. Dazu wurden alle Schnittstellen entfernt, die eine Aktion vom Benutzer als Aufgerufenem statt Aufrufendem gefordert haben. Dazu gehörten die Schnittstellen `Input` und `Confirmation`. Damit der Benutzer trotzdem mit dem System interagieren kann wurde die Schnittstelle `BookingSelection` erweitert. Sie wird vom `TravelPlanner` bereitgestellt. Diese Schnittstelle enthält vier Signaturen, über die der Benutzer mit dem System interagieren kann. Mit `BookingSelection.bookSelected` kann der Benutzer einen gewünschten Flug, mit seiner Kreditkarte buchen. Mithilfe des Dienstes `BookingSelection.getFlightOffers` kann der Benutzer Flüge anfordern. Schließlich kann der Benutzer mit `BookingSelection.releaseCCD` und `BookingSelection.declassifyCCD` seine Kreditkartendaten für eine bestimmte Fluglinie anfordern und freigeben. Außerdem wurde in dem Modell die Signatur der Schnittstelle `Commision` angepasst. Die Signatur `void payCommision()` wurde in `bool payCommision(int offerId)` geändert, damit die Fluglinie und Reiseagentur wissen, für was eine Provision gezahlt werden soll. Der geänderte Rückgabetyp signalisiert, ob der Vorgang erfolgreich war. Die Änderungen wurden in Abbildung A.2 markiert.

Die graphische Darstellung des Ressourcen-Umgebungs-Modells ist in Abbildung 5.3 abgebildet. Das Ressourcen-Umgebungs-Modell besteht aus drei `ResourceContainer`s und drei `LinkingResources`. Der erste `ResourceContainer` modelliert ein Handy (`MobilePhone`).

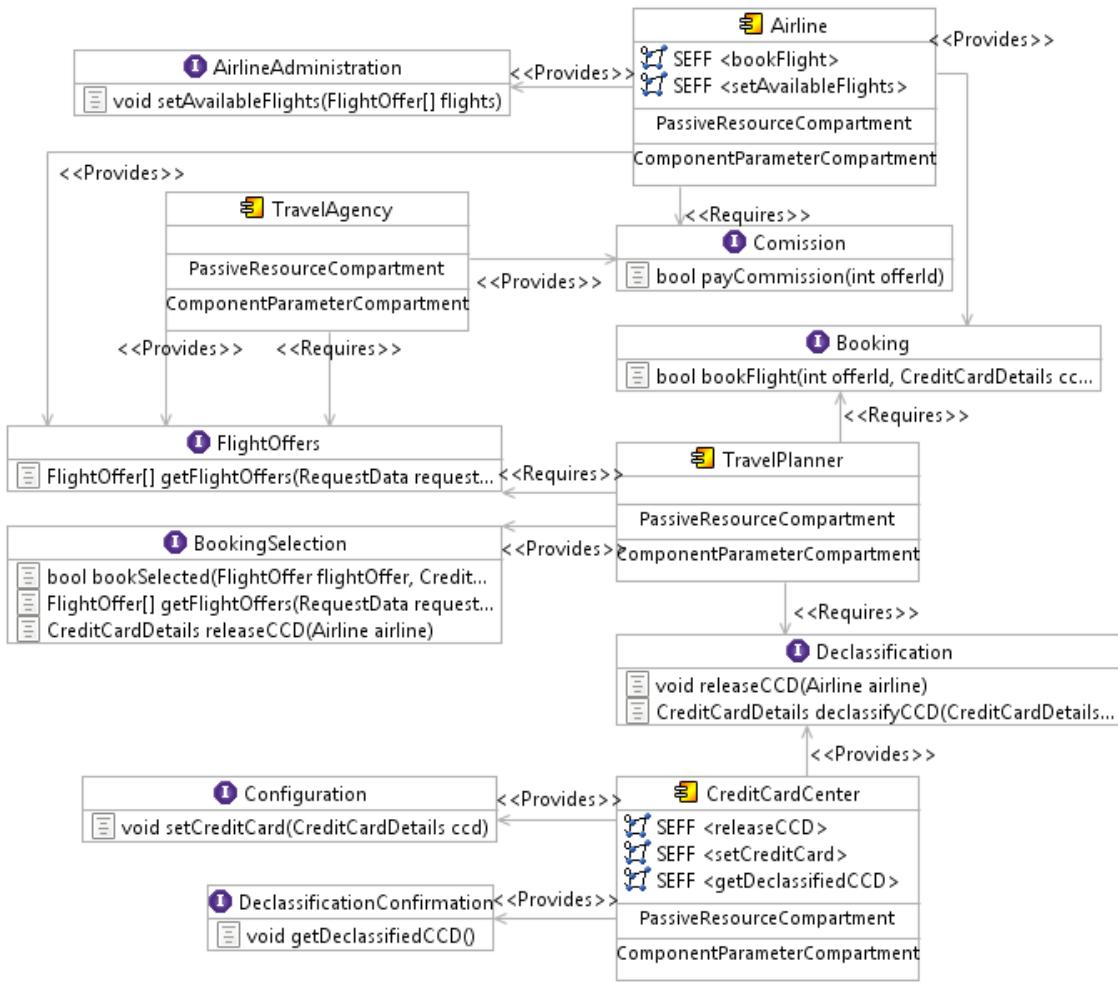


Abbildung 5.2.: Komponenten-Repository-Modell des Travelplanner

Dieser ist mit dem zweiten ResourceContainer, der einen Server der Fluglinie (Airline-Server) modelliert, über zwei der LinkingResources verbunden. Die Erste modelliert eine 4G- und Internetverbindung (4g+Internet). Die zweite LinkingResource modelliert eine Wlan- und Internetverbindung (OpenWifi+Internet). AirlineServer ist außerdem mit dem dritten ResourceContainer verbunden. Dieser modelliert den Server der Reiseagentur (AgencyServer). Travelagency und AirlineServer sind über eine LinkingResource verbunden, die eine Internetverbindung modelliert (Internet).

Damit das PCM-Modell vervollständigt wird, wurde der TravelPlaner um ein Nutzungsmodell erweitert. Die graphische Darstellung ist in Abbildung 5.4 abgebildet. Das Nutzungsmodell besteht aus vier EntryLevelSystemCalls, die die Benutzerinteraktion aus dem Sequenzdiagramm aus Abbildung 5.1 darstellen. Dabei ruft der Benutzer zunächst den Dienst `BookingSelection.getFlightOffers` auf, um Flüge abzurufen. Im Anschluss gibt der Benutzer seine Kreditkartendaten frei, indem er die beiden Dienste `BookingSelection.releaseCCD` und `Declassification.declassify` aufruft. Schließlich bucht er

5. Validierung



Abbildung 5.3.: Ressource-Umgebungs-Modell des Travelplanner

mit dem Aufruf `Bookingselection.bookSelected` den gewünschten Flug bei der Fluglinie und bezahlt, mit den freigegebenen Kreditkartendaten.

In Abbildung A.3 ist das Systemmodell und in Abbildung A.4 ist das Komponenten-Allokations-Modell des Travelplanner abgebildet. Die Modelle werden in der Transformation und Vertraulichkeitsanalyse verwendet. Sie werden im weiteren Verlauf der Bachelorarbeit nicht diskutiert, da keine Erweiterungen für die Modelle entwickelt wurden.

5.1.2. Vertraulichkeitsmodell

In [9] wurde ein Meta-Modell erstellt, um Vertraulichkeit zu modellieren und später den Datenfluss analysieren zu können. Im Folgenden wird ein Teil dieses Meta-Modells beschrieben. Dieser Teil des Meta-Modells wird benötigt um das Travelplanner-Fallbeispiel zu erweitern, damit es im Anschluss von der Vertraulichkeitsanalyse aus [9] analysiert werden kann.

