Abbildung von UMLsec-Vertraulichkeitsanalysen auf Data-Centric Palladio

Masterarbeit von

Philip Müller

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

Erstgutachter:	Prof. Dr. Ralf H. Reussner
Zweitgutachter:	Jun.-Prof. Dr.-Ing. Anne Koziolek
Betreuender Mitarbeiter:	M. Sc. Stephan Seifermann

11.05.2018 – 09.04.2019

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, 08.04.2019

. . . . . .

(Philip Müller)

Zusammenfassung

Die Vertraulichkeit von Daten ist heutzutage ein wichtiger Aspekt von Informationssystemen. Ein unzureichender Schutz dieser Daten kann hohe Bußgelder oder den Verlust von Kunden zur Folge haben.

Die sichere Behandlung von sensitiven Daten stellt damit eine zentrale Qualitätseigenschaft eines Softwaresystems dar und muss bereits während des Architekturentwurfs beachtet werden, um teure Änderungen spät im Entwicklungsprozess zu vermeiden.

Ein Ansatz zur Modellierung von Sicherheitseigenschaften auf der Architekturebene ist Data-Centric Palladio (DCP). DCP reichert das Architekturmodell um ein Datenflussmodell an, um Analysen der Vertraulichkeitseigenschaften von Daten zu ermöglichen. Es ist jedoch unklar, ob der in DCP gewählte Ansatz bezüglich der Ausdrucksmächtigkeit äquivalent zu etablierten Ansätzen der Modellierung und Analyse von Sicherheitseigenschaften ist.

Daher wird in dieser Arbeit die Ausdrucksmächtigkeit von DCP mit der Ausdrucksmächtigkeit von UMLsec verglichen. Dazu werden UMLsec-Analysen anhand von definierten Auswahlkriterien auf ihre Eignung untersucht, in DCP umgesetzt zu werden. Für Analysen, die sich für eine Umsetzung in DCP eignen, werden äquivalente Analysen in DCP definiert und implementiert. Zudem werden Modelltransformationen zwischen den Eingabemodellen der UMLsec-Analyse und der DCP-Analyse spezifiziert, mit derer Hilfe die Evaluation der erstellten DCP-Analyse erfolgt.

Die UMLsec-Analysen Secure Links und Secure Dependencies sind in dieser Arbeit als DCP-Analysen umgesetzt. Es ist zu erkennen, dass der datenflussbasierte Ansatz von DCP dazu führt, dass in den DCP-Varianten dieser beiden Analysen wesentlich detailliertere Analyseergebnisse ermittelt werden können. Dagegen ist für die kontrollflussbasierte UMLsec-Analyse Fair Exchange keine Umsetzung in DCP möglich, da der Analyseansatz von DCP nicht in der Lage ist, einen Angreifer zu modellieren, der aktiv den Kontrollfluss des Systems modifiziert. Schließlich ist an der UMLsec-Analyse des sicheren Informationsflusses gezeigt, dass DCP zwar in der Lage ist, eine semantisch ähnliche Analyse auf einer höheren Abstraktionsebene umzusetzen, der direkte Vergleich zu der UMLsec-Analyse allerdings aufgrund des hohen Abstraktionsunterschieds dieser Analysen nicht möglich ist.

Für den Vergleich der Ausdrucksmächtigkeit der beiden Ansätze ist schließlich gezeigt, dass weder DCP, noch UMLsec strikt mächtiger ist, als der jeweils andere Ansatz. Stattdessen ist der erwartete Anwendungsfall von erheblicher Bedeutung für die Auswahl des geeigneten Analyseansatzes.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmusverzeichnis

1. Einleitung

In der heutigen Gesellschaft ist die Vertraulichkeit von Daten von zunehmender Wichtigkeit. Zentrale Aspekte der Ökonomie, wie z. B. Kommunikations-, Transport- oder Finanzdienste, und des Gesundheitswesen, wie z. B. die elektronische Gesundheitsakte, basieren auf vernetzten Informationssystemen, welche über das Internet verbunden sind. Da das Internet auch nicht-vertrauenswürdigen Parteien zugänglich ist, existiert jedoch eine große Angriffsfläche und die Möglichkeit, aus beliebiger Entfernung Angriffe auf die Systeme auszuführen. Sensitive Daten, wie z. B. Kreditkarteninformationen, müssen daher sicher aufbewahrt und übertragen werden. [1, Seite 3]

Dies wird von der aktuellen Rechtslage reflektiert. Es existieren strikte Anforderungen an Informationssysteme, die personenbezogene Daten erheben und verarbeiten. Seitdem die Datenschutzgrundverordnung in Kraft getreten ist, hat das Missachten dieser Anforderungen hohe Bußgelder zur Folge. Beispielweise musste das soziale Netzwerk Knuddels ein Bußgeld in Höhe von 20.000 Euro zahlen, weil Nutzerdaten und Klartextpasswörter von Hackern erbeutet werden konnten und anschließend veröffentlicht wurden [2]. Aber auch außerhalb der EU können Verletzungen der Vertraulichkeit von personenbezogenen Daten rechtliche Konsequenzen haben. So wurde z. B. die Firma LinkedIn verklagt, weil nicht ausreichende Maßnahmen zum Schutz von Nutzerdaten unternommen wurden [3]. Ein unangemessene Handhabung von personenbezogenen Daten kann eine Firma auch ungeachtet der rechtlichen Konsequenzen schaden. Beispielsweise hat Facebook als Konsequenz des Cambridge-Analytica-Skandals [4] einen erheblichen Verlust des Vertrauens der Nutzer hinnehmen müssen [5].

Die sichere Behandlung von sensitiven Daten und die Einhaltung von Rechtsnormen sorgt für eine erhöhte Komplexität von Softwaresystemen. Aufgrund dieser Komplexität ist die Entwicklung von sicheren Systemen schwierig und fehleranfällig [1, Kapitel 1] und die Erfassung von Sicherheitseigenschaften eines Systems wird häufig auf spätere Phasen des Entwicklungsprozesses verschoben [6]. In diesen Phasen ist die Fehlerbehebung von Sicherheitsproblemen jedoch teuer und manchmal gar nicht möglich. Ferner handelt es sich bei Sicherheitsproblemen zu 50% um Schwachstellen des Entwurfs [7, Seite 139]. Diese Schwachstellen können nicht durch eine Inspektion des Programmcodes gefunden werden. Es ist daher effizienter, kostengünstiger und teilweise notwendig, die für die Anwendung notwendigen Sicherheitseigenschaften bereits im Architekturentwurf zu erfassen und sie in allen Phasen des Entwicklungsprozesses zu beachten [8].

Eine Möglichkeit, um Sicherheitseigenschaften bereits während des Architekturentwurfs zu erfassen, ist die Modellierung des Softwaresystems unter der Verwendung eines Ansatzes zur Spezifikation und Analyse von Sicherheitseigenschaften des modellierten Systems.

Dabei ist das generelle Vorgehen zunächst ein Modell des Systems zu konstruieren, das so abstrakt wie möglich ist, ohne dabei an Intuitivität einzubüßen. Anschließend wird anhand des Modells eine Implementierung abgeleitet. Dies kann automatisiert durch Code-Generation geschehen oder manuell, wobei dann zumindest Testsequenzen aus dem Modell generiert werden, sodass die Implementierung auf Konformität zum Modell überprüft werden kann. Im Gegensatz zu formalen Beweismethoden, welche in der Industrie nur selten Anwendung finden [9, 10], benötigt der modellbasierte Ansatz wenig zusätzliches Training der Entwickler. [1, Kapitel 1]

Ein solcher Ansatz ist Data-Centric Palladio (DCP) [6, 11, 12]. Es handelt sich dabei um eine auf Datenflüssen basierende Erweiterung des Palladio Component Model (PCM) [13]. Das PCM ist eine Architektur-Beschreibungssprache zur Modellierung von Informationssystemen [12]. Die Erweiterung mit DCP ermöglicht es, Datenflüsse explizit zu modellieren und Datenklassen mit Vertraulichkeitseigenschaften zu annotieren. Unter der Angabe eines Sicherheitsziels und des erweiterten Modells kann die Einhaltung des Sicherheitsziels analysiert werden. Weil diese Analyse auf dem erweiterten Architekturmodell operiert, kann sie bereits früh im Entwicklungsprozess eingesetzt werden und dadurch potentielle Sicherheitsprobleme frühzeitig aufdecken.

Es ist jedoch unklar, ob der Ansatz von DCP bezüglich der Ausdrucksmächtigkeit äquivalent zu etablierten Ansätzen ist. Insbesondere ist dies deshalb relevant, da die Festlegung auf einen Modellierungsansatz bereits sehr früh im Entwicklungsprozess geschieht. Sollte im späteren Verlauf des Entwicklungsprozesses eine Sicherheitsanalyse nötig werden, die von DCP nicht unterstützt wird, kann der Wechsel zu einem anderen Modellierungsansatz, der diese Analyse unterstützt, hohe Kosten verursachen.

Ziel dieser Arbeit ist daher die Untersuchung der Ausdrucksmächtigkeit von DCP in Bezug auf einen etablierten Ansatz zur Modellierung von Sicherheitseigenschaften. Der dafür ausgewählte Ansatz ist UMLsec [1]. Dabei handelt es sich um eine Erweiterung der Modellierungssprache UML, die es ermöglicht, Sicherheitseigenschaften abzubilden und verschiedene Analysen auf diesen erweiterten Modellen durchzuführen. UMLsec eignet sich deshalb für diese Betrachtung, da es bereits in mehreren Fallstudien [14, 15, 16, 17] eingesetzt und validiert ist und die domänenunabhängige Modellierung von Sicherheitseigenschaften erlaubt, anstatt auf eine bestimmte Domäne spezialisiert zu sein.

Um Erkenntnisse über die Ausdrucksmächtigkeit von DCP in Bezug auf UMLsec zu gewinnen, werden verschiedene UMLsec-Analysen betrachtet und auf ihre Umsetzbarkeit in DCP untersucht. Sollten sie sich für eine Umsetzung in DCP eignen, wird eine äquivalente Analyse in DCP definiert, implementiert und eine Modelltransformation spezifiziert, die Eingabemodelle der UMLsec-Analyse (bzw. DCP-Analyse) auf Eingabemodelle der DCP-Analyse (bzw. UMLsec-Analyse) abbildet. Durch diese Transformationen kann evaluiert werden, ob die in DCP definierte Analyse äquivalent zu der UMLsec-Analyse ist und ob Szenarien existieren, in denen die Ergebnisse der beiden Analysen voneinander abweichen, wodurch potentiell Rückschlüsse auf ihre zueinander relative Ausdrucksmächtigkeit gezogen werden können. So wird festgestellt ob Analysen existieren, die sich nicht oder nur

eingeschränkt in DCP umsetzen lassen oder in denen die in DCP definierte Analyse eine höhere Ausdrucksmächtigkeit aufweist, als die korrespondierende UMLsec-Analyse.

Aufgrund der Menge von UMLsec-Analysen kann in dieser Arbeit keine vollständige Betrachtung für jede einzelne UMLsec-Analyse erfolgen. Daher ist zu erwarten, dass keine abschließende Aussage über die Ausdrucksmächtigkeit von DCP in Bezug auf UMLsec getroffen werden kann. Stattdessen sollen die im Rahmen dieser Arbeit hervorgebrachten DCP-Analysen und Erkenntnisse über die Ausdrucksmächtigkeit als eine Grundlage für weitere Betrachtungen dienen und die Vorgehensweise der Spezifikation der Modelltransformation als Schema für weitere Transformationen fungieren.

Die Arbeit ist folgendermaßen strukturiert. In Kapitel 2 werden die Grundlagen eingeführt. Dazu gehören insbesondere die für das Verständnis der UMLsec- und DCP-Analyse notwendigen Informationen. Nachfolgend werden in Kapitel 3 verwandte Arbeiten betrachtet. Kapitel 4 stellt das iterative Vorgehen dar, das für die Auswahl und Umsetzung einer Analyse angewandt wurde. In Kapitel 5 und Kapitel 6 werden jeweils die UMLsec-Analysen Secure Links und Secure Dependencies und die dazu definierten DCP-Analysen und spezifizierte Modelltransformationen vorgestellt. In diesen Kapiteln werden auch Erkenntnisse vorgestellt, die bei der Erstellung der jeweiligen Analyse und Transformationen gewonnen wurden. Auf der Basis dieser Erkenntnisse erfolgt in Kapitel 7 eine Betrachtung der Ausdrucksmächtigkeit von DCP in Relation zu UMLsec. In diesem Kapitel werden auch die Analysen vorgestellt, die als Teil dieser Arbeit betrachtet wurden, aber sich nicht für die Umsetzung in DCP eignen. Schließlich werden in Kapitel 8 die umgesetzten Analysen und spezifizierten Modelltransformationen evaluiert. Abschließend erfolgt in Kapitel 9 eine Zusammenfassung und ein Ausblick auf weiterführende Forschungsfragen.

2. Grundlagen

Dieses Kapitel gibt eine Übersicht über die relevanten Konzepte, die zum Verständnis dieser Arbeit notwendig sind. Dabei werden zunächst die Modellierung von Softwaresystemen und die für diese Arbeit relevanten Modellierungssprachen betrachtet. Anschließend folgt eine Erklärung von Data-Centric Palladio und UMLsec, den untersuchten Ansätzen zur Analyse von Sicherheitseigenschaften.

2.1. Modellierung von Softwaresystemen

Modelle werden in der Softwaretechnik dazu genutzt, eine abstrakte Darstellung eines Softwaresystems zu erhalten. Dabei können verschiedene Modelle unterschiedliche Sichten oder Perspektiven auf das zu modellierende System bieten. Dies dient häufig als gemeinsame Basis für die Diskussion des existierenden oder vorgeschlagenen Systems, als Dokumentation eines existierenden Systems oder als detaillierte Systembeschreibung, aus der eine Implementation generiert werden kann. [18, Seiten 118–121]

Ein Beispiel für eine Modellierungssprache ist die Unified Modeling Language (UML) [19]. Das UML-Metamodell definiert eine Vielzahl von Modellelementen und Beziehungen zwischen diesen, sodass mit UML unterschiedliche Sichten auf ein Softwaresystem modelliert werden können. Die Darstellung von UML-Modellen geschieht üblicherweise grafisch in Form von Diagrammen. Generell wird dabei zwischen Struktur- und Verhaltensdiagramme unterschieden.

Ein anderer Ansatz bezüglich der Modellierung und Analyse von Softwaresystemen ist der Palladio-Ansatz [13]. Der Fokus liegt auf der Vorhersage von Qualitätseigenschaften, wie z. B. Performanz oder Zuverlässigkeit. Dadurch ist es möglich, frühzeitig zu entscheiden, ob die Softwarearchitektur den gewünschten Qualitätsansprüchen genügt, um somit teure Änderungen spät im Entwicklungsprozess zu vermeiden.

Der Palladio-Ansatz besteht aus dem Palladio-Komponentenmodell (Palladio Component Model, PCM), modellgetriebenen Analysen und dem Palladio-Entwicklungsprozess. Das PCM ist eine domänenspezifische Sprache und wird verwendet, um eine Softwarearchitektur zu spezifizieren und zu dokumentieren. Durch das PCM beschriebene Softwarearchitekturen bestehen aus fünf separaten Modellen. Das Komponentenverzeichnis (repository) beschreibt Datentypen, Komponenten und die Schnittstellen der Komponenten. Das Systemmodell beschreibt, wie Komponenteninstanzen (assembly contexts) miteinander verbunden werden, um ein System zu erzeugen. Während die Ausführungsumgebung

(resource environment) die zur Verfügung stehenden Rechenressourcen und die diese verbindenden Netzwerkressourcen (linking resources) beschreibt, stellt das Verteilungsmodell (allocation model) die Allokation von Komponenteninstanzen auf Rechenressourcen der Ausführungsumgebung dar. Schließlich beschreibt das Nutzungsmodell (usage model) das Nutzerverhalten im Umgang mit dem System.

Die Analysen finden bei Palladio nicht direkt auf dem in PCM spezifizierten Architekturmodell statt, sondern auf analytischen Modellen, die aus dem Architekturmodell abgeleitet werden. Dies erlaubt es, analysespezifische Details getrennt vom Architekturmodell zu definieren und ermöglicht somit, dass kein Expertenwissen über die Analyse notwendig ist, um das Architekturmodell zu verstehen. Die Generation der Analysemodelle und die Analyse selbst können automatisiert erfolgen.