Eines dieser Elemente aus dem Meta-Modell ist `DataSet`. Ein `DataSet` modelliert verschiedene Arten von Informationen, die in einem System verarbeitet werden. Die Idee ist, dass man diese `DataSets` Akteuren zuordnet. Ein Akteur kann dabei eine Person oder ein System sein. Ein Akteur hat nur Zugriff auf Informationen aus den `DataSets`, die ihm zugeordnet wurden. `DataSets` können z.B. der Benutzer, die Fluglinie oder die Reiseagentur sein. Mit dem Element `Location` können Orte für Resource-Container und Linking-Resources spezifiziert werden. Ein Ort kann z.B. der Raum sein, indem sich der Server

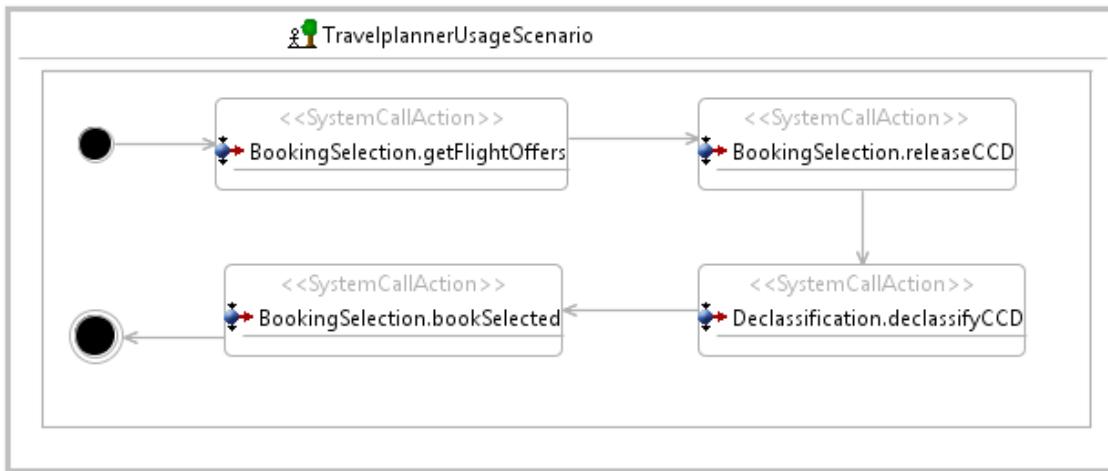


Abbildung 5.4.: Nutzungsmodell des Travelplanner

der Reiseagentur oder der Fluglinie befindet. Das Element TamperProtection erlaubt es einen Sicherheitsmechanismus für Resource-Container und Linking-Resources zu spezifizieren. Mithilfe von LocationAndTamperProtectionPair kann ein Paar aus einer Location und einer TamperProtection spezifiziert werden. So ein Paar kann z.B. der Raum indem der Server der Reiseagentur steht und ein Passwort sein. Da im Fallbeispiel Sicherheitsmechanismen nicht betrachtet werden, wird das Feld für den Sicherheitsmechanismus im Weiteren nicht betrachtet. Schließlich kann mit dem Element ParametersAndDataPair ein Parameter aus dem Komponenten-Repository-Modell mit einem DataSet verknüpft werden. Ein Beispiel für ein solches ParametersAndDataPair könnte der Parameter *ccd*, der die Kreditkartendaten enthält und das DataSet *Benutzer* sein. Das heißt, dass Akteure mit Zugang zu dem DataSet *Benutzer* auch Zugang zu dem Parameter *ccd* haben.

Mit diesem Modell, lassen sich Eingabe und Ausgabe eines Systems spezifizieren. Den Datenfluss innerhalb von Komponenten zu modellieren ist, mit dieser Modellierung, nicht möglich. Komponenten werden wie eine Blackbox betrachtet. Bei der Modellierung aus Kapitel 4 ist das anders. Dort kann der Datenfluss innerhalb von Komponenten modelliert werden.

Das Vertraulichkeitsmodell enthält außerdem ein Angreifermodell. Ein Angreifer enthält dabei, zu welchem Ort (Location) er Zutritt hat und welche DataSets ihm zugeordnet wurden. Dadurch kann die Vertraulichkeitsanalyse prüfen, ob jeder Angreifer nur auf die Daten Zugriff hat, auf die er Zugriff haben darf. Ein Angreifer kann z.B. ein Mitarbeiter der Reiseagentur sein, der Zugriff zum Server der Reiseagentur und dem DataSet Reiseagentur hat. Im Rahmen dieser Bachelorarbeit werden Angreifer nicht betrachtet. Deshalb wird das Angreifermodell, aus dem Vertraulichkeitsmodell, für die Validierung, wiederverwendet.

Wie bereits bei der Beschreibung der verschiedenen Analysen in Abschnitt 3.4 erwähnt, muss für die Vertraulichkeitsanalyse Eingabe und Ausgabe eines Systems spezifiziert werden. Dazu müssen die Elemente aus dem Vertraulichkeitsmodell in das zu untersuchende System integriert werden. Dies ist mithilfe von MDSDProfiles möglich [11]. MDSDPro-

5. Validierung

files basiert auf EMF-Profiles [12] und erlaubt es das Meta-Modell mit einem Profil zu erweitern, ohne dieses ändern zu müssen. Das Profil besteht dabei aus Stereotypen. Ein Stereotyp besteht aus einer Referenz zu einem Element, dass erweitert werden soll und aus einer Referenz zu dem Element, dass erweitern soll. Mithilfe dieser Stereotypen können Elemente durch Annotationen erweitert werden.

Die Arbeit von Kramer et. al. [9] bietet ein solches Modell an. Damit ist es möglich das Komponenten-Repository- und das Ressourcen-Umgebungs-Modell mit Elementen aus dem Vertraulichkeitsmodell zu erweitern. Um die Ergebnisse aus der Transformation in die beiden Modelle zu integrieren, benötigt es zwei Stereotypen aus dem Profiles-Modell. Die Beiden sind in Abbildung 5.5 abgebildet. Mithilfe des InformationFlow-Stereotypen lässt sich das Komponenten-Repository-Modell um einen Informationsfluss erweitern. Dabei können ParameterAndDataPair-Elemente an eine Signatur oder Schnittstelle im Komponenten-Repository-Modell angehängt werden. Der LocationAndTamper-Stereotyp erweitert das Ressourcen-Umgebungs-Modell um eine Zugangsspezifikation. Damit ist es möglich Resource-Container und Linking-Resources im Ressourcen-Umgebungs-Modell mit dem Element LocatonAndTamperProtectionPair zu erweitern .

Diese beiden Modelle, das Profiles-Modell und die restlichen PCM-Modelle dienen als Eingabe für die Vertraulichkeitsanalyse.

Da eine Modellinstanz der Modellierung aus Kapitel 4 durch die Analyse analysiert werden soll, muss diese zunächst transformiert werden. Im Anschluss muss die Transformation mithilfe der Stereotypen aus dem Profiles-Modell mit dem Komponenten-Repository- und Ressourcen-Umgebungs-Modell verknüpft werden. Anschließend kann die Vertraulichkeitsanalyse angewendet werden.

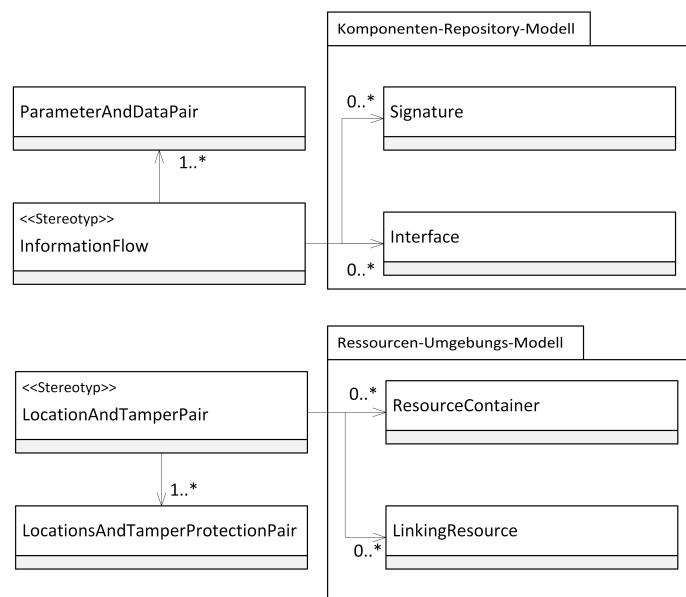


Abbildung 5.5.: Stereotypen, mit denen das PCM um Elemente, aus dem Vertraulichkeitsmodell, erweitert werden kann

5.1.3. Datenflussmodellierung

In diesem Abschnitt wird die Daten- und Datenflussmodellierung für den Travelplanner, mit dem in Kapitel 4 entwickeltem Meta-Modell, beschrieben. Die Modellierung basiert dabei, auf dem beschriebenen Modell aus Abschnitt 5.1 und der Spezifikation aus [9].

Zunächst wurden für jeden Resource-Container, aus Abbildung 5.3, ein ResourceContainerPropertyContainer erstellt, der auf den jeweiligen Resource-Container referenziert. Dabei sind die Resource-Container MobilePhone mit dem Ort UserControlled, der AirlineServer mit dem Ort Airline und AgencyServer mit dem Ort Agency verknüpft. Für die einzelnen Linking-Resources wurden auch jeweils ein LinkingResourcePropertyContainer erstellt. Dabei sind die Linking-Resources 4G+Internet mit dem Ort Street und Internet, OpenWifi+Internet mit dem Ort Coffeeshop und Internet mit dem Ort Internet verknüpft. In Abbildung 5.6 ist die Erweiterung des Resource-Containers MobilePhone und der Linking-Resource 4G+Internet beispielhaft dargestellt.

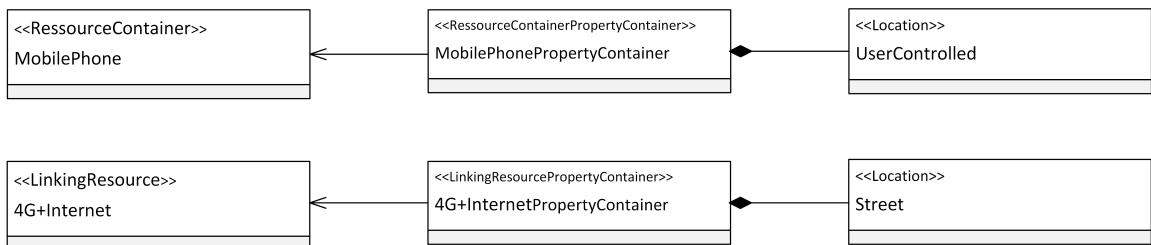


Abbildung 5.6.: Erweiterung von *MobilePhone* und *4G+Internet* um einen Standort

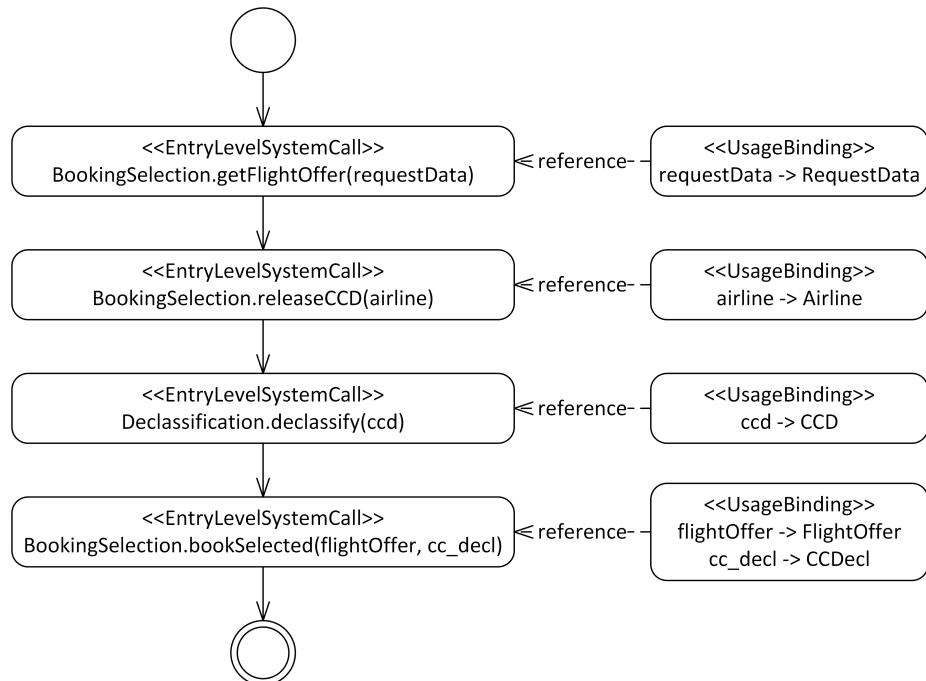


Abbildung 5.7.: Erweiterung des Nutzungsmodells um Bindings

5. Validierung

Als Nächstes wurde mithilfe der Erweiterung für den Domänenexperten, aus Abschnitt 4.7, das Nutzungsmodell um einen Datenfluss erweitert. Dazu wurden die EntryLevel-SystemCalls des Nutzungsmodells, aus Abbildung 5.4, jeweils von einem Binding referenziert. Dabei sind die Parameter aus `BookingSelection.bookSelected` mit den Datenklassen `FlightOffer` und `CCDecl` verknüpft. Der Parameter `requestData` des Dienstes `BookingSelection.getFlightOffers` ist mit der Datenklasse `RequestData` verknüpft. Schließlich sind die Parameter der beiden Dienste `BookingSelection.releaseCCD` und `Declassification.releaseCCD` mit der Datenklasse `Airline` verknüpft. In Abbildung 5.7 ist die Erweiterung des Nutzungsmodells abgebildet.

Mithilfe von DFSEFFs, die in Abschnitt 4.6 beschrieben wurden, wurde der Datenfluss innerhalb von Komponenten modelliert und somit das Komponenten-Repository-Modell erweitert. Die Komponente `Travelplanner` beinhaltet vier DFSEFFs. Der Erste modelliert den Datenfluss innerhalb von `BookingSelection.getFlightOffers`. Innerhalb des DFSEFF, werden die Daten an den Dienst `TravelAgency.getFlightOffers` weitergegeben. Dies wird mit einer `DataFlowExternalAction` modelliert. Sie enthält ein `Binding`, das den Parameter `requestData` mit der Datenklasse `RequestData` verknüpft. Im Anschluss wird mit `CreateData` das Rückgabedatum `FlightOfferReturn` erstellt. Abbildung 5.8 veranschaulicht das Verhalten.

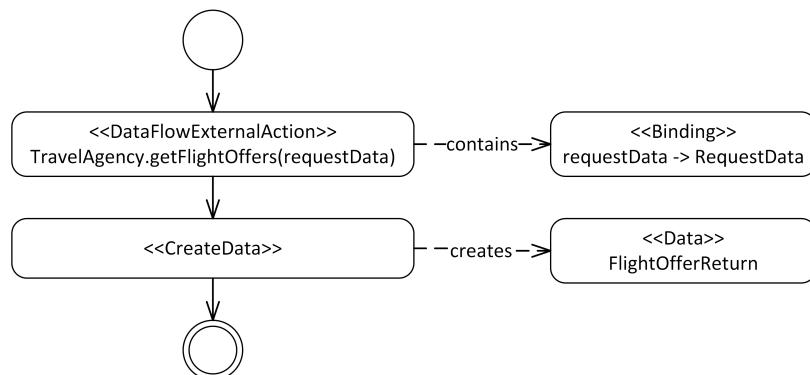


Abbildung 5.8.: DFSEFF des Aufrufs `BookingSelection.getFlightOffers`

Der Datenfluss innerhalb von `TravelAgency.getFlightOffers` wird in einem DFSEFF in `TravelAgency` modelliert. Der Aufruf des Dienstes `Airline.getFlightOffers` wird, auch hier, mit einer `DataFlowExternalAction` modelliert. Sie enthält ein `Binding`, das den Parameter `requestData` mit der Datenklasse `RequestData` verknüpft. Im Anschluss wird mit `CreateDataAction` das Rückgabedatum `FlightOfferReturn` erstellt. Das Verhalten ist in Abbildung 5.9 abgebildet.

Der zweite DFSEFF, im `Travelplanner`, modelliert den Datenfluss innerhalb des Dienstes `BookingSelection.releaseCCD`. Der Aufruf des Dienstes `CreditCardCenter.releaseCCD` wird mithilfe einer `DataFlowExternalAction` modelliert. Sie enthält ein `Binding`, das den Parameter `airline` mit der Datenklasse `Airline` verknüpft. Daraufhin wird mit `CreateData` das Rückgabedatum `CCD` erstellt. In Abbildung 5.10 wird dieses Verhalten veranschaulicht.

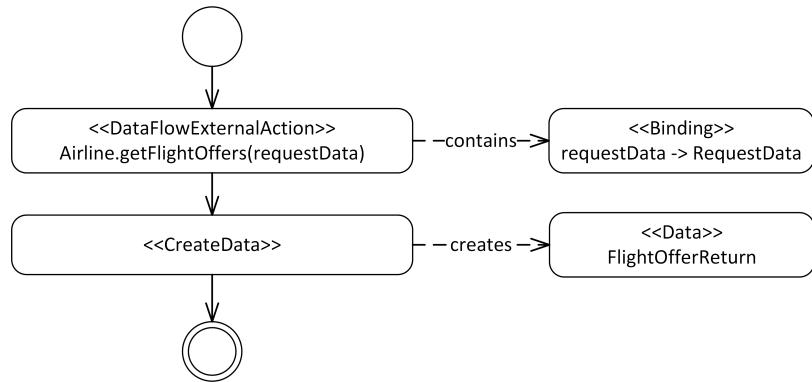


Abbildung 5.9.: DFSEFF des Aufrufs `TravelAgency.getFlightOffers`

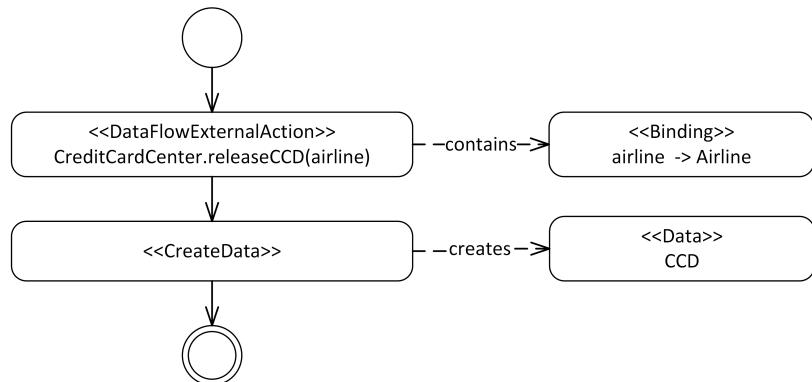


Abbildung 5.10.: DFSEFF des Aufrufs `BookingSelection.releaseCCD`

Im dritten DFSEFF des Travelplanner, wird der Datenfluss innerhalb des Dienstes `Declassification.declassifyCCD` modelliert. Dabei wird mit `CreateData` das Rückgabedatum `CCDecl` erstellt. Eine Veranschaulichung des Verhaltens ist in Abbildung 5.11 abgebildet.