Schließlich definiert Palladio verschiedene Rollen für den Entwicklungsprozess. Komponentenentwickler sind dafür verantwortlich, Komponenten zu spezifizieren und zu implementieren, während Softwarearchitekten wiederverwendbare Architekturen entwerfen. Die Modellierung der Ausführungsumgebung wird vom Systementwickler (system deployer) umgesetzt und der Domänenexperte modelliert das übliche Nutzerverhalten und ist verantwortlich für die Identifizierung relevanter Anwendungsfälle.

Eine weitere Möglichkeit ein Softwaresystem zu modellieren ist die Datenflussmodellierung. Dabei wird von den genauen Kontrollflüssen und Strukturen abstrahiert und lediglich der Informationsfluss durch das System untersucht. Insbesondere für parallele Systeme kann dies eine das Verständnis fördernde Abstraktion sein. Datenflüsse werden in der Regel durch Datenflussgraphen dargestellt. Ein Datenflussgraph ist ein bipartiter Graph, dessen zwei Knotentypen Akteure (actor) und Verbindungen (link) genannt werden [20, 21]. Akteure beschreiben Datenoperationen, während Verbindungen Daten von einem Akteur empfangen und an einen oder mehrere Akteure übermitteln [21]. Ein Akteur führt seine Operation durch, wenn alle für diese Operation notwendigen Operanden zur Verfügung stehen. Im Gegensatz zu sequentiellen Kontrollflüssen kann bei Datenflüssen also ein höchst paralleler Ablauf verschiedener Operationen vorliegen. Abschließend ist zu erwähnen, dass Ansätze existieren, mit denen in Datenflussgraphen auch Kontrollflüsse modelliert werden können [22].

2.2. Data-Centric Palladio

Data-Centric Palladio [6, 11, 12] ist eine Erweiterung der Architekturbeschreibungssprache PCM mit dem Zweck, Datenflüsse in Softwarearchitekturen zu modellieren, um dadurch die Analyse von Sicherheitseigenschaften, wie z. B. Vertraulichkeit, zu ermöglichen. Der Ausgangspunkt dieses Ansatzes stellt das in PCM definierte Architekturmodell dar. Das um Datenflüsse angereicherte Architekturmodell wird im Folgenden als Eingabemodell bezeichnet, da es im Kontext dieser Arbeit hauptsächlich als Eingabe für die zu betrachtenden Sicherheitsanalysen zu verstehen ist. Um die Sicherheitsanalyse durchzuführen wird das Eingabemodell zunächst in ein vereinfachtes Zwischenmodell, das sogenannte

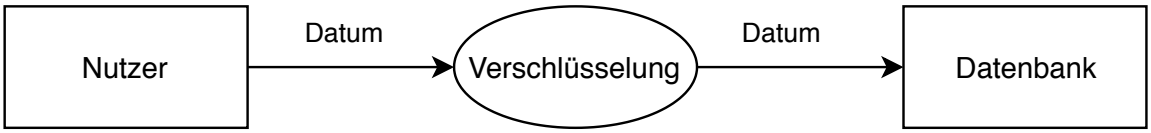

Abbildung 2.1.: Ein Beispiel für einen Datenfluss.

Operationsmodell, transformiert. Das Operationsmodell enthält nur die für die Analyse notwendigen Informationen. Anschließend wird aus dem Operationsmodell durch eine Modell-zu-Text-Transformation ein Prolog-Programm erzeugt, welches eine textuelle Repräsentation des Operationsmodells darstellt. Dies erlaubt es, den Analysealgorithmus als deklarative Abfrage an eine bestehende Prolog-Schnittstelle zu definieren, anstatt einen prozeduralen Algorithmus programmieren zu müssen, der das Modell direkt analysiert.

Auf der Abstraktionsebene des Eingabemodells werden Datenflüsse als Datenflussgraphen nach Yourdon und Constantine [23, Seite 43–46] dargestellt. Dabei wird zwischen Datenquellen (data sources), -senken (data sinks), -übertragungen (data transmissions) und -verarbeitungsvorgängen (processing operations) unterschieden. Eine Datenquelle erzeugt Daten und ist somit Startpunkt des Datenflusses. Eine Datensenke erhält Daten und terminiert den Datenfluss. Dagegen erhält ein Datenverarbeitungsvorgang ein eingehendes Datum und erzeugt ein neues, ausgehendes Datum. Zwischen Quellen, Senken und Verarbeitungsvorgängen beschreiben Datenübertragungen die Verbindungen und sind in dem Graph als Kanten dargestellt. Sie sind mit der Datenklasse, die sie übertragen, annotiert. Ein Beispiel für einen einfachen Datenfluss ist in Abbildung 2.1 dargestellt. Ein *Nutzer* erzeugt Daten der Datenklasse *Datum*, welche über den Verarbeitungsvorgang *Verschlüsselung* zu der Senke *Datenbank* fließen und dort gespeichert werden.

Daten oder Ressourcen können mit Charakteristiken annotiert werden. Charakteristiken sind eine endliche Menge von Werten, die dazu dienen die für die Sicherheitsanalysen notwendigen zusätzlichen Informationen bereitzustellen. Datenverarbeitungsvorgänge können dazu verwendet werden, aus den Charakteristiken eines eingehenden Datums automatisch die Charakteristiken des ausgegebenen Datums abzuleiten. In manchen Fällen ist es ausreichend, die Charakteristiken des eingehenden Datums auf das ausgegebene Datum zu kopieren. Um jedoch komplexe Datenflüsse zu modellieren ist es häufig notwendig, die Charakteristiken aufgrund des Datenverarbeitungsvorganges zu ändern.

Ein Beispiel dafür ist in Abbildung 2.2 dargestellt. Hierbei sind die Daten um zwei Charakteristiken erweitert. Der Charakteristiktyp *Sensitivität* beschreibt den Geheimhaltungsbedarf der Daten und kann die Charakteristiken *Geheim* und *Verschlüsselt* annehmen. Die vom Nutzer ausgehenden Daten haben initial die Charakteristik *Geheim*. Die Daten, die von dem Datenverarbeitungsvorgang *Verschlüsselung* ausgegeben werden, haben keine explizite Annotation von Charakteristiken. Stattdessen definiert dieser Verarbeitungsvorgang eine Abbildung der Charakteristiken des eingehenden Datums auf die Charakteristiken des ausgehenden Datums. Der Verarbeitungsvorgang weist dem Charakteristiktyp *Sensitivität* des ausgehenden Datums die Charakteristik *Verschlüsselt* zu. Analog ist der Ort aus dem die Daten ursprünglich stammen als Charakteristiktyp modelliert. Diese Charakteristik

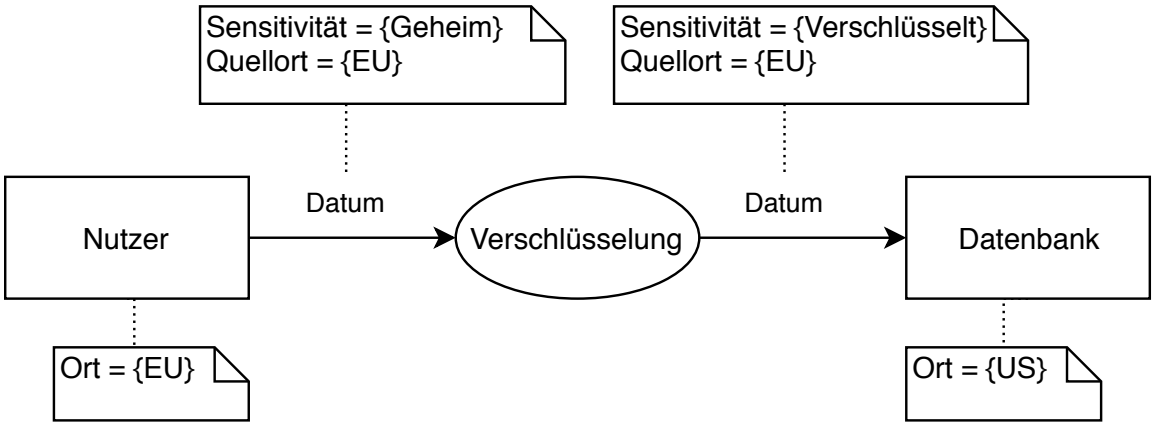

Abbildung 2.2.: Der Datenfluss aus Abbildung 2.1 annotiert um Charakteristiken.

wird von der *Verschlüsselung* unverändert vom Eingabedatum auf das Ausgabedatum kopiert.

Charakteristiken können ebenfalls an andere Modellelemente annotiert werden. In diesem Beispiel ist der Standort der Rechnerknoten (resource containers) als Charakteristiktyp *Ort* modelliert. Der *Nutzer* hat die Charakteristik *EU* und die Datenbank ist mit *US* annotiert. So kann eine Analyse z. B. überprüfen, ob sensitive Daten unverschlüsselt außerhalb der EU gespeichert werden. Um die automatische Analyse zu gewährleisten, handelt es sich bei allen in dieser Arbeit betrachteten Charakteristiktypen um Enumerationen anstelle von z. B. nicht typsicheren Strings.

Das Operationsmodell [12, 24] repräsentiert die Datenflüsse und ihre Effekte auf die Charakteristiken der Daten. Es dient zur Entkopplung des Analyseverfahrens von der Architekturbeschreibungssprache. Eine vereinfachte Form des Metamodells des Operationsmodells ist in Abbildung 2.3 dargestellt. Elemente der Klasse *Systemnutzung* (SystemUsage) sind der Startpunkt einer Aufrufsequenz. Eine Aufrufsequenz besteht dabei aus Operationen und Operationsaufrufen. Ein *Operationsaufruf* (OperationCall) hat immer eine *Operation* als Ziel des Aufrufs und eine Operation kann weitere Operationsaufrufe enthalten. Die Operationsaufrufe einer Operation sind eine geordnete Liste. Beim Aufruf einer Operation werden alle Aufrufe dieser Liste sequentiell ausgeführt, bevor der Rückgabewert an den Aufrufer der Operation zurückgegeben wird. Zusätzlich kann eine Operation eine Menge von *Variablen* enthalten. Diese repräsentieren Parameter, Rückgabevariablen und Zustandsvariablen (state variables). Ein Operationsaufruf kann die Charakteristiken von Parametern und Zustandsvariablen durch die Elemente der Klasse *Variablenzuweisung* (VariableAssignment) spezifizieren. Analog spezifiziert eine Operation die Charakteristiken ihrer Rückgabevariablen. Ein Charakteristiktyp wird im Operationsmodell als eine wohldefinierte Menge von booleschen Variablen modelliert und als Attribut eines Datums bezeichnet. Variablenzuweisungen werden durch aussagenlogische Formeln repräsentiert. In dem in Abbildung 2.2 dargestellten Beispiel wäre der Charakteristiktyp *Sensitivität* durch die zwei booleschen Variablen *Geheim* und *Verschlüsselt* dargestellt. Für die vom Nutzer ausgehenden Daten gilt im korrespondierenden Operationsmodell, dass *Geheim* wahr

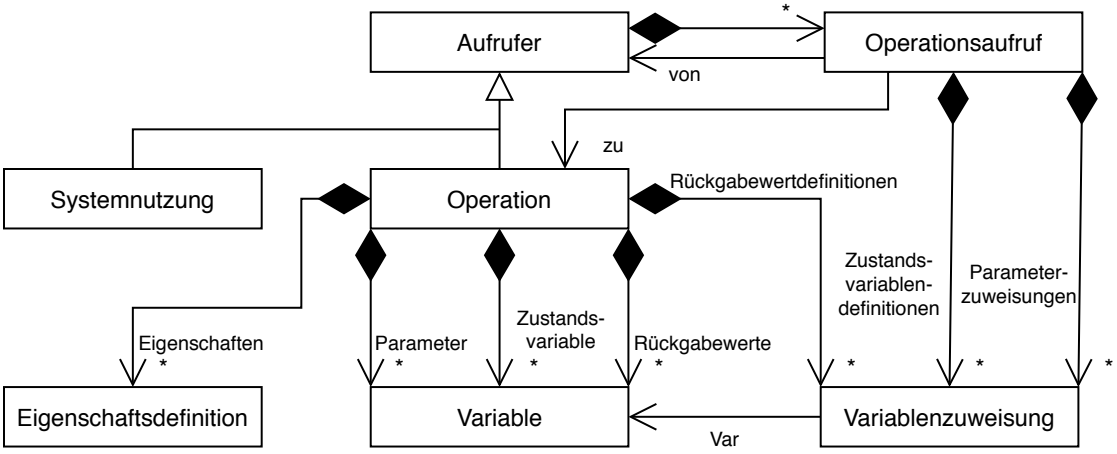

Abbildung 2.3.: Vereinfachtes Metamodell des Operationsmodells. Basierend auf [12].

und *Verschlüsselt* falsch ist. Die konkreten Charakteristiken werden im Operationsmodell als Wert eines Attributs bezeichnet.

Abschließend verfügen *Operationen* noch über *Eigenschaftsdefinitionen* (PropertyDefinitions), die Charakteristiken repräsentieren, welche an andere Modellelemente als die Daten selbst annotiert werden. Der Charakteristiktyp *Ort* aus Abbildung 2.2 ist ein Beispiel für eine solche Eigenschaft (Property). Analog zu Attributen wird die konkrete Charakteristik (z. B. *EU*) als Wert einer Eigenschaft (z. B. *Ort*) bezeichnet. Diese Unterscheidung zwischen Attributen und Eigenschaften hat den Zweck zwischen sich im Verlauf des Datenflusses ändernden Werten (die Werte eines Attributs) und den einer Operation zugewiesenen statischen Werten (die Werte einer Eigenschaft) zu unterscheiden. Dies erleichtert die Definition einer Abfrage zur Sicherheitsanalyse an die Abfrage-Schnittstelle.

Ein Beispiel für ein Operationsmodell ist in Abbildung 2.4 dargestellt. Eine Operation *Nutzer* ruft die Operation *Verschlüsselung* auf, welche anschließend die Operation *Datenbank* aufruft. Dabei sind die Parameter jeweils vom Typ *Datum*. Dieser Datentyp verfügt über die zwei Attribute *Sensitivität* und *Quellort*. Die Operation *Verschlüsselung* setzt für das ausgehende Datum *b* den Wert *Geheim* des Attributs *Sensitivität* auf falsch und den Wert *Verschlüsselt* des gleichen Attributs auf wahr. Alle Werte des Attributs *Quellort* werden von dem Parameter *a* kopiert. Die aussagenlogische Formel *pa* bezeichnet dabei eine Parameterreferenz auf die Werte *V* des Attributs *Quellort* des Parameters *a*. Eine mögliche Abfrage an dieses Modell, wäre z. B. ob eine Operation existiert, deren Wert *US* der Eigenschaft *Ort* wahr ist und die einen Parameter erhält, dessen Wert *EU* des Attributs *Quellort* und dessen Wert *Geheim* des Attributs *Sensitivität* wahr sind.

Diese Abfragen werden in einem Prolog-Programm realisiert, das aus dem Operationsmodell generiert wird. Im Gegensatz zu der Transformation des Eingabemodells in das Operationsmodell handelt es sich bei der Transformation des Operationsmodells in das Prolog-Programm um keine weitere Vereinfachung. Stattdessen werden die Datenflüsse und die Struktur des Operationsmodells durch Prolog-Fakten dargestellt und eine Menge

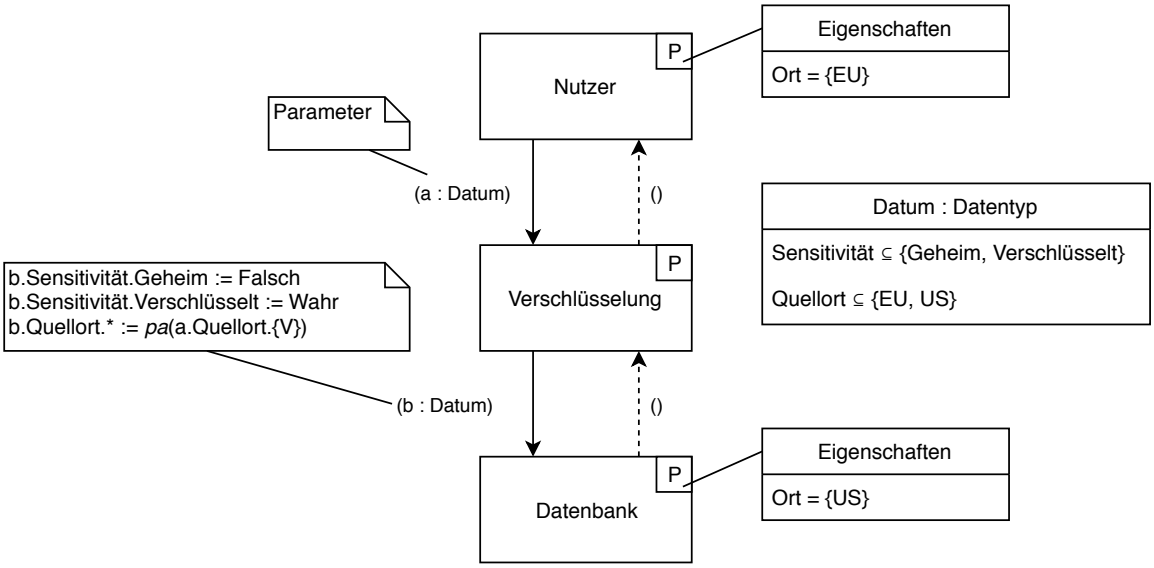

Abbildung 2.4.: Ein Beispiel für eine Operationsmodellinstanz. Basierend auf [24].

von Prädikaten definiert, mit denen es möglich ist, Abfragen an diese textuelle Repräsentation des Operationsmodells zu stellen. Dies wird im Folgenden als Abfrage-Schnittstelle bezeichnet. Unter Verwendung dieser Abfrage-Schnittstelle ist es möglich, analysespezifische Abfragen zu definieren, welche in das entstehende Prolog-Programm eingefügt werden. Die Transformation aus dem Operationsmodell und das Hinzufügen der in einer eigenen Datei vorliegenden analysespezifischen Prolog-Abfragen läuft automatisiert ab. Anstatt die vollständige Schnittstelle [24] zu beschreiben, wird im Folgenden eine beispielhafte Abfrage an das bestehende Beispiel aus Abbildung 2.4 definiert, um die grundlegende Idee zu erklären. Textuell kann die Beispielabfrage folgendermaßen formuliert werden. Gibt es eine Operation *OP*, für die die der Wert *US* der Eigenschaft *Ort* wahr ist und die über einen Eingabeparameter *P* verfügt, für den gilt:

- Der Wert *Geheim* des Attributs *Sensitivität* ist wahr und
- Der Wert *EU* des Attributs *Quellort* ist wahr

In der in Algorithmus 2.1 dargestellten Prolog-Abfrage sind alle verwendeten Prädikate vordefiniert und die dazugehörigen Fakten werden als Teil der Transformation des Prolog-Programms aus dem Operationsmodell generiert. Der einzige manuelle Aufwand ist die Definition der Abfrageformel. Dieser Aufwand ist jedoch einmalig, wenn die Abfrage parametrisierbar definiert wird. Zum einfacheren Verständnis wird in diesem Beispiel jedoch darauf verzichtet. Bei der Beschreibung eines Prädikats wird die im SWI-Prolog-Handbuch [25] verwendete Syntax *Prädikat\Arität* verwendet.

In Zeile 2 ist *isOperation\1* genau dann wahr, wenn *OP* der Name einer Operation ist. Hier dient es als Generator für alle Operationen des Operationsmodells. Um diese Operationen auf die für diese Analyse relevanten Operationen zu filtern werden *hasProperty\2* (Zeile 3) und *operationProperty\3* (Zeile 4) verwendet. Das Prädikat *hasProperty\2* ist genau dann wahr, wenn die Operation *OP* eine Eigenschaftsdefinition für die Eigenschaft mit dem

```
1 securityViolation(OP) :-
2     isOperation(OP),
3     hasProperty(OP, 'Ort'),
4     operationProperty(OP, 'Ort', 'US'),
5     operationParameter(OP, P),
6     S = [OP|_],
7     stackValid(S),
8     callArgument(S, P, 'Sensitivität', 'Geheim'),
9     callArgument(S, P, 'Quellort', 'EU').
```

Algorithmus 2.1.: Ein Beispiel für eine Prolog-Abfrage einer DCP-Analyse.

Namen *Ort* hat, während *operationProperty\3* nur wahr ist, wenn die Operation *OP* für die Eigenschaft *Ort* den Wert *US* gesetzt hat.

In Zeile 5 wird *operationParameter\2* als Generator verwendet, um alle Parameter *P* der Operation *OP* zu finden. Für diese Analyse ist nur die aktuelle Operation interessant. Tatsächlich steht jedoch die vollständige Aufrufhierarchie zur Verfügung, um auch komplexere Bedingungen zu überprüfen. Das Prädikat *callArgument\4* bekommt als ersten Eingabeparameter die Aufrufhierarchie, verwendet aber nur die aktuelle Operation. Deswegen wird in Zeile 6 die Aufrufhierarchie *S* zugewiesen und in Zeile 7 überprüft, ob es sich um eine gültige Hierarchie handelt. Schließlich kann mit dem Prädikat *callArgument\4* überprüft werden, ob die aktuelle Operation der Aufrufhierarchie *S* einen Eingabeparameter *P* besitzt, für den *Geheim* des Attributs *Sensitivität* wahr ist. Analog wird der Wert *EU* des Attributs *Quellort* in Zeile 9 geprüft.

Wenn eine gültige Variablenbelegung für *OP*, *P* und *S* gefunden wird, liegt ein Szenario vor, in dem vertrauliche Daten, die ursprünglich aus der EU stammen, von einer Operation, deren zugehöriger Rechnerknoten außerhalb der EU platziert ist, verarbeitet werden. Je nach Kontext kann dies eine relevante Sicherheitslücke sein, die behoben werden muss. Wird keine solche Variablenbelegung gefunden, kann durch die Analyse keine Sicherheitslücke festgestellt werden.

2.3. UMLsec

UMLsec [1] ist eine Erweiterung der Modellierungssprache UML und dient zur Modellierung von Sicherheitseigenschaften eines Systems. Die Erweiterung erfolgt in Form eines UML-Profils und folgt damit dem üblichen, nicht-invasiven Erweiterungsmechanismus von UML anstelle einer Erweiterung des UML-Metamodells. Ein UML-Profil besteht unter anderem aus Stereotypen, Eigenschaftsdefinitionen (tag definitions) und Einschränkungen (constraints). Stereotypen erweitern die Semantik bestehender Typen oder Klassen des UML-Metamodells. Syntaktisch werden sie mit dem Namen des Stereotypen in

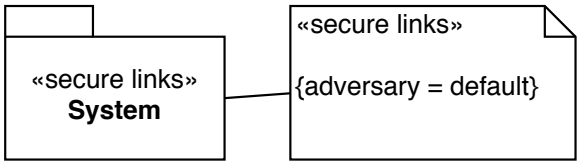

Abbildung 2.5.: Ein mit einem Stereotyp annotiertes Paket.

Guillemets («») dargestellt. Dabei sind Stereotypen immer einem bestehenden Modellelement zugeordnet. In Abbildung 2.5 ist der Stereotyp «`secure links`» zu sehen, der für UML-Pakete definiert und dem Paket *System* zugeordnet ist. Ein Stereotyp kann um Eigenschaftsdefinitionen erweitert werden. Wird dieser Stereotyp auf ein konkretes Modellelement angewandt, kann für jede Eigenschaftsdefinition der Eigenschaftswert (tagged value) gesetzt werden. Dabei handelt es sich um Schlüssel-Wert-Paare, die üblicherweise in der Syntax *{Schlüssel = Wert}* in einem mit dem Modellelement verbundenen Kommentar dargestellt werden. Ein Beispiel für einen Eigenschaftswert ist in Abbildung 2.5 die Definition der Angreifermächtigkeit des Stereotyp «`secure links`». Schließlich kann UML um Einschränkungen erweitert werden, welche erfüllt sein müssen, um ein gültiges UML-Diagramm darzustellen. Der Stereotyp «`secure links`» verfügt über Einschränkungen, die die Einhaltung der Bedingungen der Secure-Links-Analyse sicherstellen. Diese Analyse ist in Kapitel 5 näher erläutert.

Die Stereotypen von UMLsec können in drei verschiedene Arten unterteilt werden.

- Sicherheitsannahmen über die physischen Eigenschaften eines Systems, z. B. der Stereotyp «`Internet`», der beschreibt, dass ein Kommunikationspfad (communication path) eine Verbindung über das Internet repräsentiert.
- Sicherheitsanforderungen an Daten oder logische Elemente des Systems, z. B. der Stereotyp «`secrecy`», der beschreibt, dass die Daten zwischen zwei Softwareartefakten nicht von Unbefugten gelesen werden sollen.
- Sicherheitsregeln (security policies), die von Teilen des Systems erfüllt werden sollen. Ein Beispiel hierfür ist der Stereotyp «`secure links`», der besagt, dass das an Abhängigkeiten zwischen Softwareartefakten geforderte Sicherheitsniveau von den physischen Kommunikationspfaden erfüllt werden muss.

Die ersten beiden Arten dienen dazu, das System um zusätzliche Informationen zu erweitern, welche für die Sicherheitsanalysen relevant sind. Die Sicherheitsregeln stellen Einschränkungen dar, die von dem Modell erfüllt werden müssen. Ist ein solcher Stereotyp an einen Teil des Modells annotiert, aber die Einschränkungen werden nicht erfüllt, ist das Resultat ein ungültiges Modell.

Um automatisiert überprüfbare Einschränkungen zu definieren, verwendet UMLsec ein analysierbares Modell der Verhaltenssemantik des verwendeten Teils von UML. In UML können Objekte und Systemkomponenten Nachrichten austauschen. Daher erhält jedes Objekt oder Subsystem O in dem Analysemodell aus UMLsec eine Eingabewarteschlange in_O und eine Ausgabewarteschlange out_O. Das Versenden einer Nachricht von Objekt oder Subsystem S zu einem Objekt oder Subsystem R erfolgt folgendermaßen.

- S platziert die Nachricht M in out_S
- Die Nachrichten werden von den Ausgabewarteschlangen auf die gewünschten Eingabewarteschlangen verteilt. Dabei wird M aus out_S entfernt und in in_R eingefügt.
- R entfernt die Nachricht M aus in_R und verarbeitet ihren Inhalt.

Das Verhalten eines Objekts ist durch Verhaltensdiagramme oder Sequenzdiagramme modelliert. Das Verhalten eines Subsystems kann über das Verhalten der enthaltenen Objekte modelliert werden, indem ein Aktivitätsdiagramm zwischen dem Verhalten der einzelnen enthaltenen Komponenten koordiniert. So wird für jedes Objekt oder jede Komponente C eine sogenannte UML-Maschine $[\![C]\!]$ definiert. Dabei handelt es sich um einen Zustandsautomat, der mit seiner Umgebung über Nachrichten kommuniziert. Das Verhalten eines Angreifers wird ebenfalls als UML-Maschine modelliert. Um die Sicherheit eines Systems S vor einem Angreifer A zu evaluieren, wird die UML-Maschine $[\![S]\!]$ unter der Präsenz des Angreifers A ausgeführt. Dies wird als $[\![S]\!]_A$ dargestellt. $[\![S]\!]_A$ entspricht der Ausführung von $[\![S]\!]$, wobei nach jeder Iteration der Systemausführung der Angreifer gemäß seiner Mächtigkeit die Inhalte der Eingabe- und Ausgabewarteschlangen nicht-deterministisch ändern kann. Über das Lesen oder die Modifikation der Warteschlangen werden Angriffe abgebildet.

Des Weiteren definiert UMLsec mehrere Sicherheitseigenschaften, die für verschiedene Sicherheitsanalysen relevant sind. Aus diesem Grund werden sie hier kurz erläutert. Eine vollständige, formale Definition unter Verwendung der UML-Maschine ist für diese Arbeit nicht notwendig und wird daher an dieser Stelle ausgelassen.

Mit Vertraulichkeit (secrecy) ist die Eigenschaft gemeint, die sicherstellt, dass Daten nur von legitimierten Parteien gelesen werden können. Ein System S erhält die Vertraulichkeit eines Datums d, wenn das System niemals Daten ausgibt aus denen d abgeleitet werden kann. Genauer ist die Vertraulichkeit von d genau dann verletzt, wenn es einen Angreifer A gibt, der zu Beginn der Ausführung von $[\![S]\!]_A$ das Datum d nicht kannte, aber eine Eingabesequenz so existiert, dass A nach der Ausführung von $[\![S]\!]_A$ d kennt. In UMLsec wird geforderte Vertraulichkeit durch den Stereotyp «`secrecy`» oder die Eigenschaft `secrecy` des Stereotyps «`critical`» dargestellt.

Analog dazu ist die Integrität definiert. Wenn während der Ausführung $[\![S]\!]_A$ eines Systems S unter der Präsenz eines Angreifers A eine Variable einen Wert zugewiesen bekommt, der von dem Wert abweicht, den diese Variable bei gleicher Eingabesequenz ohne die Präsenz des Angreifers A hätte, gilt die Integrität dieser Variable als verletzt, da der Angreifer für die Änderung dieser Variable verantwortlich sein muss. Diese Definition stellt das Gegenstück zur Definition der Vertraulichkeit dar. Vertraulichkeit verhindert den Informationsfluss aus geschützten Quellen zu nicht vertrauenswürdigen Empfängern, während Integrität den Informationsfluss in der entgegengesetzten Richtung verhindert. In UMLsec wird geforderte Integrität durch den Stereotyp «`integrity`» oder die Eigenschaft `integrity` des Stereotyps «`critical`» dargestellt.

Abschließend existiert eine weitere Art der Definition von Vertraulichkeit und Integrität von Daten in UMLsec. In diesem Fall wird zwischen Daten mit hoher Sensitivität und hoher

Vertrauenswürdigkeit (high) und Daten mit geringer Sensitivität und Vertrauenswürdigkeit (low) unterschieden. Wenn vertrauenswürdige Komponenten eines Systems mit nicht vertrauenswürdigen Komponenten interagieren, muss sichergestellt werden, dass kein indirekter Informationsfluss von vertrauenswürdigen zu nicht vertrauenswürdigen Komponenten besteht. Daher wird die Sicherheitsregel definiert, dass kein Informationsfluss von hoher Sensitivität zu niedriger Sensitivität stattfinden darf (no down-flow). Daten von niedriger Sensitivität dürfen Daten von hoher Sensitivität beeinflussen, aber nicht umgekehrt. Genauer ausgedrückt darf die Ausgabe eines Datums niedriger Sensitivität nicht von Eingaben hoher Sensitivität abhängen. Dies stellt eine andere Art der Definition von Vertraulichkeit dar. Das Gegenstück dazu ist die Regel, dass kein Informationsfluss von niedriger Sensitivität zu hoher Sensitivität stattfinden darf (no up-flow). Damit wird sichergestellt, dass nicht vertrauenswürdige Komponenten des Systems nicht Daten hoher Sensitivität beeinflussen. Genauer darf die Ausgabe eines Datums hoher Sensitivität nicht von Eingabedaten niedriger Sensitivität abhängen. Das ist eine Alternative zu der Definition von Integrität. Zusammen werden diese beiden Regeln sicherer Informationsfluss (secure information flow) oder Nichteinmischung (non-interference) genannt. Wenn diese Eigenschaft für Daten gefordert ist, wird dies in UMLsec durch den Stereotyp «`high`» oder die Eigenschaft `high` des Stereotyps «`critical`» modelliert.

3. Verwandte Arbeiten

In diesem Kapitel wird eine Übersicht über existierende Arbeiten, die in Bezug zu dem in dieser Arbeit verwendeten Ansatz stehen, präsentiert. Dabei wird zunächst auf weitere Ansätze zur Modellierung und Analyse von Sicherheitseigenschaften auf Architekturebene eingegangen und dargestellt, inwieweit sie sich von dem in dieser Arbeit für den Vergleich gewählten Ansatz UMLsec unterscheiden. Anschließend wird eine Übersicht über andere Arbeiten gegeben, in denen verschiedene Sicherheitsanalysen und -modellierungsansätze miteinander verglichen werden. Hierbei wird insbesondere auf die zum Vergleich verwendeten Methoden geachtet und darauf, inwieweit diese Methoden für den Vergleich zwischen UMLsec und DCP anwendbar sind. Abschließend werden andere Ansätze betrachtet, in denen UML-Modelle auf PCM-Modelle abgebildet werden, um die Erkenntnisse dieser Ansätze auf die in dieser Arbeit erstellten Modelltransformationen anzuwenden.