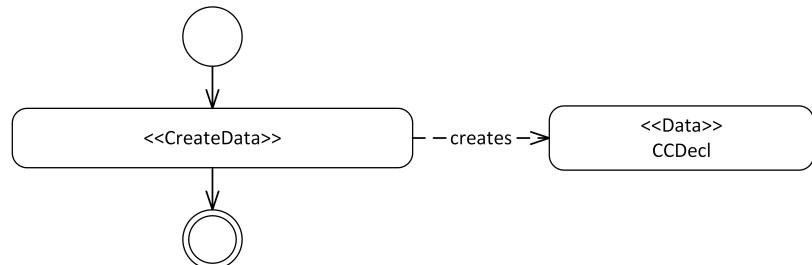


Abbildung 5.11.: DFSEFF des Aufrufs `Declassification.declassifyCCD`

Im vierten und letzten DFSEFF des Travelplanner wird der Datenfluss innerhalb von `BookingSelection.bookSelected` modelliert. Der Aufruf des Dienstes `Airline.bookFlightOffer` wird mithilfe einer `DataFlowExternalAction` modelliert. Sie enthält zwei Bindings. Das Erste verknüpft den Parameter `flightOffer` mit der Datenklasse `FlightOffer` und das Zweite verknüpft den Parameter `ccd_decl` mit der Datenklasse `CCDecl`. Schließlich

5. Validierung

wird mit CreateData das Rückgabedatum *BookingConfirmation* erstellt. Das Verhalten ist in Abbildung 5.12 abgebildet.

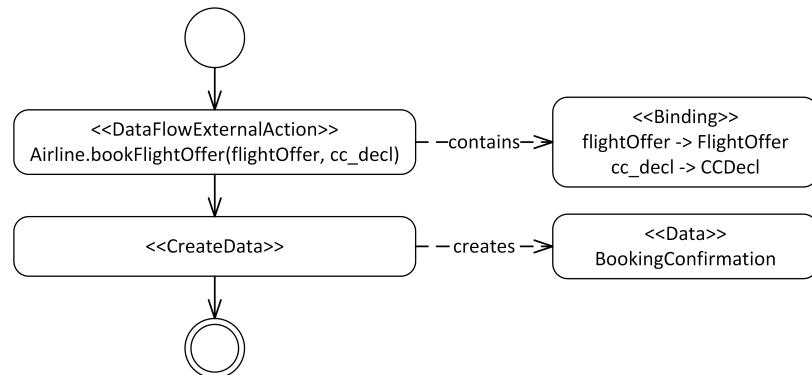


Abbildung 5.12.: DFSEFF des Aufrufs *BookingSelection.bookSelected*

Der letzte DFSEFF der Modellierung, beschreibt den Datenfluss innerhalb des Dienstes *Booking.bookFlight*. Der Aufruf des Dienstes *TravelAgency.payCommision* wird mit einer DataFlowExternalAction modelliert. Sie enthält ein Binding, das den Parameter *flightOffer* mit der Datenklasse *FlightOffer* verknüpft. Mit CreateData wird im Anschluss das Rückgabedatum *BookingConfirmation* erstellt. In Abbildung 5.13 ist das Verhalten veranschaulicht.

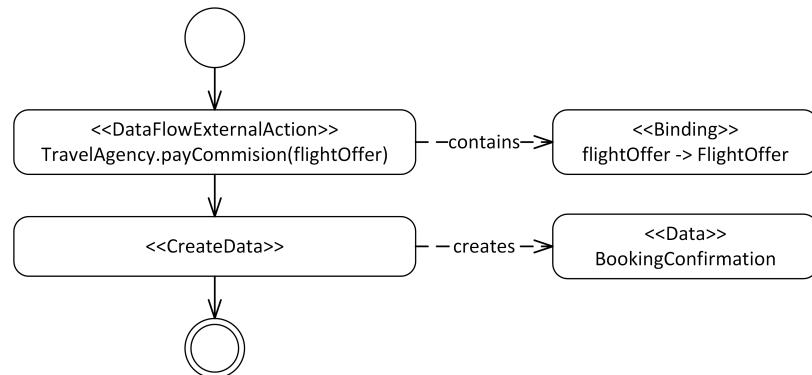


Abbildung 5.13.: DFSEFF des Aufrufs *Airline.bookFlight*

5.1.4. Angreifermodell

Vor der Vertraulichkeitsanalyse muss die Travelplanner-Modellinstanz, die mit Daten und Datenflüssen erweitert wurde, mit der Transformation aus Abschnitt 5.2 transformiert werden. Im Anschluss muss noch ein Angreifermodell erstellt werden. Da die Modellierung der Bachelorarbeit keinen Angreifer vorsieht, wird das Angreifermodell aus dem Vertraulichkeitsmodell (Unterabschnitt 5.1.2) wiederverwendet. Dieses Angreifermodell ist in Abbildung 5.14 abgebildet. Das abgebildete Angreifermodell setzt eine abgeschlossene Transformation und Datenflussanalyse voraus. In dem Angreifermodell gibt

es drei Angreifer. Der Erste ist der Benutzer (*User*). Dieser darf Zugang zu den Daten aus dem DataSet *MobilePhone* haben. Außerdem hat der physikalischen Zugang zum Resource-Container *MobilePhone* und zu den Linking-Resources *4G+Internet* und *Open-Wifi+Internet*. Der Angreifer *Airline*, darf nur auf Daten aus dem DataSet *AirlineServer* zugreifen. Physikalischen Zugriff hat dieser nur auf den Resource-Container *AirlineServer*. Der letzte Angreifer ist die *TravelAgency*. Diese darf auf Daten aus dem DataSet *AgencyServer* zugreifen und hat physikalischen Zugang zum Resource-Container *AgencyServer*.

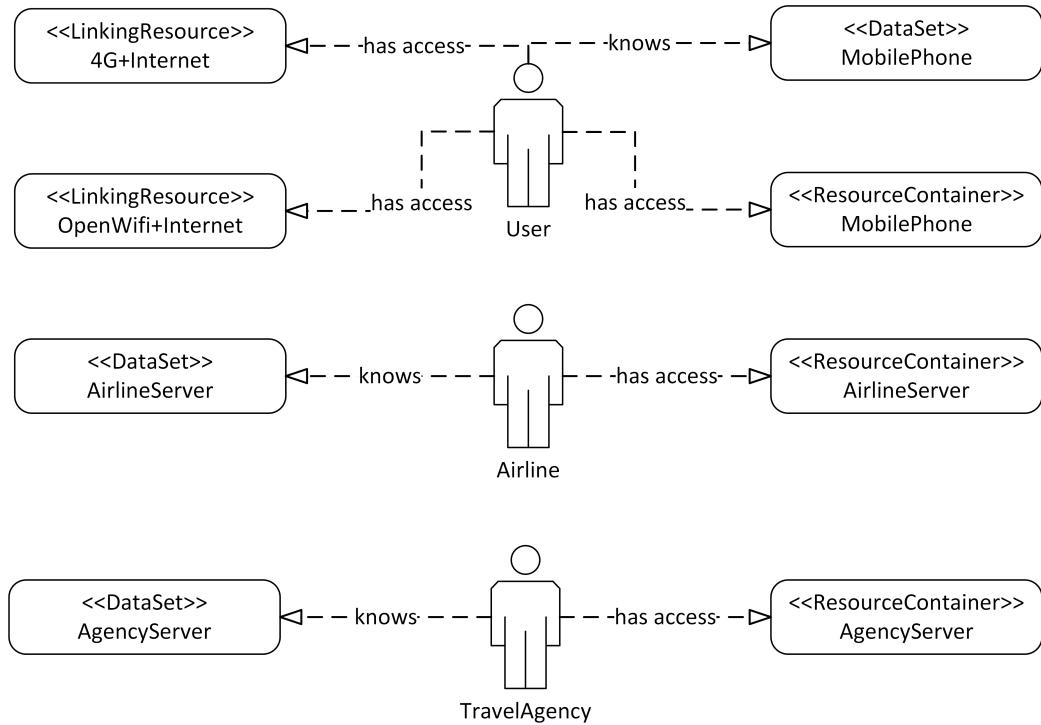


Abbildung 5.14.: Angreifermodell der Modellinstanz, nachdem sie transformiert wurde

5.2. Transformation und Datenflussanalyse

Im Folgenden wird die Transformation beschrieben. Sie ist das letzte Ergebnis dieser Arbeit, nach dem Völter und Stahl Prozess. Es handelt sich um eine Transformation, die eine Modellinstanz aus Kapitel 4 in eine Modellinstanz des Vertraulichkeitsmodells aus Unterabschnitt 5.1.2 transformiert, um im Anschluss eine Vertraulichkeitsanalyse nach Kramer et. al. [9] durchführen zu können.