3.1. Ansätze zur Sicherheitsanalyse

Zunächst ist zu erwähnen, dass für diese Arbeit nur Analyseansätze relevant sind, die auf Architekturebene operieren. Es gibt verschiedene Ansätze, die den Programm-Code als Eingabe benötigen, wie z. B. Code-Verifikation [26] oder das Überprüfen welche Daten im Speicher direkt von Nutzereingaben beeinflusst werden können (taint analysis) [27]. Diese Ansätze erlauben allerdings keine frühzeitige Aufdeckung von Sicherheitsproblemen und damit auch keine frühzeitige Behebung dieser Probleme. Da dies jedoch ein zentrales Ziel von DCP ist, werden im Folgenden lediglich Ansätze betrachtet, die auf der Basis eines Architekturmodells operieren und keinen Programmcode zur Durchführung der Analyse benötigen.

Sharma u. a. [28] stellen einen Ansatz vor, in dem die Software-Architektur als Markov-Kette modelliert wird, auf deren Basis Vorhersagen für Performanz, Zuverlässigkeit, Sicherheit und Cache-Performanz getroffen werden. Dabei setzt die Sicherheitsanalyse darauf, dass für jede verwendete Komponente ein Verwundbarkeitsindex (vulnerability index) definiert ist, der die Wahrscheinlichkeit einer Verwundbarkeit dieser Komponente in einer Ausführung beschreibt. Aus diesen einzelnen Verwundbarkeitsindizes wird unter Verwendung der Markov-Kette ein Verwundbarkeitsindex für das gesamte System ermittelt. Diese Art des Ansatzes ist im Kontext dieser Arbeit nicht für einen Vergleich mit DCP geeignet, da die in DCP definierten Sicherheitsanalysen das Ziel haben, konkrete Sicherheitsprobleme in der

Architektur zu finden und nicht eine Wahrscheinlichkeit für eine Sicherheitsverletzung zurückzugeben.

In [29] wird von Almorsy u. a. ein Ansatz vorgestellt, in dem Architekturmodelle durch Sicherheitsszenarien ergänzt werden, zu denen Metriken durch OCL-Ausdrücke definiert werden. Durch eine statische Analyse des Systems und dieser Szenarien werden Ergebnisse für die Metriken der Sicherheitsszenarien erzeugt. Es gibt hierbei sowohl Szenarien, die sich für eine Umsetzung in DCP eignen, wie z. B. das Angriffsszenario Man-in-the-Middle, dessen OCL-Ausdruck überprüft, ob zwei Komponenten existieren, die über einen unverschlüsselten Kommunikationskanal kommunizieren und keine kryptographischen Verfahren zum Erhalt von Vertraulichkeit und Integrität verwenden. Andere Szenarien verwenden jedoch OCL-Ausdrücke, welche ein Verhältnis oder eine andere reellwertige Ausgabe liefern. Diese eignen sich nicht für einen Vergleich mit DCP. Bei allen Metriken und OCL-Ausdrücken handelt es sich aber lediglich um Beispiele und die Autoren weisen darauf hin, dass diese Beispiele nicht vollständig sind und wiederverwendbare Metriken und OCL-Ausdrücke für die Szenarien von Sicherheitsexperten entwickelt werden müssen. Sie eignen sich daher in dieser Form nicht als Basis für einen Vergleich der Ausdrucksmächtigkeit in Bezug auf DCP.

Ein weiterer Ansatz betrachtet sichere Objektflüsse in Service-orientierten Architekturen [30]. Dieser Ansatz erlaubt es, Vertraulichkeit und Integrität von wichtigen Objekten, die zwischen verschiedenen Teilnehmern eines auf Service-orientierten Architekturen basierenden Geschäftsprozesses kommuniziert werden, sicherzustellen. Ähnlich zu UMLsec wird hierfür ebenfalls eine UML-Erweiterung präsentiert, die die Modellierung von sicheren Objektflüssen ermöglicht. Im Gegensatz zu diesem Ansatz, der im Wesentlich eine Taint-Analyse (taint analysis) darstellt, bietet UMLsec eine größere Menge von Analysen. Für die Betrachtung der Aussagemächtigkeit von DCP ist daher UMLsec vorzuziehen.

Abschließend sind Ansätze zu erwähnen, die nicht nur eine Analyse bereitstellen, sondern, wie auch DCP, einen Entwicklungsprozess zur Entwicklung von sicherheitskritischen Systemen definieren. IFlow [31] ist ein Ansatz zur modellgetriebenen Entwicklung von verteilten Anwendungen. Dabei wird ein Entwicklungsprozess definiert, in dem zunächst ein abstraktes Modell der Systeme, des Verhaltens und der Sicherheitsanforderungen erstellt wird. Aus diesem Modell wird zum einen Programmcode generiert, auf dem automatisierte Analysen überprüfen können, ob sicherer Informationsfluss gegeben ist. Zum anderen wird das abstrakte Modell in ein formales Modell transformiert, auf dem mittels eines Theorembeweisers zusätzliche Sicherheitsanalysen durchgeführt werden können. Da UMLsec ein größeres Spektrum von Analysen bietet und nicht nur Analysen des Informationsflusses, ist UMLsec für den Vergleich in dieser Arbeit vorzuziehen.

Ein weiterer Ansatz, der einen Entwicklungsprozess definiert basiert auf der Kombination von Secure Tropos und UMLsec [32]. Secure Tropos wird verwendet, um Sicherheitsanforderungen auf einer soziotechnischen Ebene zu modellieren und zu analysieren. In [32] wird von Mouratidis u. a. ein Ansatz vorgestellt, wie aus dem Secure-Tropos-Modell UMLsec-Modelle generiert werden können, sodass insgesamt ein Entwicklungsprozess entsteht, der sowohl die Modellierung der sozialen und technologischen Dimension von Sicherheit erlaubt. Für die technologischen Analysen wird UMLsec verwendet, mit dem in dieser

Arbeit DCP verglichen wird. Die Betrachtung von Entwicklungsprozessen ist nicht Fokus dieser Arbeit. Für eine zukünftige Arbeit kann es jedoch sinnvoll sein, diesen Ansatz bezüglich eines Vergleichs der Entwicklungsprozesse zu erwägen.

3.2. Methoden zum Vergleich von Sicherheitsmodellierungsansätzen

Es existieren mehrere Methoden, um Sicherheitsmodellierungsansätze miteinander zu vergleichen und zu evaluieren. In der Regel werden dabei bestimmte Aspekte der gewählten Ansätze miteinander verglichen. So wird in einer Umfrage von Villarroel u. a. [33] ein von Khwaja und Urban entwickeltes Rahmenwerk [34] zur Evaluation und zum Vergleich von Softwarespezifikationen und Spezifikationstechniken verwendet. Vallarroel u. a. vergleichen damit elf verschiedene Methoden, um sichere Softwaresysteme zu entwerfen. Dabei ist der Kern des Vergleichs jedoch die Qualität der Spezifikationen dieser Ansätze und das Vergleichskriterium z. B. die Verständlichkeit oder Vollständigkeit. Im Rahmen dieser Arbeit soll jedoch die Ausdrucksmächtigkeit von DCP im Vergleich zu UMLsec untersucht werden und nicht die Qualität der Spezifikation dieser Ansätze.

Dagegen wird von Matulevičius und Dumas [35] ein Vergleich von SecureUML und UMLsec auf die Eignung für Szenarien der rollenbasierten Zugriffskontrolle (role-based access control) vorgestellt. Hierbei wird sowohl ein genereller Vergleich zwischen UMLsec und SecureUML anhand eines vorher aufgestellten Kriterienkatalogs vorgenommen, als auch eine Abbildung der Konzepte der rollenbasierten Zugriffskontrolle auf die unterschiedlichen Modellelemente, die diese repräsentieren. Ein ähnlicher Ansatz wird von Mazur und Ksiezopolski [36] vorgestellt. Dabei handelt es sich um den Vergleich von SecureUML, UMLsec, PL/SQL und QoP-ML. Es wird auf die Verwendung eines Rahmenwerks [37], das eine Menge von Kriterien für die Bewertung und den Vergleich der Qualität von Sicherheitsmodellen definiert, zurückgegriffen. Die beiden genannten Ansätze untersuchen und vergleichen allerdings nur die Sicherheitsmodelle auf Eigenschaften wie z. B. semantische Qualität oder syntaktische Qualität. Dagegen ist der Fokus dieser Arbeit die Untersuchung der Sicherheitsanalysen, die in UMLsec und DCP möglich sind. Es wird keine qualitative Bewertung der Sicherheitsmodelle vorgenommen.

Schließlich existiert ein Vergleich verschiedener Entwicklungsprozesse für sichere Softwaresysteme [38]. Auch hier wird wieder anhand von Kriterienkatalogen, die Kriterien für die Bewertung von Entwicklungsprozessen bereitstellen, ein Vergleich zwischen elf Ansätzen gezogen. Alle diese Ansätze sind im Kontext dieser Arbeit nicht anwendbar, da hier die Ausdrucksmächtigkeit zwischen DCP und UMLsec untersucht wird. Da der Fokus nicht auf dem Vergleich der Spezifikationstechnik, der Qualität der Modelle oder dem Entwicklungsprozess liegt, sind Kriterienkataloge, die diese Aspekte behandeln, für das Vorgehen dieser Arbeit nicht anwendbar. Des Weiteren konnten keine Vergleiche von Sicherheitsmodellierungsansätzen bezüglich der definierten Sicherheitsanalysen gefunden werden.

3.3. Modelltransformationen zwischen UML und PCM

Klatte beschreibt in [39] die Konsistenzhaltung zwischen UML-Modellen und Palladio-Komponentenverzeichnissen. Dabei werden Modellelemente gemäß ihrer Semantik aufeinander abgebildet, z. B. erfolgt die Abbildung einer Komponente des PCM-Komponentenverzeichnisses auf eine UML-Komponente. Für diese Arbeit kann die definierte Abbildung als Basis für die Modelltransformationen dienen. Dabei können jedoch hauptsächlich strukturelle Komponenten wiederverwendet werden. Die Wiederverwendbarkeit ist außerdem abhängig von der betrachteten Analyse eingeschränkt. In dieser Arbeit muss die Semantik der Analyse erhalten bleiben. Da die Analyse in DCP über ein anderes Paradigma realisiert ist als die UMLsec-Analyse, müssen hier einige strukturelle Elemente bzw. Stereotypen des UMLsec-Modells als Daten oder Datenflüsse in DCP umgesetzt werden. Des Weiteren ist die Erstellung mancher Modellelementen in DCP durch die Notwendigkeit des Datenflusses bedingt, z. B. müssen in der Secure-Links-Analyse abhängig von der Menge an unabhängigen Datenflusspfaden durch das System gegebenenfalls zusätzliche Komponenteninstanzen erzeugt werden, für die keine korrespondierenden Artefakte in dem UMLsec-Modell existieren.

4. Methode

In diesem Kapitel wird die Vorgehensweise der Untersuchung beschrieben. Um einen Rückschluss auf die Ausdrucksmächtigkeit von UMLsec und DCP ziehen zu können, wird die Abbildbarkeit von UMLsec-Sicherheitsanalysen auf äquivalente DCP-Analysen untersucht. Hierzu werden verschiedene Analysen aus UMLsec als Kandidaten für eine Darstellung in DCP in Betracht gezogen. Die Kandidaten, die hierfür am besten geeignet sind, werden anschließend im Kontext dieser Arbeit betrachtet. Dazu muss für die gewählte Sicherheitsanalyse zunächst eine äquivalente Analyse in DCP konzipiert werden. Anschließend kann eine Transformation zwischen dem Eingabemodell der UMLsec-Analyse und dem Eingabemodell der konzipierten DCP-Analyse definiert werden. Zur besseren Evaluierbarkeit wird zusätzlich eine Transformation definiert, die das DCP-Eingabemodell zurück in ein UMLsec-Eingabemodell überführt. Abschießend können anhand der Abstraktionsunterschiede der Analysen und der Uneindeutigkeiten der Transformation der Eingabemodelle Szenarien identifiziert werden, die in UMLsec oder in DCP nicht darstellbar sind. Aus diesen Szenarien werden die Unterschiede in der Ausdrucksmächtigkeit hergeleitet. Die jeweils pro Analyse notwendigen Schritte von der Auswahl einer Analyse bis zur Umsetzung in DCP werden im Folgenden näher beschrieben und sind in Abbildung 4.1 schematisch dargestellt.

4.1. Auswahlkriterien

Die Auswahl der in dieser Arbeit betrachteten UMLsec-Sicherheitsanalysen erfolgt nach definierten Kriterien. Diese sind im Folgenden beschrieben. Bei dem ersten Auswahlkriterium handelt es sich um die Frage, ob eine Referenzimplementation der betrachteten UMLsec-Analyse existiert. Dies ist zum einen relevant, da die Beschreibung einer UMLsec-Analyse in dem Buch von Jürjens [1, Kapitel 3 und 4] im Allgemeinen keine erschöpfende Erklärung der gültigen Eingabemodelle und der Behandlung von Randfällen beinhaltet. Die Referenzimplementation dient daher dazu, das Verhalten in solchen Fällen zu ermitteln. Weiter ist die Referenzimplementation für die Überprüfung der Äquivalenz der DCP-Analyse zu der entsprechenden UMLsec-Analyse relevant. Im Folgenden ist ein Überblick gegeben, wie überprüft wird, dass tatsächlich eine äquivalente Analyse in DCP definiert wird. Dazu werden für eine ausreichende Menge von Testfällen die Ergebnisse der UMLsec-Analyse mit den Ergebnissen der konzipierten und implementierten DCP-Analyse verglichen. Dies setzt jedoch das Vorhandensein einer Implementation der UMLsec-Analysen voraus, die als Grundlage für den Vergleich verwendet werden kann.

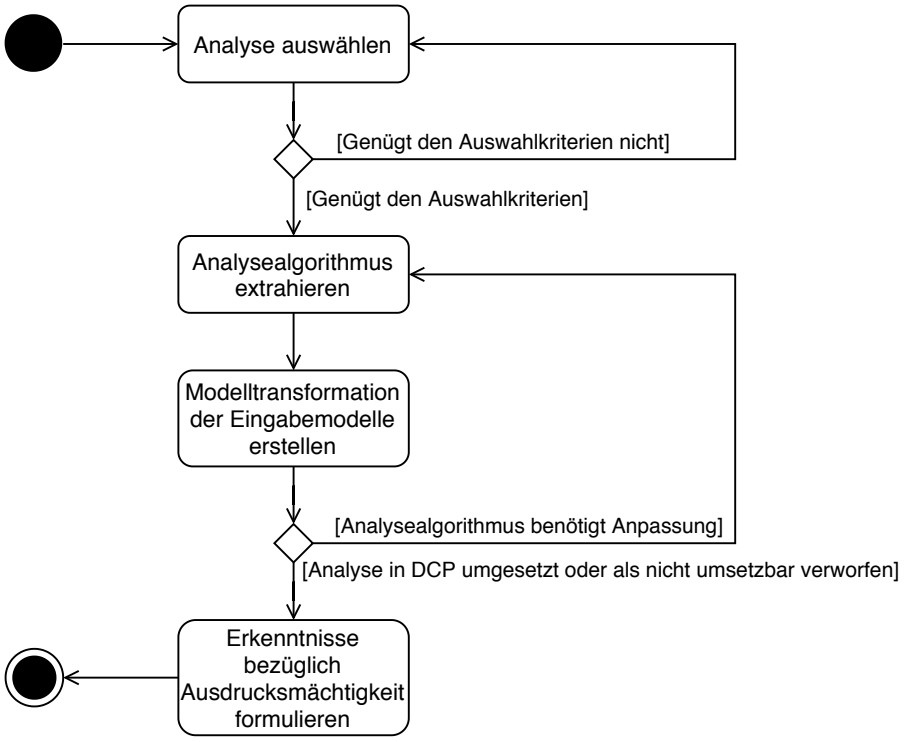

Abbildung 4.1.: Schematischer Ablauf der Umsetzung einer UMLsec-Sicherheitsanalyse in DCP.

Es existieren hierfür zwei Anwendungen, die UMLsec-Analysen implementieren. Zum einen das veraltete und nicht mehr unterstützte UML Analysis Tool [40] und zum anderen das neuere, aber weniger ausführlich dokumentierte CARiSMA-Werkzeug [41]. Für die Evaluation der erstellten DCP-Analyse wird das CARiSMA-Werkzeug präferiert, da es den aktuelleren Stand der Forschung und Entwicklung darstellt und für viele Analysen bereits Beispiele von Eingabemodellen bietet, auf deren Basis weitere Testfälle erzeugt werden können. Die Existenz einer Referenzimplementation einer UMLsec-Analyse ist demnach ein zu beachtendes Kriterium bei der Auswahl einer UMLsec-Analyse.

Das zweite Auswahlkriterium ist die Ausführlichkeit der Dokumentation der UMLsec-Analyse. Es ist bereits beschrieben, dass die Beschreibungen der Analysen in dem Buch von Jürjens [1, Kapitel 3 und 4] in der Regel keine erschöpfende Betrachtung aller gültigen Eingabemodelle oder Randfälle liefert. Für diese Arbeit ist es ausreichend, wenn die Dokumentation ausführlich genug ist, dass die Semantik der Analyse erfasst werden kann. Dies ist dann der Fall, wenn aus den definierten UML-Einschränkungen der Analyse logische Formeln extrahiert werden können, die dieselbe Semantik beschreiben und als Basis für die Definition der DCP-Analyse dienen können.

Schließlich ist das letzte Auswahlkriterium der Anwenderfokus der Analyse. UMLsec erlaubt das Modellieren und Analysieren von Sicherheitseigenschaften auf einer Vielzahl von unterschiedlichen Abstraktionsebenen. Darunter sind für Entwickler schnell verständliche UML-Modelle in Form von Aktivitäts-, Verteilungs- oder Klassendiagram-

men, aber auch kompliziertere Modelle zur Darstellung von Struktur und Kontrollflüssen von kryptographischen Protokollen. Letztere sind für Entwickler ohne Hintergrund in Kryptographie nicht ohne Weiteres verständlich. Da der Ansatz von DCP jedoch darauf fokussiert ist, auch von Entwicklern ohne kryptographisches Wissen verwendet werden zu können, werden im Kontext dieser Arbeit präferiert UMLsec-Analysen für den Vergleich gewählt, die Sicherheitseigenschaften von Softwaresystemen auf einer Abstraktionsebene analysieren, die für diese Art von Entwicklern verständlich ist.

Ein Faktor, der nicht als Auswahlkriterium in Frage kommt, aber dennoch zu erwähnen ist, ist der Abstraktionsunterschied der UMLsec-Eingabemodelle einer Analyse zu den erwarteten DCP-Eingabemodellen. Es wird keine Einschränkung bezüglich dieses Abstraktionsunterschieds getroffen, da zu erwarten ist, dass sowohl aus der Betrachtung einer Analyse mit niedrigem bzw. hohem Abstraktionsunterschied zwischen den Eingabemodellen, relevante Erkenntnisse bezüglich der Ausdrucksmächtigkeit gewonnen werden können.

Um diese Auswahlkriterien in nachfolgenden Kapiteln einfacher referenzieren zu können, werden sie im Folgenden nummeriert und als Fragen formuliert. Diese Fragen werden bei der Begründung der Auswahl der betrachteten Analysen beantwortet.

1. Existiert eine Referenzimplementation für die betrachtete Analyse?
2. Ist die Dokumentation der Analyse ausreichend?
3. Handelt es sich bei der Analyse um ein Szenario, das üblicherweise von Entwicklern modelliert werden kann ohne dafür weiteres Wissen über z. B. kryptographische Protokolle zu benötigen?

4.2. Analyse

Für eine Analyse, die allen Auswahlkriterien genügt, wird eine äquivalente Analyse in DCP umgesetzt. Hierbei ist zwischen dem eigentlichen Analysealgorithmus und den Eingabemodellen zu unterscheiden. Da die Eingabemodelle abhängig von dem gewählten Analysealgorithmus sind, wird zunächst der Analysealgorithmus erstellt und erst anschließend die Transformation zwischen den Eingabemodellen definiert.

Der Analysealgorithmus aus UMLsec (siehe Abschnitt 2.3) verwendet einen nicht-deterministischen Ansatz, der auf intern miteinander kommunizierenden Zustandsautomaten für das System und einem Zustandsautomaten für den Angreifer basiert. Dagegen verwendet DCP einen deterministischen Ansatz basierend auf einem datenflussbasierten Zwischenmodell, dem Operationsmodell, aus welchem Prolog-Fakten und -Schnittstelle generiert werden (siehe Abschnitt 2.2).

Aufgrund dieses Unterschieds in der allgemeinen Herangehensweise an die Sicherheitsanalyse ist eine allgemeingültige Abbildung des Analysealgorithmus von UMLsec auf entsprechende Prolog-Abfragen nicht realistisch. Stattdessen wird die Intention der betrachteten Sicherheitsanalyse untersucht. Für diese existiert eine textuelle Beschreibung

in dem Buch von Jürjens [1, Kapitel 3 und 4]. Zusätzlich wird zur Verifikation des korrekten Verständnisses der Analyse eine der UMLsec-Referenzimplementationen eingesetzt, um verschiedene Eingabemodelle zu analysieren und die Ergebnisse mit der Erwartung zu vergleichen. Schließlich werden Prolog-Prädikate definiert, die eine Analyse in DCP umsetzen, die bezüglich ihrer Intention zu der UMLsec-Analyse äquivalent ist.

Das Operationsmodell entsteht durch eine Transformation aus dem DCP-Eingabemodell. Daher existiert hier eine Abhängigkeit zu der Erstellung der Modelltransformationen zwischen den Eingabemodellen von UMLsec und DCP. Um die konkrete Prolog-Abfrage zu definieren, werden daher Annahmen über die Existenz von Operationen, Eigenschaften und Struktur der Operationsmodellinstanz eines DCP-Eingabemodells getroffen. Diese Annahmen bilden anschließend die Basis für die Erstellung der Modelltransformationen. Bei dem aus diesem Schritt entstehenden Artefakt handelt es sich also um eine Menge von Prolog-Prädikate, die die Analyse darstellen, und eine Menge von Annahmen an die Operationsmodellinstanzen, auf denen die Analyse durchgeführt wird.

4.3. Modelltransformation

Die Erstellung der Modelltransformationen umfasst sowohl die Erstellung der Transformation des UMLsec-Eingabemodells in ein DCP-Eingabemodell, als auch die Erstellung einer Transformation für die umgekehrte Richtung, in der das DCP-Eingabemodell in ein UMLsec-Eingabemodell transformiert wird. Die Transformationsrichtung von UMLsec nach DCP ist offensichtlich für die Umsetzung der Analyse in DCP notwendig. Dagegen wird die Rückrichtung, von DCP nach UMLsec, primär für die Evaluation der DCP-Analyse umgesetzt.

Es ist zu erwähnen, dass es sich bei den beiden Transformationen formal nicht um eine Abbildung und ihre Umkehrabbildung handelt. Insbesondere in der Richtung von DCP nach UMLsec ist zu erwarten, dass ein Informationsverlust auftreten wird. Dies ist hauptsächlich dadurch bedingt, dass die Eingabemodelle für die UMLsec-Analysen häufig nur die notwendige Sicht für die Analyse darstellen, während die Eingabemodelle für die DCP-Analyse vollständige PCM-Modelle sind, welche um Datenfluss-Modelle erweitert wurden.

Die eigentliche Erstellung der Transformation erfolgt zunächst über die Erstellung einer Abbildung der Modellelemente aufeinander, die dieselbe Semantik haben. Dies ist naheliegender bei Eingabemodellen mit geringem Abstraktionsunterschied, während bei Eingabemodellen mit großem Abstraktionsunterschied eine stärkere Abweichung von der Semantik des UMLsec-Eingabemodells zu erwarten ist. Wie bereits beschrieben, muss die Transformation so gebaut werden, dass die Annahmen, die bei der Erstellung der DCP-Analyseformel getroffen wurden, erfüllt sind. Es existiert also eine starke Abhängigkeit von der definierten Prolog-Abfrage zu der Struktur der Operationsmodellinstanz und damit auch zu dem DCP-Eingabemodell. Es ist daher zu erwarten, dass mehrfach über die erstellte Analyseformel und die Modelltransformationen iteriert werden muss, bevor eine

Balance zwischen ausreichender Ähnlichkeit der Eingabemodelle, Erhalt der Intention der Analyse und technischer Umsetzbarkeit als Prolog-Abfrage an die bestehende Prolog-Schnittstelle gefunden wird.

Des Weiteren ist an dieser Stelle die Sonderstellung der Angreifermächtigkeit zu erwähnen. In UMLsec wird die Mächtigkeit des Angreifers als Teil des Eingabemodells definiert. Eine Änderung der Angreifermächtigkeit hat also immer eine Änderung einer Annotation des Eingabemodells zur Folge. In DCP dagegen sollen auf einem Operationsmodell verschiedene Analysen durchgeführt werden. Insbesondere können dabei die verschiedenen Analysen verschiedene Angreifermächtigkeiten voraussetzen. Aus diesem Grund ist es für die DCP-Sicherheitsanalyse sinnvoller, den Angreifer als vordefinierte Prolog-Abfrage zu modellieren, die zur Laufzeit abgefragt werden kann, wenn diese Analyse von einem Nutzer gewünscht wird. Dadurch hat eine Änderung der Angreifermächtigkeit keine Änderung am Eingabemodell und damit auch keine erneute Transformation in das Operationsmodell zur Folge.

Die folgenden Kapitel beschreiben jeweils eine betrachtete Analyse und folgen in der Struktur dem Schema, das hier beschrieben ist und erklären zusätzlich die jeweils betrachtete Analyse.

5. Secure Links

Bei Secure Links [1, Seite 59] handelt es sich um eine strukturelle Sicherheitsanalyse aus UMLsec, die überprüft, ob Sicherheitsanforderungen an die Abhängigkeiten von Softwareartefakten vom physischen Kommunikationspfad zwischen den Rechnerknoten, auf denen die Softwareartefakte ausgeführt werden, erfüllt sind. Im Folgenden wird zunächst eine Erklärung der UMLsec-Secure-Links-Analyse und der verwendeten Eingabemodelle gegeben. Anschließend sind die Gründe erläutert, die Secure Links geeignet für die Umsetzung in DCP machen. Darauf folgt die Beschreibung der Analyse in DCP. Zunächst werden dafür die Prolog-Abfragen erklärt, die den Kern des DCP-Analysealgorithmus darstellen. Abschließend wird die Modelltransformation zwischen den Eingabemodellen beschrieben.

Die Secure-Links-Sicherheitsanalyse erhält als Eingabemodell ein Verteilungsdiagramm, von dem bestimmte Elemente mit UMLsec-Stereotypen annotiert sind. Die minimale Form eines solchen Diagramms ist in Abbildung 5.1 dargestellt. Abhängigkeiten zwischen Artefakten werden um gewünschte Sicherheitseigenschaften annotiert. In dem Beispiel ist der Stereotyp «`secrecy`» dargestellt, der aussagt, dass jeglicher Datenaustausch zwischen den beiden Artefakten vertraulich stattfinden muss, also nicht von unautorisierten dritten Parteien gelesen werden darf. Kommunikationspfade sind mit ihrem physischen Typ annotiert. In Abbildung 5.1 ist dies durch den Stereotyp «`LAN`» dargestellt. Die Stärke des Angreifers bestimmt, welche Aktionen der Angreifer auf Kommunikationspfaden eines Typs ausführen kann. Der Standardangreifer ist nicht in der Lage mit Daten, die über einen Kommunikationspfad vom Typ «`LAN`» übertragen werden, zu interagieren. In diesem Beispiel liegt gemäß der Secure-Links-Analyse dementsprechend keine Sicherheitslücke vor. Ein stärkerer Angreifer hätte potentiell aber die Möglichkeit auf einen Kommunikationspfad des Typs «`LAN`» lesend zuzugreifen. In diesem Fall würde die Analyse dies erkennen und dem Nutzer den relevanten Kommunikationspfad mitteilen.

Im Folgenden wird die Analyse formal definiert. Dazu wird zunächst das Eingabemodell betrachtet. Das Eingabemodell besteht aus mindestens einem Subsystem, welches mit dem «`secure links`» Stereotyp annotiert ist. Dieser Stereotyp dient zum einen als Marker für die Subsysteme auf denen die Secure-Links-Analyse durchgeführt werden soll. Zum anderen enthält der Stereotyp eine Eigenschaftsdefinition namens `{adversary}` mit der die Angreifermächtigkeit definiert werden kann. Dazu wird ein Angreifer aus einer vordefinierten Liste von Angreifern ausgewählt.

Innerhalb des Subsystems sind die für die Analyse relevanten Modellelemente Rechnerknoten, Artefakte, Abhängigkeiten, Kommunikationspfade und Verteilungsbeziehungen.

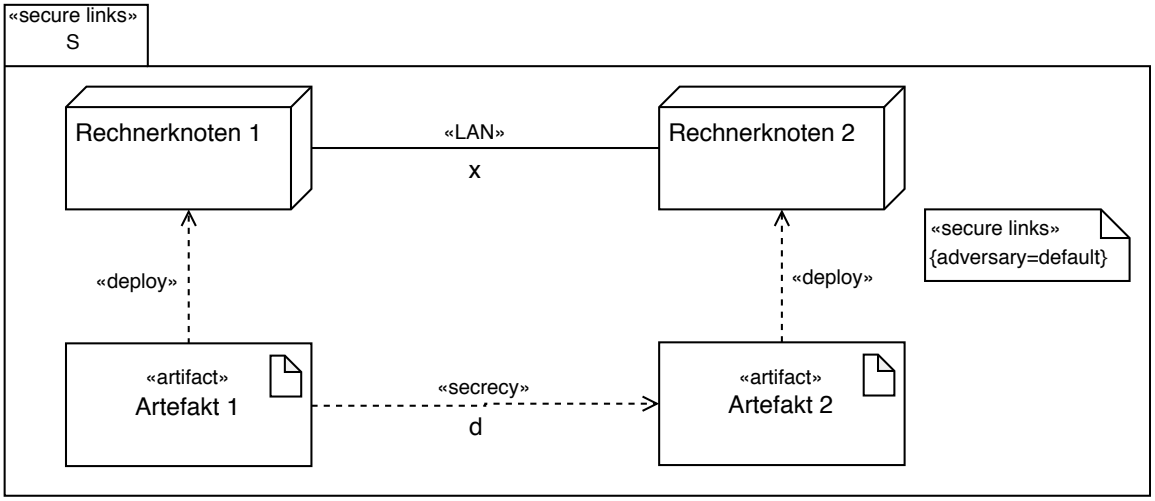

Abbildung 5.1.: Ein Secure-Links-Modell ohne Sicherheitslücke.

Dabei sind lediglich Kommunikationspfade und Abhängigkeiten mit Stereotypen erweitert. Für Kommunikationspfade sind die Stereotypen «`Internet`», «`LAN`», «`wire`» und «`encrypted`» definiert, die jeweils den Typ der physischen Verbindung zwischen den Rechnerknoten repräsentieren. Es ist gefordert, dass maximal einer dieser Stereotypen auf einen Kommunikationspfad angewandt wird.

Die relevanten Stereotypen für die Abhängigkeiten sind «`secrecy`», «`integrity`» und «`high`». Diese Stereotypen entsprechen den Bedeutungen von Vertraulichkeit, Integrität und hoher Sensitivität wie sie in Abschnitt 2.3 definiert sind. Durch den Stereotyp «`secrecy`» ist gefordert, dass Dritte keinerlei Informationen über die gesendeten Daten erhalten oder ableiten können. Dagegen fordert «`integrity`», dass die Daten nicht von Dritten modifiziert werden können. Der Stereotyp «`high`» stellt schließlich sowohl Integrität, als auch Vertraulichkeit der Daten sicher und repräsentiert Daten mit hoher Sensitivität. Im Gegensatz zu «`secrecy`» und «`integrity`» verwendet «`high`» jedoch die in Abschnitt 2.3 dargestellte alternative Definition über den Kommunikationsfluss. Daher handelt es sich hierbei um eine eigene zu betrachtende Eigenschaft und nicht nur um die Komposition von «`secrecy`» und «`integrity`». Anders als die Stereotypen, die auf Kommunikationspfade angewandt werden, können auf Abhängigkeiten mehrere der zuvor genannten Stereotypen auf einmal angewandt werden.

Gültige Eingabemodelle sind Eingabemodelle, die gemäß der UML-Spezifikation [19] gültig sind, für jede mit einem Stereotyp erweitere Abhängigkeit mindestens einen korrespondierenden Kommunikationspfad besitzen und den erwähnten Einschränkungen bezüglich der Stereotypanwendung genügen.