Die Transformation besteht aus einer Generator- und einer Transformator-Klasse. Der Generator ist für das Laden der Ressource und Modelle, sowie das Abspeichern des transformierten Modells zuständig. Der Transformator übernimmt die eigentliche Transformation. Die Transformation überträgt Elemente aus der Modellierung dieser Bachelorarbeit in Elemente des Vertraulichkeitsmodells. Außerdem werden die Datenflüsse analysiert. Nach der Analyse werden passende Elemente für das Vertraulichkeitsmodell erstellt. Im Folgenden soll diese Analyse und Transformation beschrieben werden.

5. Validierung

Die Datenflussanalyse verfolgt die Daten durch das System, die zuvor vom Benutzer eingegeben wurden. Dabei wird für jedes Datum, die durchlaufenden Komponenten gesammelt. Das Sammeln geschieht mithilfe der Klasse `BindingAndComponentPair`. Die Klasse besteht aus einem Feld für ein `Binding` und einem Feld für alle Komponenten, durch die dieses `Binding` geflossen ist. Ist der Datenfluss wieder beim Ursprung angelangt, werden die einzelnen Daten in `DataSets` eingeordnet. Dabei wird für jeden Resource-Container aus dem Ressourcen-Umgebungs-Modell ein `DataSet` erstellt. Mithilfe des Komponenten-Allokations-Modells kann identifiziert werden, welche Komponente auf welchem Resource-Container ausgeführt wird. Daten die durch eine Komponente geflossen sind, werden in das `DataSet` eingeordnet, das zu dem Resource-Container gehört, auf dem die Komponente ausgeführt wird. Dabei wird aus einem Datum ein `ParameterAndDataPair`. In das `ParameterAndDataPair` wird der Parameter aus dem `Binding` und alle `DataSets`, durch die das Datum geflossen ist, eingetragen.

Zur Veranschaulichung der Transformation, soll Abbildung 5.15 dienen. Die Abbildung stellt den Datenfluss des Aufrufs `BookingSelection.releaseCCD` als Datenflussdiagramm dar. Es wurde die Notation für Datenflussdiagramme aus Abschnitt 3.3 verwendet. Zu Beginn identifiziert die Analyse, dass das Datum *Airline* vom Benutzer aus, zur Komponente *Travelplanner* fließt. Daraufhin wird mithilfe eines neuen `BindingAndComponentPair` festgehalten, dass das `Binding`, des Datums, durch die *TravelPlanner* Komponente fließt. Im Anschluss fließt das Datum weiter zur *CreditCardCenter* Komponente. In das bestehende `BindingAndComponentPair` wird die *CreditCardCenter* Komponente hinzugefügt. Da der Datenfluss nicht weitergeht, wird das `BindingAndComponentPair` analysiert. Die Analyse kommt zu dem Schluss, dass das Datum *Airline* durch die Komponenten *TravelPlanner* und *CreditCardCenter* geflossen ist. Daraufhin identifiziert sie, dass die beiden Komponenten auf dem selben Resource-Container ausgeführt werden. Diese Information wird aus dem Komponenten-Allokations-Modell abgeleitet (hier: Abbildung A.3). Daraufhin erstellt sie ein `DataSet` mit dem Namen des Resource-Containers. In diesem Fall wird ein `DataSet` mit dem Namen *MobilePhone* erstellt. Da das Vertraulichkeitsmodell das Verhalten zwischen Parametern und nicht Daten betrachtet, wird der Parameter *airline*, der mit dem Datum *Airline* verknüpft ist identifiziert. Mithilfe eines `ParameterAndDataPair` aus dem Vertraulichkeitsmodell, wird dieser Parameter mit dem `DataSet` *MobilePhone* verknüpft.

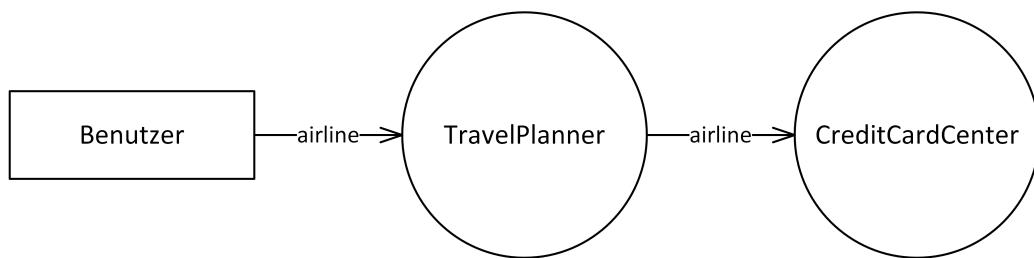


Abbildung 5.15.: Beispiel Datenfluss für die Transformation

Elemente des Ressourcen-Umgebungs-Modells, aus Abschnitt 4.4, werden außerdem in ein vergleichbares Element aus dem Vertraulichkeitsmodell transformiert. Dabei werden aus `ResourceContainerPropertyContainer` und `LinkingResourcePropertyContainer` je-

weils ein `LocationAndTamperProtectionPair`. Die `Location`-Elemente, die in den Containern enthalten sind, werden dabei auf `Location`-Elemente aus dem Vertraulichkeitsmodell abgebildet. Weitere Elemente aus der Erweiterung wurden nicht umgesetzt, da diese im Fallbeispiel nicht vorkommen.

5.3. Ergebnisse

In diesem Abschnitt sollen die Ergebnisse der Validierung vorgestellt werden. Dazu werden die GQM-Fragen aus Tabelle 5.1 beantwortet. Die Antworten zu den Metriken können aus Tabelle 5.2 entnommen werden.

Metrik	Antwort
<i>M1</i>	Alle Daten und Datenflüsse aus Abbildung 5.1 konnten modelliert werden
<i>M2</i>	0 Informationen müssen nochmal modelliert werden
<i>M3</i>	78 neue Elemente mussten erstellt werden
<i>M4</i>	Alle Daten konnten in DataSets eingeordnet werden
<i>M5</i>	Ja, es ist ein valides Modell generiert worden
<i>M6</i>	Keine eindeutige Antwort, da Analyse nicht funktionsfähig

Tabelle 5.2.: Antworten zu den Fragen aus Tabelle 5.1

Dazu wurde das Fallbeispiel aus Abschnitt 5.1 um Daten und Datenflüsse erweitert. Dies wurde in Unterabschnitt 5.1.3 beschrieben. Die Daten und Datenflüsse wurden aus dem Sequenzdiagramm aus Abbildung 5.1 abgeleitet. Die identifizierten Datenflüsse konnten alle modelliert werden (*M1*). Bei der Modellierung konnten die Elemente, der PCM-Modelle aus Unterabschnitt 5.1.1, wiederverwendet werden und mussten nicht neu modelliert werden (*M2*). Bei der Modellierung der Daten und Datenflüsse mussten 78 neue Elemente erstellt werden (*M3*). Von diesen 78 Elementen wurden zwölf für die Hardware-Spezifikation im Ressourcen-Umgebungs-Modell erstellt. 45 Elemente haben die DFSEFFs im Komponenten-Repository-Modell modelliert. Weitere 14 Elemente haben Datenflüsse im Nutzungsmodell modelliert und sechs Elemente haben Daten modelliert. Im Anschluss wurde das Datenflussmodell mit einer Datenflussanalyse analysiert und für die Vertraulichkeitsanalyse transformiert. Die Datenflussanalyse und Transformation wurde in Abschnitt 5.2 beschrieben. Dabei konnten alle modellierten Daten in DataSets eingeordnet werden (*M4*). Das transformierte Modell war valide (*M5*). Das Modell konnte in der Vertraulichkeitsanalyse verwendet werden (*M6*), konnte aber kein Ergebnis liefern. Der Grund, wieso die Analyse kein Ergebnis geliefert hat ist, dass die Vertraulichkeitsanalyse zum Validierungszeitpunkt defekt war. Das Travelplanner-Modell aus der Arbeit von Kramer et. al. [9] konnte auch nicht, mit der Vertraulichkeitsanalyse, analysiert werden.