Die Angreifermächtigkeit eines Secure-Links-Angreifers wird über eine Funktion wie folgt definiert: Sei ein Angreifer vom Typ A gegeben, dann existiert eine Funktion $\text{Threats}_A(s)$, die einen auf einen Kommunikationspfad anwendbaren Stereotyp

$$s \in \{\text{«}\texttt{wire}\text{»}, \text{«}\texttt{encrypted}\text{»}, \text{«}\texttt{LAN, «Internet»}\text{»}\}$$

Tabelle 5.1.: Tabellarische Darstellung des Standardangreifers für die UMLsec-Secure-Links-Analyse. [1, Seite 57]

Stereotyp	$\text{Threats}_{default}()$
«Internet»	{delete, read, insert}
«encrypted»	{delete}
«LAN»	$\varnothing$
«wire»	$\varnothing$

auf eine Menge von Strings

$$\text{Threats}_A(s) \subseteq \{\texttt{delete}, \texttt{read}, \texttt{insert}\}$$

abbildet. Die Funktion $\text{Threats}_A(s)$ spezifiziert also, welche Aktionen ein Angreifer vom Typ A im Allgemeinen auf einem Kommunikationspfad vom Typ s ausführen kann. Die Aktionen `delete`, `read` und `insert` bedeuten, dass ein Angreifer auf einem Kommunikationspfad jeweils Nachrichten löschen, lesen oder einfügen kann.

Im Kontext von Secure Links werden zwei Angreifertypen definiert. Zum einen der Standardangreifer, der folgendermaßen definiert ist: Er ist in der Lage, auf einem Kommunikationspfad vom Typ «`Internet`» Nachrichten zu lesen, löschen oder hinzuzufügen. Auf einem verschlüsselten Kommunikationspfad, wie z. B. einem virtuellen privaten Netzwerk (VPN), besitzt dieser Angreifer die Fähigkeit Nachrichten zu löschen, aber kann keine Nachrichten lesen oder hinzufügen. Es wird angenommen, dass der Angreifer keine Möglichkeit besitzt, Zugriff auf den zur Verschlüsselung verwendeten Schlüssel zu bekommen. Der Angreifer kennt also nicht den Inhalt der Nachrichten, die er löscht, womit ein gezieltes Löschen von Nachrichten diesem Angreifer nicht möglich ist. Des Weiteren wird angenommen, dass der Standardangreifer keinen Zugriff auf ein lokales Netzwerk (LAN) hat und daher keinerlei Interaktionsmöglichkeit mit solchen Kommunikationspfaden besitzt. Auf Direktverbindungen, die sicherheitskritische Systeme miteinander verbinden («`wire`»), hat der Standardangreifer ebenfalls keinen Zugriff. Dieser Angreifer ist in Tabelle 5.1 dargestellt.

Der zweite Angreifertyp, der von Jürjens [1, Seite 58] definiert wird, ist der Insiderangreifer. Dieser verfügt über physischen Zugang zu den betrachteten Systemen und Netzwerken und kennt die zur Verschlüsselung verwendeten Schlüssel. Daher hat dieser Angreifertyp auf jedem Kommunikationspfad $s \in \{$«`wire`», «`encrypted`», «`LAN, «Internet»`»$\}$ jeweils die Aktionsmöglichkeiten `delete`, `read` und `insert`. Dieser Angreifer ist in Tabelle 5.2 dargestellt.

Gegeben dieser Angreiferdefinitionen wird im Folgenden die Analyse beschrieben. Dazu muss zunächst eine weitere Funktion eingeführt werden. Gegeben eines UML-Subsystems $\mathcal{S}$, kann aus $\text{Threats}_A(s)$ die Funktion $\text{threats}_A^{\mathcal{S}}(x)$ erzeugt werden. Diese Funktion erhält als Eingabe einen Kommunikationspfad x des Subsystems $\mathcal{S}$ und einen Angreifer vom Typ A und bildet diese auf eine Menge von Strings

$$\text{threats}_A^{\mathcal{S}}(x) \subseteq \{\texttt{delete}, \texttt{read}, \texttt{insert}\}$$

Tabelle 5.2.: Tabellarische Darstellung des Insiderangreifers für die UMLsec-Secure-Links-Analyse. [1, Seite 58]

Stereotyp	$\text{Threats}_{insider}()$
«Internet»	{delete, read, insert}
«encrypted»	{delete, read, insert}
«LAN»	{delete, read, insert}
«wire»	{delete, read, insert}

ab. Die Funktion $\text{threats}_A^{\mathcal{S}}(x)$ beschreibt also, ob ein Angreifer vom Typ A in Subsystem $\mathcal{S}$ eine Aktion auf dem gegebenen Kommunikationspfad x ausführen kann.

In dem Buch von Jürjens [1] ist $\text{threats}_A^{\mathcal{S}}(x)$ nicht formal definiert. Zur Vollständigkeit und zum besseren Verständnis wird die Funktion im Folgenden als

$$\text{threats}_A^{\mathcal{S}}(x) = \begin{cases} \text{Threats}_A(s), & \text{wenn } x \text{ in } \mathcal{S} \text{ um einen Stereotyp } s \text{ erweitert} \\ & \text{ist, für den gilt, dass} \\ & s \in \{\text{«wire»}, \text{«encrypted»}, \text{«LAN, «Internet»»}\} \\ \varnothing, & \text{sonst} \end{cases}$$

definiert, was der Beschreibung von Jürjens [1, Seite 39] und den Ergebnissen der Analyse des CARiSMA-Werkzeugs [41] entspricht.

Die Analyse überprüft, ob für jedes Subsystem $\mathcal{S}$, auf das der Stereotyp «secure links» angewandt ist, für jede Abhängigkeit d mit Stereotyp $s \in \{\text{«secrecy»}, \text{«integrity»}, \text{«high»}\}$ zwischen Subsystemen oder Objekten auf verschiedenen Rechnerknoten n, m ein Kommunikationspfad l zwischen n und m existiert, sodass

- wenn $s = \text{«high»}$, dann $\text{threats}_A^{\mathcal{S}}(l) = \varnothing$,
- wenn $s = \text{«secrecy»}$, dann $\texttt{read} \notin \text{threats}_A^{\mathcal{S}}(l)$ und
- wenn $s = \text{«integrity»}$, dann $\texttt{insert} \notin \text{threats}_A^{\mathcal{S}}(l)$

gelten. Nachfolgend ist eine beispielhafte Überprüfung dieser Bedingungen anhand des Modells, das in Abbildung 5.1 dargestellt ist, durchgeführt. Es existiert eine Abhängigkeit d, die mit «secrecy» annotiert ist und ein korrespondierender Kommunikationspfad x, der mit «LAN» annotiert ist. Gegeben des Standardangreifers gilt $\text{threats}^{\mathsf{S}}_{\texttt{default}}(\mathsf{x}) = \varnothing$. Gemäß der Secure-Links-Sicherheitsanalyse liegt also keine Sicherheitslücke vor. Angenommen die Angreifermächtigkeit sei insider. Dann gilt $\text{threats}^{\mathsf{S}}_{\texttt{insider}}(\mathsf{x}) = \{\texttt{delete}, \texttt{read}, \texttt{insert}\}$. Da aber für den Stereotyp «secrecy» von d $\texttt{read} \notin \text{threats}^{\mathsf{S}}_{\texttt{insider}}(\mathsf{x})$ gefordert ist, liegt in diesem Fall eine Sicherheitslücke vor. Die Secure-Links-Analyse wird demnach mit der Begründung fehlschlagen, dass es dem Insiderangreifer möglich ist, von Kommunikationspfad d zu lesen.

Nachfolgend werden die Gründe für die Auswahl von Secure Links, die Analyse in DCP in Form von Prolog-Abfragen und die Modelltransformation zwischen den Eingabemodellen vorgestellt. Dabei wird ein Beispiel verwendet, an dem interessante Unterschiede zwischen

der Analyse und den Eingabemodellen aus DCP und UMLsec aufgezeigt werden können. Dieses Beispiel ist in Abbildung 5.2 dargestellt und wird nun erklärt. Es besteht aus drei Artefakten und drei Rechnerknoten. Dabei wird Artefakt `A1` sowohl auf Rechnerknoten `N1` und Rechnerknoten `N2` installiert, während Artefakt `A2` nur auf Rechnerknoten `N2` und Artefakt `A3` nur auf Rechnerknoten `N3` installiert werden. Artefakt `A1` hat eine Abhängigkeit `D_A1_A3` zu Artefakt `A3`, welches wiederum eine Abhängigkeit `D_A3_A2` zu Artefakt `A2` hat. Die Abhängigkeit `D_A1_A3` fordert die Behandlung der Daten als Daten mit hoher Sensitivität («`high`»), während die Abhängigkeit `D_A3_A2` lediglich Vertraulichkeit fordert («`secrecy`»). Der Kommunikationspfad `CP_N1_N3` zwischen Rechnerknoten `N1` und Rechnerknoten `N3` ist als lokales Netzwerk definiert («`LAN`») und der Kommunikationspfad `CP_N2_N3` zwischen Rechnerknoten `N2` und Rechnerkonten `N3` als verschlüsseltes Netzwerk («`encrypted`»). Bei dem betrachteten Angreifer handelt es sich um den Standardangreifer.

Während der Analyse werden mehrere separate Bedingungen überprüft. Zunächst wird Abhängigkeit `D_A1_A3` betrachtet. Es existieren hierfür zwei mögliche Verteilungen der Artefakte auf die Rechnerknoten: Die Installation von Artefakt `A1` auf Rechnerknoten `N1` und die Installation von Artefakt `A1` auf Rechnerknoten `N2`. Artefakt `A3` wird immer auf Rechnerknoten `N3` installiert. Es muss also überprüft werden, ob Kommunikationspfad `CP_N1_N3` der geforderten Sicherheitseigenschaft unter dem Standardangreifer entspricht. Das gleiche muss für Kommunikationspfad `CP_N2_N3` gelten. Anschließend wird die Abhängigkeit `D_A3_A2` untersucht. Hierfür existiert nur eine Verteilung der Artefakte: Die Installation von Artefakt `A3` auf Rechnerknoten `N3` und die Installation von Artefakt `A2` auf Rechnerknoten `N2`. Demnach wird hier Kommunikationspfad `CP_N2_N3` überprüft. Formal müssen also folgende Bedingungen überprüft werden:

1. $\text{threats}^{\mathcal{S}}_{\texttt{default}}(\texttt{CP\_N1\_N3}) = \varnothing$ (da der Stereotyp von `D_A1_A3` = «`high`»)
2. $\text{threats}^{\mathcal{S}}_{\texttt{default}}(\texttt{CP\_N2\_N3}) = \varnothing$ (da der Stereotyp von `D_A1_A3` = «`high`»)
3. $\texttt{read} \notin \text{threats}^{\mathcal{S}}_{\texttt{default}}(\texttt{CP\_N2\_N3})$ (da der Stereotyp von `D_A3_A2` = «`secrecy`»)

Da der Standardangreifer in der Lage ist, von einem verschlüsselten Kommunikationspfad Nachrichten zu löschen, gilt $\text{threats}^{\mathcal{S}}_{\texttt{default}}(\texttt{CP\_N2\_N3}) = \{\texttt{delete}\} \neq \varnothing$. Es liegt also eine Sicherheitslücke an Kommunikationspfad `CP_N2_N3` vor, da der Standardangreifer hier Nachrichten löschen kann, die mit hoher Sensitivität deklariert sind.

Es ist ersichtlich, dass die Secure-Links-Analyse die Komposition der einzelnen überprüften Abhängigkeiten und korrespondierenden Kommunikationspfade ist. Das bedeutet, dass das in Abbildung 5.2 gegebene Eingabemodell in drei separate Modelle transformiert werden kann, ohne dass das Ergebnis der Secure-Links-Analyse verändert wird (siehe Abbildung 5.3). Dies ist eine hilfreiche Eigenschaft, die bei der Definition der Datenflüsse im Operationsmodell für die Analyse in DCP verwendet wird.

Abschließend sind Szenarien zu erwähnen, in denen die von Jürjens [1, Seite 59] beschriebene Secure-Link-Analyse von dem Ergebnis des CARiSMA-Werkzeugs [41] abweicht. Per Definition werden in der Analyse nur Abhängigkeiten betrachtet, die um einen Stereotyp $s \in \{\text{«}\texttt{secrecy}\text{»}, \text{«}\texttt{integrity}\text{»}, \text{«}\texttt{high}\text{»}\}$ erweitert und zwischen Subsystemen oder

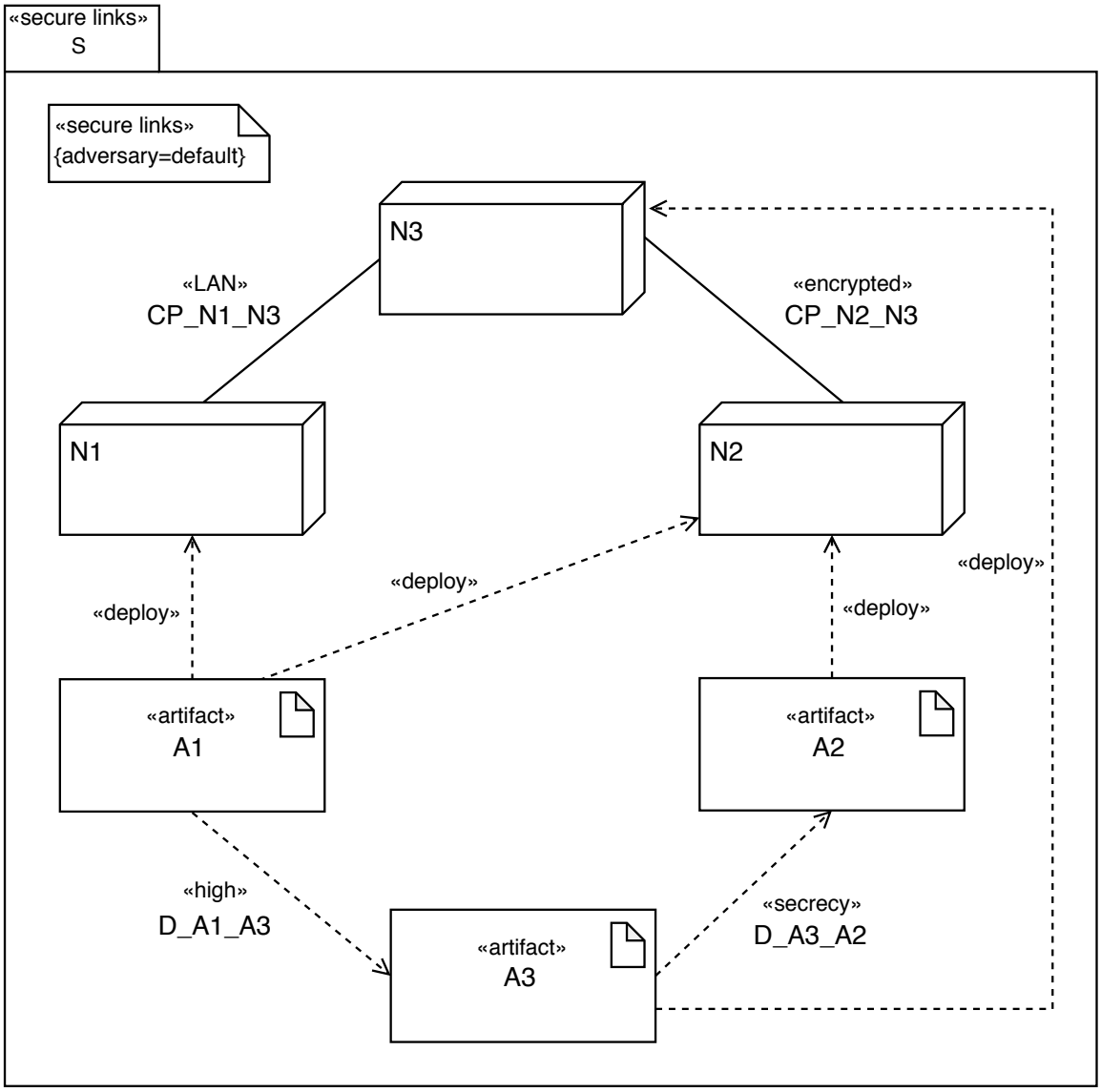

Abbildung 5.2.: Ein komplexeres Secure-Links-Modell, an dem Eigenschaften der Analyse gezeigt werden.

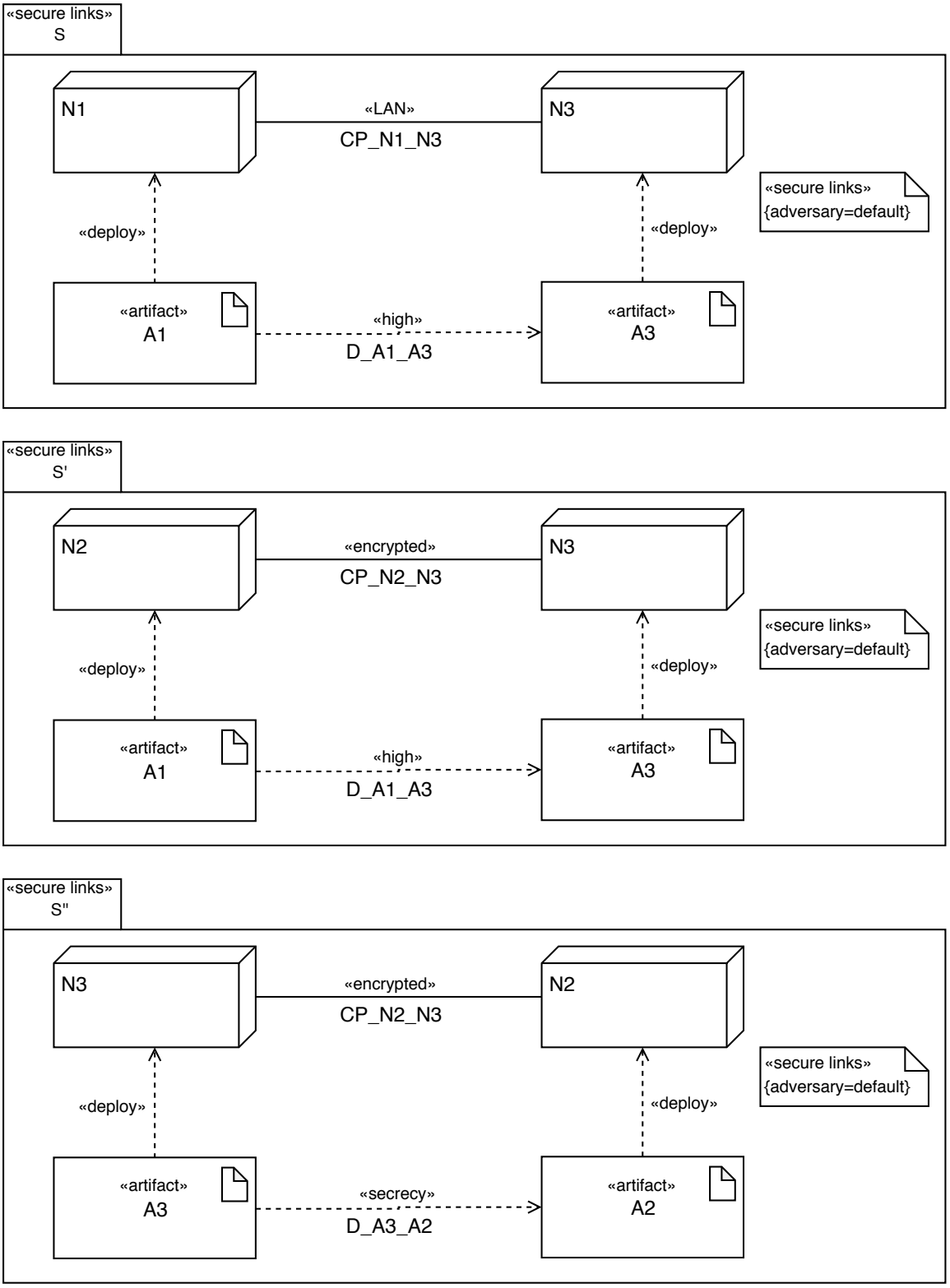

Abbildung 5.3.: Das Secure-Links-Modell aus Abbildung 5.2 unterteilt in drei separate Secure-Links-Modelle, sodass die Analyse weiterhin dasselbe Ergebnis liefert.

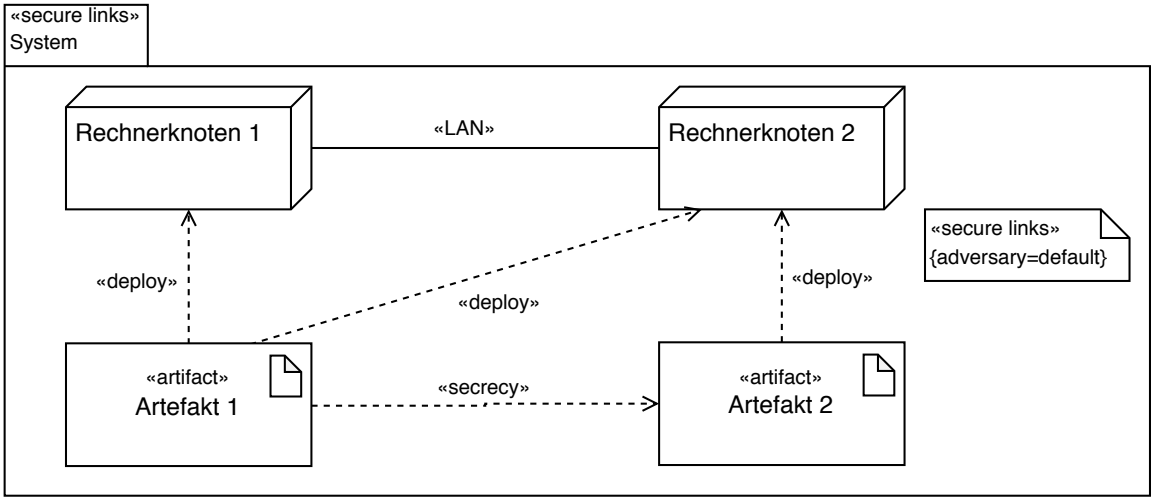

Abbildung 5.4.: Ein UMLsec-Modell, das im Kontext der Secure-Links-Sicherheitsanalyse keine Sicherheitslücke besitzt. Der Standardangreifer kann den Kommunikationspfad vom Typ «LAN» nicht angreifen und hat per Secure-Links-Angreiferdefinition auch keinen Zugriff auf die Rechnerknoten.

Objekten auf unterschiedlichen Rechnerknoten sind. Eine Abhängigkeit zwischen zwei Artefakten, die auf denselben Rechnerknoten installiert sind, sollte daher von der Secure-Links-Analyse nicht beachtet werden (siehe Abbildung 5.4). Dennoch wird in diesem Fall von dem CARiSMA-Werkzeug ein Fehler ausgegeben. Der Fehler sagt nichts über eine Sicherheitslücke aus, sondern lediglich, dass ein erwarteter Kommunikationspfad, der den Rechnerknoten mit sich selbst verbindet, fehlt. Im Folgenden wird davon ausgegangen, dass es sich dabei um eine Einschränkung der Implementation handelt, die nicht die Intention der Analyse ist.

Eine weitere Abweichung der implementierten Analyse von den Definitionen des Buches stellt der Insiderangreifer dar. Gemäß der Definition des Buches [1, Seite 58] kann dieser Angreifer die Aktionen `read`, `insert` und `delete` auf allen definierten Typen von Kommunikationspfaden («`Internet`», «`encrypted`», «`LAN`» und «`wire`») ausführen. Jedoch kann der Angreifer gemäß der verwendeten Definition des CARiSMA-Werkzeugs keine Aktionen gegen einen Kommunikationspfad des Typs «`wire`» durchführen. Die Definition des Angreifers ist in Tabelle 5.3 dargestellt. Im Kontext der Secure-Links-Analyse ist dieser Angreifer interessanter als der Insiderangreifer aus [1, Seite 58], da der Insiderangreifer nach der Definition des Buchs jede Angreiferaktion auf jedem Kommunikationspfadtyp ausführen kann und damit, sobald eine Sicherheitseigenschaft gefordert ist, immer zu einem Fehlschlag der Secure-Links-Analyse führt. Aus diesem Grund ist im Folgenden mit Insiderangreifer stets der Angreifer gemäß der Definition des CARiSMA-Werkzeugs gemeint (Tabelle 5.3).

Tabelle 5.3.: Tabellarische Darstellung des Insiderangreifers für die UMLsec-Secure-Links-Analyse gemäß der Definition des CARiSMA-Werkzeugs. [41]

Stereotyp	$\text{Threats}_{insider}()$
«Internet»	{delete, read, insert}
«encrypted»	{delete, read, insert}
«LAN»	{delete, read, insert}
«wire»	$\varnothing$

5.1. Auswahlkriterien

Unter der Betrachtung der notwendigen Kriterien für die Auswahl der Secure-Links-Analyse ist festzustellen, dass sowohl eine Referenzimplementation der Analyse in Form des CARiSMA-Werkzeugs existiert, als auch eine ausreichend detaillierte Beschreibung der Analyse in dem Buch von Jürjens [1, Seite 59] gegeben ist. Auch wenn diese Beschreibung nicht alle gewünschten Ergebnisse für Randfälle erläutert, ist es in Kombination mit dem CARiSMA-Werkzeug möglich, diese zu ermitteln. Bei den Randfällen, für die eine Betrachtung der Ergebnisse im CARiSMA-Werkzeug notwendig ist, handelt es sich zum einen um strukturelle Unterschiede zu dem gegebenen Trivialbeispiel, wenn z. B. ein Artefakt auf mehrere Rechnerknoten installiert wird. Zum anderen muss auch die Funktionsweise der Analyse, wenn mehrere UMLsec-Stereotypen der Sicherheitseigenschaften («secrecy», «integrity» oder «high») auf dieselbe Abhängigkeit angewandt werden, mittels des CARiSMA-Werkzeugs untersucht werden. Das Buch und das CARiSMA-Werkzeug liefern zusammen also ausreichend Informationen, um den Auswahlkriterien 1 und 2 zu genügen.

Abschließend ist der Anwenderfokus der Analyse betrachtet (Auswahlkriterium 3). Es kann davon ausgegangen werden, dass ein Softwareentwickler in der Regel mit dem Konzept von Verteilungsdiagrammen vertraut ist. Der Typ des verwendeten Kommunikationspfads zwischen Rechnerknoten ist entweder intuitiv zu erfassen («Internet» oder «LAN») oder braucht zum Verständnis nur eine kurze Erklärung, die kein kryptographisches Fachwissen voraussetzt («wire» oder «encrypted»). Bei den Sicherheitseigenschaften handelt es sich um die Konzepte, die für Entwickler ohne bisherige Vorkenntnisse in der Modellierung von Sicherheitsszenarien die ausführlichsten Erklärungen und Definitionen benötigen. Während eine genaue Betrachtung der Anwenderfreundlichkeit außerhalb des Bearbeitungsumfangs dieser Arbeit ist, wird im Folgenden davon ausgegangen, dass die textuellen Erklärungen von secrecy, integrity und high ausreichend sind, um ein sinnvolles Secure-Links-Modell zu erstellen. Unter dieser Annahme handelt es sich bei Secure Links um eine einfache Analyse, deren Eingabemodell auch von Anwendern ohne tiefgehende kryptographische Vorkenntnisse erstellt und damit genutzt werden kann.

5.2. DCP-Analyse

Die Sicherheitsanalyse in DCP wird als Abfrage an Prolog-Regeln formuliert, die aus einem Operationsmodell generiert werden, welches wiederum durch eine Transformation aus dem DCP-Eingabemodell entsteht. Im Folgenden wird die Prolog-Abfrage vorgestellt, die Sicherheitsverletzungen nach Definition der Secure-Links-Bedingungen erkennt. Da die Struktur und Aufrufhierarchien des Operationsmodells (siehe Abschnitt 2.2) abgefragt werden, muss zunächst die erwartete Operationsmodellinstanz betrachtet werden.

Für die Analyse der von Secure Links definierten Bedingungen sind nur Datenflüsse relevant, die von einem Rechnerknoten zu einem anderen Rechnerknoten fließen. Die grundlegende Idee der Analyse ist es, die Netzwerkressourcen, die für die Analyse relevant sind, als Operation im Operationsmodell abzubilden. Dadurch können alle Operationen generiert werden, die eine Netzwerkressource repräsentieren. Dies wird durch eine Eigenschaft vom Typ *Link* mit einem Wert $l \in \{Internet, encrypted, LAN, wire\}$ dargestellt. Für eine solche Operation wird anschließend geprüft, ob sie über Eingabeparameter oder Rückgabevariablen verfügt, welche ein Attribut vom Typ *Sensitivity* besitzen und für die mindestens einer der Werte {*Secrecy*, *Integrity*, *High*} gesetzt ist. Abschließend kann überprüft werden, ob ein Angreifer existiert, sodass für eine Operation mit Eigenschaftswert l und Eingabeparametern oder Rückgabevariablen mit Attributswerten $s \in \{Secrecy, Integrity, High\}$ die in Secure Links definierten Bedingungen der relevanten Sicherheitseigenschaften s verletzt sind.

Ein Beispiel für eine Operation, die als Repräsentant für eine Netzwerkressource dient, ist in Abbildung 5.5 dargestellt. Die Operation *PerformDataTransmissionOperation* (bzw. *ReturnDataOperation*) leitet sich aus einem Datenverarbeitungsvorgang des Typs *PerformDataTransmission* (bzw. *ReturnData*) des Eingabemodells ab. Diese Operationen beschreiben Teile des Aufrufverhaltens von Komponenteninstanzen und können daher zu Rechnerknoten zurückverfolgt werden. Dies macht sie ungeeignet als Repräsentant für die Netzwerkressource. Die Operation *DataTransmissionProxyOperation* ist eine in dieser Arbeit entstandene Erweiterung der Implementation der Transformation des Eingabemodells zum Operationsmodell. Sie agiert als Stellvertreter für das eigentliche Ziel der Operation *PerformDataTransmissionOperation* und leitet daher den Eingabeparameter a unverändert zu der Operation *ReturnDataOperation* weiter. Analog dazu wird die Rückgabevariable b ebenfalls unverändert weitergeleitet. Diese Operation dient als Repräsentant für die Netzwerkressource und verfügt über die entsprechenden Eigenschaftswerte, die sich aus den Charakteristiken der Netzwerkressource ableiten lassen.

Der UMLsec-Stereotyp, der die Abhängigkeit zwischen zwei Artefakten annotiert und die geforderten Sicherheitseigenschaften spezifiziert, wird in DCP als Charakteristik an den Daten des Datenflussmodells modelliert. Dies ist deshalb sinnvoll, da es erlaubt, die geforderten Sicherheitseigenschaften feingranularer zu betrachten. Anstatt z. B. zu spezifizieren, dass jegliche Kommunikation zwischen zwei Komponenten vertraulich ablaufen soll, können gezielt nur die Daten als vertraulich markiert werden, für die es wirklich notwendig ist. Ein daraus resultierender Vorteil ist die separate Betrachtung der Sicherheitseigenschaften von Aufrufparametern und Rückgabevariablen. In der UMLsec-

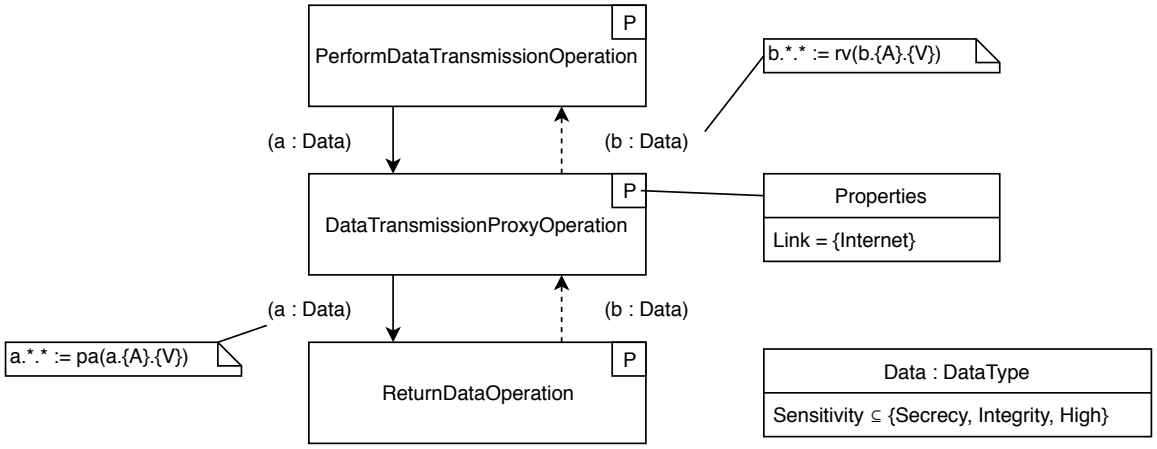

Abbildung 5.5.: Ein Beispiel für eine Operation, die als Repräsentant für eine Netzwerkressource vom Typ Internet dient.