Da dieser Schritt nicht funktioniert, soll im Folgenden die Modellinstanz, aus der Arbeit von Kramer at. el. [9], mit der transformierten Modellinstanz dieser Bachelorarbeit verglichen werden. Die beiden Modelle enthalten lediglich die Elemente `DataSet`, `Location`, `LocationAndTamperProtectionPair` und `ParameterAndDataPair`. Der erste Unterschied findet sich in den DataSets. Das Vertraulichkeitsmodell enthält die DataSets

5. Validierung

User, Airline und TravelAgency. Das transformierte Modell enthält die DataSets *MobilePhone*, *AgencyServer* und *AirlineServer*. Dieser Unterschied kann jedoch vernachlässigt werden, da die DataSets von der Bedeutung her, die Gleichen sind. Zum Vereinfachen der Beschreibung der Unterschiede werden die DataSets, des transformierten Modells, auf die des Vertraulichkeitsmodells abgebildet. Somit entsteht die Zuordnung: *MobilePhone -> User, AgencyServer -> TravelAgency und AirlineServer -> Airline*. Die Hardware-Spezifikationen für das Ressourcen-Umgebungs-Modell sind in beiden Modellen gleich und werden im weiteren Verlauf der Bachelorarbeit nicht weiter betrachtet. Stattdessen werden die ParameterAndDataPair-Elemente der beiden Modelle verglichen. Dabei ist zu beachten, dass das Vertraulichkeitsmodell nicht an das neue Komponenten-Repository-Modell aus Unterabschnitt 5.1.1 angepasst wurde. Die Unterschiede können aus Tabelle 5.3 entnommen werden. Der erste Unterschied findet sich in dem Dienst *Commision.pay-Commision*. Da dieser Dienst, wie in Unterabschnitt 5.1.1 beschrieben, um einen Rückgabewert und einen Parameter erweitert wurde, findet sich im Vertraulichkeitsmodell keine Zuordnung des Parameters und des Rückgabewerts (1, 2). Im transformierten Modell befindet sich der Parameter *flightOffer* und der Rückgabewert in allen DataSets. Der Grund hierfür ist, dass *flightOffer* durch die Komponenten *TravelPlanner*, *Airline* und *TravelAgency* fließt. Diese Komponenten werden jeweils mit einem der DataSets assoziiert, wie in der Transformation in Abschnitt 5.2 beschrieben. Der nächste Unterschied ist auch auf eine Änderung der Modellierung des Komponenten-Repository-Modells zurückzuführen. Der Dienst *BookingSelection.bookSelected* wurde um einen Parameter *ccdecl* und um einen Rückgabewert erweitert. Dabei ist der Rückgabewert im transformierten Modell in allen DataSets enthalten, da die Daten, bis zum Erstellen des Datums, durch alle Komponenten geflossen sind (5). Der Parameter *ccdecl* ist dabei nur in *User* und *Airline*, da dieser nicht an die *TravelAgency*-Komponente weitergegeben wird (4). Der Parameter *flightOffer* hingegen wird auch an die *TravelAgency* weitergegeben und ist deshalb in allen drei DataSets enthalten (3). Im Vertraulichkeitsmodell gibt es für *ccdecl* und den Rückgabewert keine Angaben. *flightOffer* ist im Vertraulichkeitsmodell nur in *User* und *Airline* enthalten, da im alten Komponenten-Repository-Modell der Parameter nicht an die *TravelAgency* über die Schnittstelle *Commision* weitergegeben wird. D.h. nicht, dass die Analyse vorher falsch war, sondern, dass sich durch die genauere Modellierung der Architektur Änderungen ergeben haben. Bei dem Dienst *Booking.bookFlight* (10,11,12) sind die Gründe, für die Unterschiede, analog zu denen des Dienstes *BookingSelection.bookSelected* (3,4,5). Der Dienst *BookingSelection.getFlightOffers* wurde in der Bachelorarbeit um einen Rückgabewert erweitert. Deshalb kann das Vertraulichkeitsmodell hier keine Angabe zu den DataSets machen (7). Bei dem Parameter *requestData* gibt es in beiden Modellen keinen Unterschied (6). Dieser ist in allen DataSets enthalten, da er durch die Komponenten *Travelplaner*, *TravelAgency* und *Airline* fließt. Deshalb ist auch der Rückgabewert, im transformierten Modell, in allen DataSets enthalten (7). Bei dem Dienst *FlightOffers.getFlightOffers* gibt es keinen Unterschied (8,9). Schließlich wurde auch der Dienst *Declassification.releaseCCD* um einen Rückgabewert erweitert. Das Vertraulichkeitsmodell bietet deshalb hier keine Angabe zu den DataSets an (14). Bei dem Parameter *airline* unterscheiden sich die beiden Modelle nicht (13). In beiden Modellen ist er im DataSet *User* enthalten, da die Daten nur durch die CreditCardCenter Komponente

fließen. Deshalb ist der Rückgabewert im transformierten Modell ebenfalls im **DataSet User** enthalten.

Dieser Vergleich zeigt, dass die Unterschiede im transformierten Modell durchaus berechtigt sind und bei einer Anpassung des Modells, aus der Arbeit von Kramer et. al. [9], zu den gleichen Ergebnissen führen sollten, sofern die Implementierung die Vertraulichkeit wahrt, sowie Implementierung und Spezifikation der Parameter-Data-Set-Zuordnung korrekt ist.

Somit wurden die Ziele aus Tabelle 5.1 erreicht und eine Verhaltensspezifikation auf Datenflussebene in Palladio ermöglicht, die auch für Qualitätsvorhersagen, hier Vertraulichkeit, genutzt werden kann.

(Dienst, Parameter)	V-DS			BA-DS		
	U	A	TA	U	A	TA
(1)(Commision.payCommision, flightOffer)	k.A	k.A	k.A	x	x	x
(2)(Commision.payCommision, return)	k.A	k.A	k.A	x	x	x
(3)(BookingSelection.bookSelected, flightOffer)	x	x		x	x	x
(4)(BookingSelection.bookSelected, ccd_decl)	k.A	k.A	k.A	x	x	x
(5)(BookingSelection.bookSelected, return)	k.A	k.A	k.A	x	x	x
(6)(BookingSelection.getFlightOffers, requestData)	x	x	x	x	x	x
(7)(BookingSelection.getFlightOffers, return)	k.A	k.A	k.A	x	x	x
(8)(FlightOffers.getFlightOffers, requestData)	x	x	x	x	x	x
(9)(FlightOffers.getFlightOffers, return)	x	x	x	x	x	x
(10)(Booking.bookFlight, fligthOffer)	x	x		x	x	x
(11)(Booking.bookFlight, cc_decl)	k.A	k.A	k.A	x	x	
(12)(Booking.bookFlight, return)	x	x	x	x	x	x
(13)(Declassification.releaseCCD, airline)	x			x		
(14)(Declassification.releaseCCD, return)	k.A	k.A	k.A	x		

V-DS: DataSet aus dem Vertraulichkeitsmodell, BA-DS: DataSet aus dem Modell der Bachelorarbeit, U: User, A: Airline, TA: TravelAgency

Tabelle 5.3.: Unterschiede der Zuordnung von Parameter in DataSets

6. Zusammenfassung und Ausblick

In dieser Bachelorarbeit wurde eine Datenflussdokumentation auf Architekturebene vorgestellt. Mit dieser Datenflussdokumentation ist es möglich Modelle des PCM um Daten und Datenflüsse zu erweitern. Die Elemente, mit denen das möglich ist, sind in Kapitel 4 beschrieben. In Kapitel 5 wurde dann mithilfe eines Fallbeispiels gezeigt, dass mit der Modellierung Daten und Datenflüsse modelliert werden können. Dabei ist eine Modellierung von Daten und Datenflüssen, ausgehend vom Benutzer des Systems, zu den Komponenten und innerhalb der Komponenten möglich. Außerdem wurde gezeigt, dass die Modellierung in Datenflussanalysen verwendet werden kann. Dazu wurden die Datenflüsse innerhalb des Modells analysiert und transformiert. Das transformierte Modell wurde verwendet um die Eingabe und Ausgabe des Fallbeispiels zu spezifizieren. Anschließend wurde versucht mithilfe einer Vertraulichkeitsanalyse nach Kramer et. al. [9] diese Modellierung zu überprüfen. Leider konnte diese, aufgrund eines Defekts, nicht vollständig ausgeführt werden. Stattdessen wurden das Vertraulichkeitsmodell und das transformierte Modell verglichen. Die Unterschiede wurden diskutiert und zu dem Ergebnis geführt, dass die Modellierung dieser Bachelorarbeit durchaus für Datenflussanalysen geeignet ist.

Die Datenflussdokumentation, die in dieser Bachelorarbeit entstanden ist, dient dabei die einzelnen Rollen und die dazugehörigen Modelle, des PCM, zu erweitern. Sie ermöglicht dem Komponentenentwickler das Verhalten innerhalb von Komponenten um einen Datenfluss zu erweitern. Somit wird das Komponenten-Repository-Modell erweitert. Auch dem Domänenexperten wird es ermöglicht Datenflüsse zu modellieren. Die Datenflüsse gehen vom Benutzer aus und erweitern das Nutzungsmodell. Schließlich wird dem Software-Verteilungsexperten ermöglicht eine Hardware-Spezifikation im Ressourcen-Umgebungs-Modell durchzuführen.

Für zukünftige Arbeiten könnte die Modellierung dieser Bachelorarbeit mit weiteren Fallbeispielen validiert werden. Indem weitere Fallbeispiele erstellt oder existierende aus der Literatur mit Datenflüssen erweitert werden, könnte die Modellierung genauer validiert werden. Als Beispiel würde sich z.B. das zweite Fallbeispiel aus der Arbeit von Kramer et. al. [9] eignen, da dieses bereits über modellierte PCM-Modelle verfügt. Außerdem könnten weitere Datenflussanalysen betrachtet werden. Dabei könnten andere Sicherheitseigenschaften überprüft werden. Die Datenflussanalyse aus der Arbeit **UMLsec** [6] könnte sich dazu eignen. Dafür müsste die dortige Modellierung und Analyse betrachtet werden und im Anschluss eine Transformation geschrieben werden, die ein Modell dieser Bachelorarbeit in ein **UMLsec**-Modell transformiert. Dazu könnten die Fallbeispiele von **UMLsec** mithilfe des PCM nachmodelliert, mit Daten und Datenflüssen erweitert und die jeweiligen Ergebnisse verglichen werden. Um die Modellierung besser in Palladio-Workbench zu integrieren, könnte ein graphischer Editor erstellt werden, mit dem Daten und Datenflüsse erstellt werden können. Damit könnte die Akzeptanz bei den Architekten geprüft werden. Die Masterarbeit **Flexible Graphical Editors for Extensible Modular**

6. Zusammenfassung und Ausblick

Meta Models von Michael Junker, setzt sich mit dem Thema auseinander, wie sich erweiterbare Meta-Modelle auf flexible graphische Editoren auswirken. Dabei wurde auch die Erweiterung dieser Bachelorarbeit betrachtet. Außerdem wird in einer weiteren Masterarbeit von Philipp Weimann diese Bachelorarbeit auch betrachtet. In der Masterarbeit werden Applikationen untersucht, die verteilt in der Cloud liegen. Auslagerungen sollen dabei automatisch erkannt werden und bei Verstoß gegen vorgegebene Richtlinien eine alternative Verteilung des Systems ermittelt und angewendet werden.

Literatur

- [1] Victor Basili. "A Methodology for Collecting Valid Software Engineering Data". In: *IEEE Transactions on Software Engineering* SE-10.6 (1984), S. 728–738.
- [2] Steffen Becker. "The palladio component model". In: *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering - WOSP/SIPEW '10* March (2010), S. 257. URL: <http://portal.acm.org/citation.cfm?doid=1712605.1712651>.
- [3] Tom DeMarco. *Structured Analysis and System Specification*. Prentice Hall. Prentice Hall PTR, 1979.
- [4] Claudia Eckert. *IT-Sicherheit: Konzepte-Verfahren-Protokolle*. Oldenbourg Wissenschaftsverlag, 2013.
- [5] Ronald Eikenberg. *Hacker steuern Jeep Cherokee fern*. <http://www.heise.de/security/meldung/Hacker-steuern-Jeep-Cherokee-fern-2756331.html>. [Online; abgerufen am 27.09.2016].
- [6] Jan Jürjens. *Secure Systems Development with UML*. Springer-Verlag Berlin Heidelberg, 2005.
- [7] Torsten Kleinz. *Der Feind hört mit: IP-Telefone von Snom aus dem Netz angreifbar*. <https://www.heise.de/security/meldung/Der-Feind-hoert-mit-IP-Telefone-von-Snom-aus-dem-Netz-angreifbar-2517201.html>. [Online; abgerufen am 27.09.2016].
- [8] Heiko Koziolek und Jens Happe. "A QoS Driven Development Process Model for Component-Based Software Systems". In: *Proceedings of the 9th International Symposium on Component Based Software Engineering (CBSE 2006)* (2006), S. 336–343. URL: http://dx.doi.org/10.1007/11783565%7B%5C_7D25.
- [9] Max E Kramer u. a. *Model-Driven Specification and Analysis of Confidentiality in Component-Based Systems*.
- [10] Max E. Kramer u. a. *Specification and Verification of Confidentiality in Component-Based Systems*. Poster at the 35th IEEE Symposium on Security and Privacy. San Jose, California, USA, 2014. URL: <http://www.ieee-security.org/TC/SP2014/posters/KRAME.pdf>.
- [11] Max Kramer u. a. "Extending the Palladio Component Model using Profiles and Stereotypes". In: *Proceedings of the Palladio Days 2012* (2012).
- [12] Philip Langer u. a. "EMF Profiles: A Lightweight Extension Approach for EMF Models." In: *Journal of Object Technology* 11.1 (2012), S. 1–29. URL: <http://dblp.uni-trier.de/db/journals/jot/jot11.html#LangerWWC12>.

- [13] Philipp Merkle. *Palladio.TX*. <https://sdqweb.ipd.kit.edu/wiki/Palladio.TX>. [Online; abgerufen am 27.09.2016].
- [14] PASE. <https://sdqweb.ipd.kit.edu/wiki/PASE>. [Online; abgerufen am 27.09.2016].
- [15] Ponemon Institute. “2015 Cost of Data Breach Study: Global Analysis”. In: May (2015). URL: <https://nhlearningsolutions.com/Portals/0/Documents/2015-Cost-of-Data-Breach-Study.PDF>.
- [16] Ralf Reussner u. a. *Modeling and Simulating Software Architectures - The Palladio Approach*.
- [17] Herbert Stachowiak. *Allgemeine Modelltheorie*. 1973, S. 131–133.
- [18] Czarnecki Krzysztof Stahl Thomas, Voelter Markus. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006. ISBN: 0470025700.
- [19] Thomas Stahl u. a. *Modellgetriebene Softwareentwicklung Techniken, Engineering, Management*. 2007.
- [20] Kurt Stenzel u. a. “A model-driven approach to noninterference”. In: *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications* 5.3 (2014), S. 30–43.
- [21] Misha Strittmatter u. a. “A Modular Reference Structure for Component-based Architecture Description Languages”. In: *2nd International Workshop on Model-Driven Engineering for Component-Based Systems (ModComp)*. CEUR, 2015, S. 36–41. URL: <http://ceur-ws.org/Vol-1463/paper6.pdf>.

A. Anhang

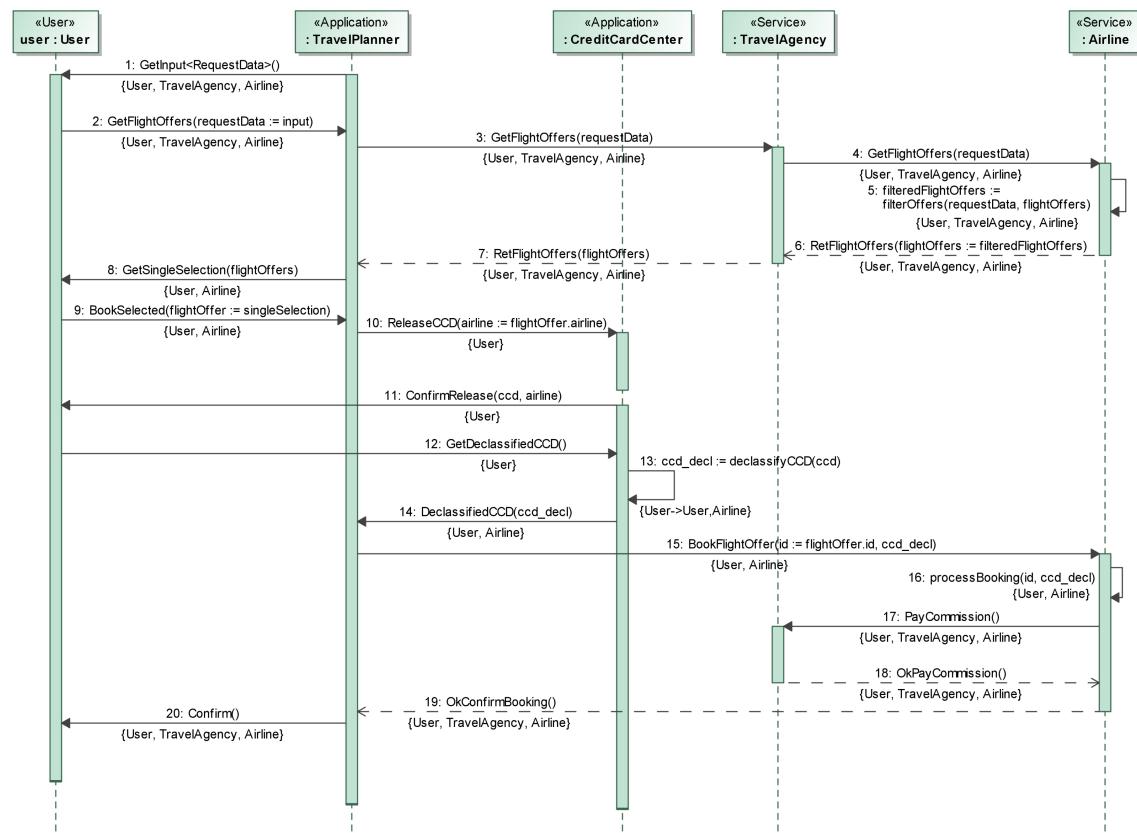


Abbildung A.1.: Sequenzdiagramm zur Buchung eines Fluges nach [20].

A. Anhang

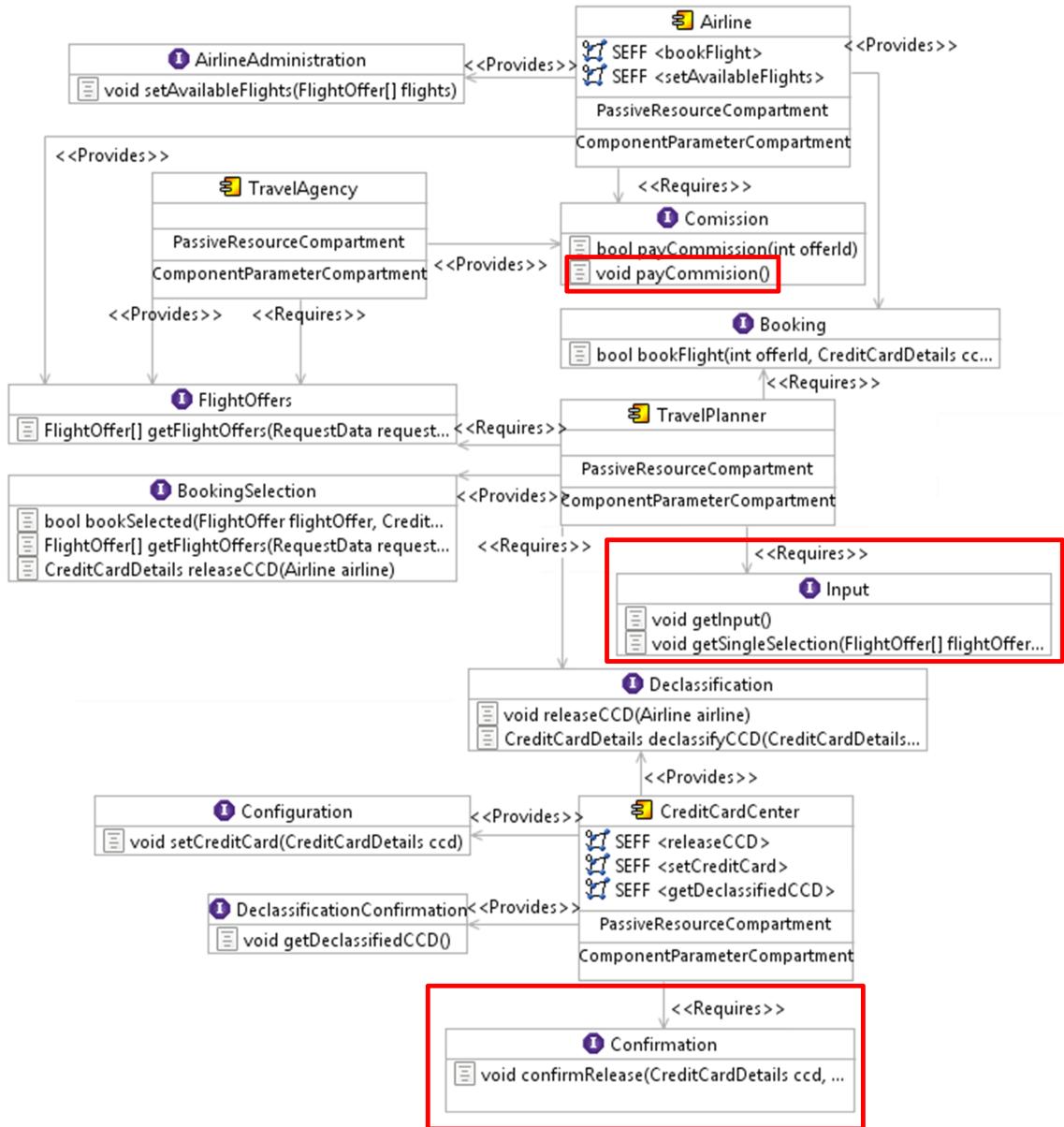


Abbildung A.2.: Komponenten-Repository-Modell des Travelplanner nach [9]. Markierte Elemente wurden entfernt

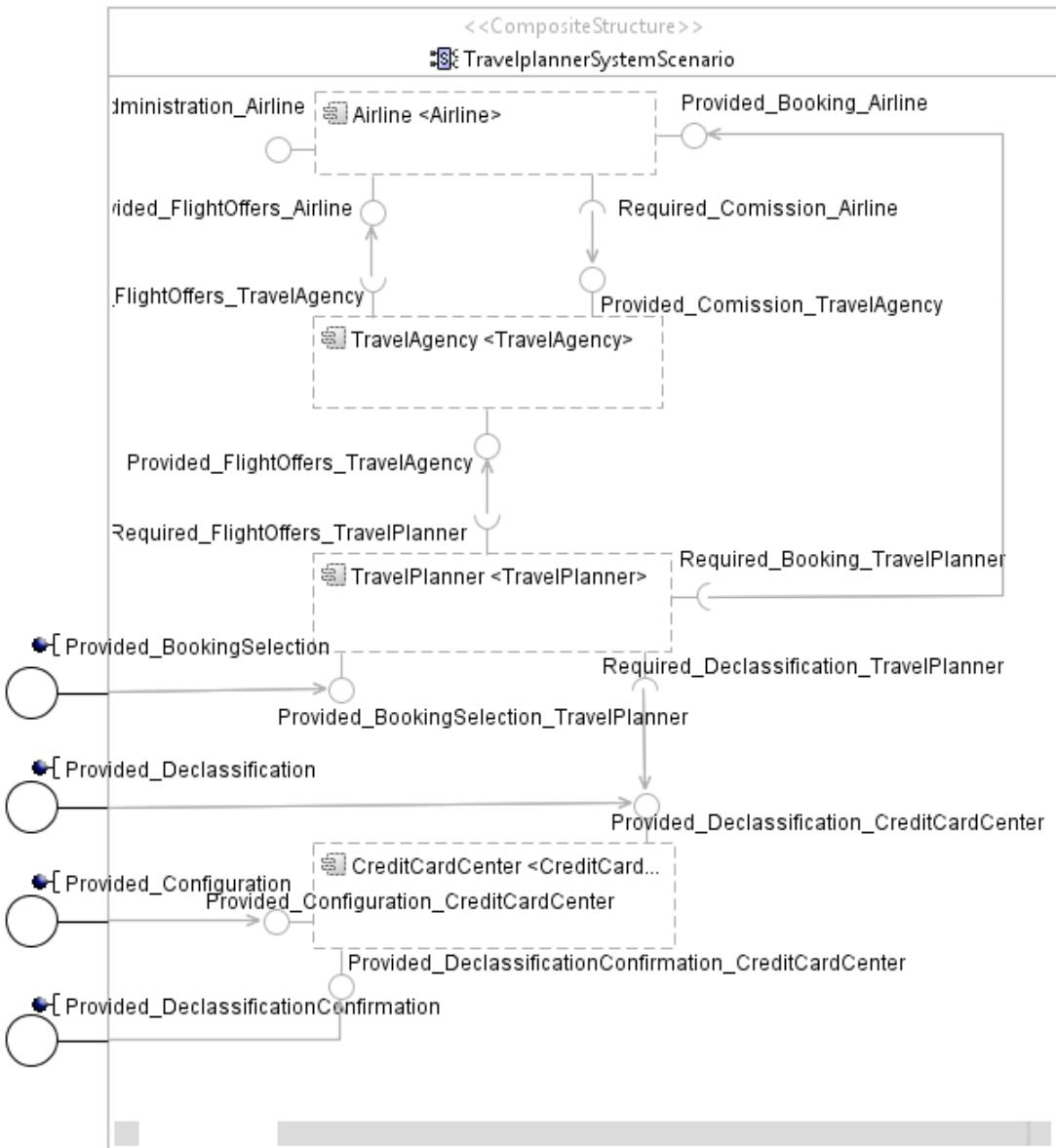


Abbildung A.3.: Systemmodell des Travelplanner

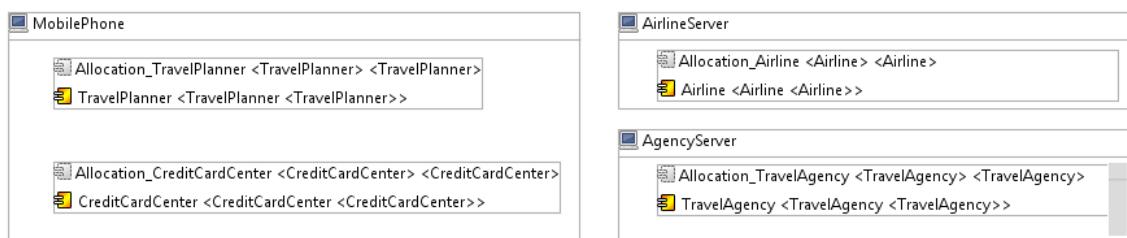


Abbildung A.4.: Komponenten-Allokations-Modell des Travelplanner