Secure-Links-Analyse ist immer gefordert, dass jegliche Kommunikation der spezifizierten Sicherheitseigenschaft entsprechen muss. Es gibt jedoch viele Fälle, in denen Parameter und Rückgabevariablen nicht dieselben Sicherheitseigenschaften erfüllen müssen. Ein Beispiel hierfür wäre ein Server, der einen Service zur sicheren Generation von Zufallswerten aus einem physikalischen Zufallszahlengenerator zur Verfügung stellt. In diesem Fall sind die Aufrufparameter nicht vertraulich, da selbst ein weiterer Aufruf mit denselben Parametern einen anderen Rückgabewert zurückliefert. Je nach Verwendungszweck der Zufallszahl ist die Rückgabevariable aber als vertraulich einzustufen. Für dieses Szenario muss in der Modellierung nach UMLsec eine Überabschätzung getroffen werden und in beide Richtungen Vertraulichkeit gefordert werden, während in DCP die Charakteristiken der Eingabe- und Rückgabedaten getrennt voneinander gesetzt werden können.

Es entstehen daher vier Möglichkeiten, die Modelle der UMLsec-Analyse in DCP umzusetzen.

1. Die Sicherheitseigenschaften sind sowohl für Eingabeparameter als auch Rückgabevariablen gefordert.
2. Die Sicherheitseigenschaften sind nur für Eingabeparameter gefordert.
3. Die Sicherheitseigenschaften sind nur für Rückgabevariablen gefordert.
4. Die Sicherheitseigenschaften sind weder für Eingabeparameter noch für Rückgabevariablen gefordert.

Option 4 ist offensichtlich nicht sinnvoll, um die Semantik der Analyse zu erhalten und hier nur zur Vollständigkeit aufgeführt. Für die Transformation von UMLsec-Eingabemodellen in DCP-Eingabemodelle wird Option 1 verwendet, da diese am ehesten der Semantik der UMLsec-Analyse entspricht. Option 2 und 3 sind dennoch in einigen Szenarien sinnvoll und es ist zu erwähnen, dass diese Optionen ohne Änderung an der Implementation der DCP-Analyse modelliert werden können.

Die Mächtigkeit des Angreifers wird direkt in den Prolog-Regeln definiert. Im Folgenden werden die eingeführten Prolog-Prädikate, die die Secure-Links-Sicherheitsanalyse bilden,

```
1 secureLinks(ATT, OP, D, ACT, L, DS) :-
2     hasLinkWithDataSensitivity(OP, L, DS, D),
3     secureLinksViolation(ATT, ACT, L, DS).
```

Algorithmus 5.1.: Die Definition des Prolog-Prädikats *secureLinks\6*.

erklärt. Den Einsprungpunkt in die Analyse stellt das Prädikat *secureLinks\6* dar. Es ist in Algorithmus 5.1 definiert. Dieses Prädikat wird als Generator verwendet und liefert eine Parameterbelegung für jede im Modell vorhandene Sicherheitsverletzung. Alle Parameter dienen dazu, den Nutzer über den Grund des Fehlschlags der Analyse zu informieren. Die Variable *ATT* ist die Bezeichnung des Angreifers. Sie kann sowohl explizit vom Aufrufer gesetzt werden, um die Analyse auf einen Angreifer einzuschränken, oder als freie Variable belassen werden, sodass die Analyse für alle definierten Angreifer durchgeführt wird. *OP* beschreibt die Operation, an der die Sicherheitsverletzung aufgetreten ist. Da es sich bei dieser Operation, gemäß der beschriebenen Struktur des Operationsmodells, um eine *DataTransmissionProxyOperation* handelt, kann sie auf die dazugehörige Netzwerkressource abgebildet werden. Der Parameter *D* beschreibt, ob es sich bei der Variable, die für die Sicherheitsverletzung verantwortlich ist, um einen Aufrufparameter oder eine Rückgabevariable handelt. Die Angreiferaktionen, die dem Angreifer an einer Sicherheitsverletzung zur Verfügung stehen, sind in dem Parameter *ACT* definiert. *L* liefert die *Link*-Eigenschaft der Operation und damit den physischen Typ der Netzwerkressource, z. B. *Internet*. *DS* liefert das *Sensitivity*-Attribut des Datums, z. B. *Secrecy*.

Die Parameter *ATT*, *OP* und *ACT* dienen dazu äquivalente Informationen zu der Implementation der UMLsec-Secure-Links-Analyse auszugeben. Die Datenflussrichtung *D* ist notwendig, da es dieses Konzept in UMLsec nicht gibt, aber in DCP hier unterschieden werden muss. Die beiden Parameter *L* und *DS* sind nur Hilfestellungen zur Validierung und Evaluation.

Das Prädikat *secureLinks\6* delegiert an zwei weitere Prolog-Prädikate. Zunächst wird das Prädikat *hasLinkWithDataSensitivity\4* abgefragt, das alle Operationen generiert, die über die *Link*-Eigenschaft verfügen und über die Daten fließen, die mit dem *Sensitivity*-Attribut annotiert sind. Das Ergebnis dieser Abfrage sind also alle Operationen, die für die Secure-Links-Analyse relevant sind. Anschließend wird das Prädikat *secureLinksViolation\4* abgefragt. Es generiert alle Parameterbelegungen, in denen das Ausführen einer der dem Angreifer verfügbaren Aktionen *ACT* auf einer Netzwerkressource vom Typ *L* eine Sicherheitsverletzung für das betrachtete Modell darstellt.

Das Prädikat *hasLinkWithDataSensitivity\4* ist in Algorithmus 5.2 dargestellt. Dieses Prädikat ist der Generator, der alle Parameterbelegungen für Operationen mit Eigenschaftswert *L* und Eingabeparametern oder Rückgabevariablen mit Attributswert *DS* erzeugt. Dies ist das einzige Prädikat der Analyse, das die aus dem Operationsmodell erzeugte Prolog-Schnittstelle direkt verwendet. In Zeile 2 und 3 wird *OP* eine Operation zugewiesen, die eine Eigenschaft vom Typ *Link* mit dem Wert *L* hat. Zeile 4 und 5 weisen *S* die Aufrufhierarchie zu und stellen sicher, dass es sich um eine valide Aufrufhierarchie handelt.

```
1  hasLinkWithDataSensitivity(OP, L, DS, D) :-
2      isOperation(OP),
3      operationProperty(OP, 'Link', L),
4      S=[OP|_],
5      stackValid(S),
6      operationState(OP, ST),
7      D = 'CallParameter',
8      preCallState(S, OP, ST, 'Sensitivity', DS).
9
10 hasLinkWithDataSensitivity(OP, L, DS, D) :-
11     isOperation(OP),
12     operationProperty(OP, 'Link', L),
13     S=[OP|_],
14     stackValid(S),
15     operationReturnValue(OP, R),
16     D = 'ReturnValue',
17     returnValue(S, R, 'Sensitivity', DS).
```

Algorithmus 5.2.: Die Definition des Prolog-Prädikats *hasLinkWithDataSensitivity\4*.

Anschließend wird in Zeile 6 *ST* eine der Zustandsvariablen von *OP* zugewiesen. Die Zuweisung von *D* in Zeile 7 ist nur für die Ausgabe an den Nutzer relevant. Schließlich wird in Zeile 8 überprüft, ob unter der Aufrufhierarchie *S*, *DS* wahr ist. Dabei ist *DS* der Wert des Attributs *Sensitivity* der Zustandsvariable *ST* der Operation *OP*, bevor der Aufruf zu der Operation an der Spitze der Aufrufhierarchie durchgeführt wird. Anders ausgedrückt wird damit überprüft, dass vor dem Aufruf der nächsten Operation der zu übergebende Eingabeparameter den Attributswert *DS* besitzt. Die Definition der zweiten Regel (Zeile 10) erfolgt analog und gilt für die Rückgabevariablen. Daher werden in Zeile 15 die Rückgabevariablen zugewiesen und nicht die Zustandsvariablen. In Zeile 17 wird überprüft, ob die Rückgabevariable der aktuellen Operation den Attributswert *DS* für das Attribut *Sensitivity* gesetzt hat.

Das Prädikat *secureLinksViolation\4* ist der Kern der Analyse und gegeben durch drei Bedingungen, die den bereits vorgestellten Analysebedingungen der Secure-Links-Analyse entsprechen. Es ist in Algorithmus 5.3 dargestellt. Zur Überprüfung der Vertraulichkeitsbedingung (Zeile 1) gegenüber eines Angreifers *ATT* und einem *Link*-Eigenschaftswert *L* werden die Angreiferaktionen *ACT* des Angreifers *ATT*, die er auf einer Netzwerkressource vom Typ *L* ausführen darf, generiert. Dies passiert durch die Abfrage an das Prädikat *attacker\3* (Zeile 3). Anschließend wird überprüft ob die Aktion `read` in der Menge der für den Angreifer *ATT* möglichen Aktionen *ACT* enthalten ist (Zeile 5). Die Überprüfung der Integritätsbedingung (Zeile 7) und der Bedingung der hohen Sensitivität (Zeile 13) erfolgt analog. Für die Integrität wird in Zeile 11 überprüft, ob der Angreifer die Aktion `insert` auf der Netzwerkressource von Typ *L* ausführen darf. Dagegen wird für Daten

```
1  secureLinksViolation(ATT, ACT, L, DS) :-
2      DS = 'Secrecy',
3      attacker(ATT, ACT, L),
4      is_list(ACT),
5      member('read', ACT).
6
7  secureLinksViolation(ATT, ACT, L, DS) :-
8      DS = 'Integrity',
9      attacker(ATT, ACT, L),
10     is_list(ACT),
11     member('insert', ACT).
12
13 secureLinksViolation(ATT, ACT, L, DS) :-
14     DS = 'High',
15     attacker(ATT, ACT, L),
16     is_list(ACT),
17     length(ACT, X),
18     X > 0.
```

Algorithmus 5.3.: Die Definition des Prolog-Prädikats *secureLinksViolation\4*.

```
1  attacker(
2      'default',
3      ['read', 'insert', 'delete'],
4      'Internet'
5  ).
6  attacker('default', ['delete'], 'Encrypted').
```

Algorithmus 5.4.: Die Definition des Prolog-Prädikats *attacker\3* für den Standardangreifer.

hoher Sensitivität überprüft, ob die Menge der Angreiferaktionen leer ist (Zeile 17 und 18).

Das Prädikat *attacker\3* ist durch Fakten gegeben, die der Angreiferdefinition aus UMLsec entsprechen. Jeder Fakt beschreibt einen Angreifer mit dem Bezeichner *ATT* und die Liste von möglichen Angreiferaktionen *ACT*, die er auf einer Netzwerkressource von Typ *L* ausführen darf. Dieses Prädikat ist in Algorithmus 5.4 dargestellt. Beispielsweise werden für die Überprüfung der Vertraulichkeitsbedingung des Prädikats *secureLinksViolation\4* für den Standardangreifer zwei Parameterbelegungen für *ACT* und *L* generiert. Einmal die Variablenbelegung $ACT = \{\texttt{read}, \texttt{insert}, \texttt{delete}\}$ und $L = \textit{Internet}$ und einmal $ACT = \{\texttt{delete}\}$ und $L = \textit{Encrypted}$. Im ersten Fall ist es möglich, dass eine Sicherheitsverletzung vorliegt. Im zweiten Fall ist die Abfrage `member('read', ACT)` falsch, daher kann unter dem Standardangreifer keine Verletzung der Vertraulichkeit an Netzwerkressourcen des Typs *Encrypted* vorkommen.

Der Ablauf der Analyse wird im Folgenden anhand des in Abbildung 5.2 gegebenen Beispiels dargestellt. Hierfür wird der Kürze halber nur ein vereinfachter Auszug aus dem korrespondierenden Operationsmodell verwendet. Das vollständige Operationsmodell und die Eingabemodelle sind als Datensatz verfügbar [42]. Die Betrachtung erfolgt unter dem Standardangreifer. Das vereinfachte Operationsmodell ist in Abbildung 5.6 dargestellt. Die relevanten Aufrufhierarchien beginnen jeweils bei den beiden Operationen *A1_N1_to_A3* und *A1_N2_to_A3*. Diese Operationen korrespondieren zu der Abhängigkeit *D_A1_A3* und erzeugen daher jeweils die Daten *a* und *b* mit dem Attributswert *High* auf wahr und alle anderen Attributswerte des Attributs *Sensitivity* auf falsch gesetzt. Anschließend werden die Stellvertreteroperationen *A1_N1_to_A3_Proxy* und *A1_N2_to_A3_Proxy* aufgerufen. Diese korrespondieren jeweils zu den Kommunikationspfaden *CP_N1_N3* und *CP_N2_N3*. Daher verfügt die Operation *A1_N1_to_A3_Proxy* über den Eigenschaftswert *LAN* und *A1_N2_to_A3_Proxy* über den Eigenschaftswert *Encrypted*. Die von der Operation *A3_to_A2* erzeugten Daten korrespondieren zu der Abhängigkeit *D_A3_A2* und werden daher mit dem Attributswert *Secrecy* initialisiert. Abschließend ist *A3_to_A2_Proxy* ebenfalls ein Repräsentant für den Kommunikationspfad *CP_N2_N3* und verfügt daher über den Eigenschaftswert *Encrypted*.

Für den Standardangreifer läuft die Analyse folgendermaßen ab. Zunächst sei die Variable $ATT = default$ durch den Nutzer gebunden. Die erste Abfrage des Prädikats *secureLinks\6* generiert eine Parameterbelegung so, dass *OP* eine Operation ist, die über den *Link*-Eigenschaftswert *L* verfügt und über die Daten mit *Sensitivity*-Attribut *DS* fließen. In dem gegebenen Beispiel wäre eine mögliche Parameterbelegung beispielsweise $OP = A1\_N2\_to\_A3\_Proxy$, $L = Encrypted$, $DS = High$ und $D = CallParameter$. Mit diesen Parametern gebunden wird dann das Prädikat *secureLinksViolation\4* aufgerufen. Da $DS = High$ gilt, werten die ersten beiden Prolog-Regeln des Prädikats zu falsch aus. Erst in der dritten Regel werden durch das Prädikat *attacker\3* die möglichen Angreiferaktionen des Standardangreifers ermittelt. Da $L = Encrypted$ gilt, sind die generierten Angreiferaktionen $ACT =$ `delete`. Da damit die Menge *ACT* nicht leer ist, ist eine Parameterbelegung gefunden und wird dem Nutzer ausgegeben. Die Überprüfung der Rückgabevariable, also für $D = ReturnValue$, erfolgt anschließend und analog. Abschließend wird gleichermaßen für die Operationen *A1_N1_to_A3_Proxy* und *A3_to_A2_Proxy* verfahren. Für *A1_N1_to_A3_Proxy* wird keine Parameterbelegung gefunden, da der Standardangreifer über keine Angreiferaktionen verfügt, die er auf einer Netzwerkressource vom Typ *LAN* durchführen darf. Da die Operation *A3_to_A2_Proxy* über den Eigenschaftswert *Encrypted* verfügt, darf der Angreifer zwar hier die `delete`-Aktion anwenden, aber da in diesem Fall $DS = Secrecy$ gilt, ist die im Prädikat *secureLinksViolation\4* definierte Bedingung $\texttt{read} \in ACT$ nicht erfüllt. Es kann hier also auch keine Parameterbelegung gefunden werden, die diese Prädikate erfüllt. Die dem Nutzer ausgegebene Parameterbelegung ist in Algorithmus 5.5 dargestellt.

Die Secure-Links-Analyse schlägt also mit zwei Fehlerfällen fehl. Es ist wiederzuerkennen, dass genau die drei Szenarien, die in dem aufgeteilten Beispielmodell (siehe Abbildung 5.3) dargestellt sind, überprüft werden. Im Allgemeinen ist jedoch zu sagen, dass die Analyse auch für komplexere Datenflüsse funktioniert. Dass an dieser Stelle für jede dieser drei Szenarien ein neues Datum erzeugt wird, dient nur der einfacheren Abbildung der UMLsec-

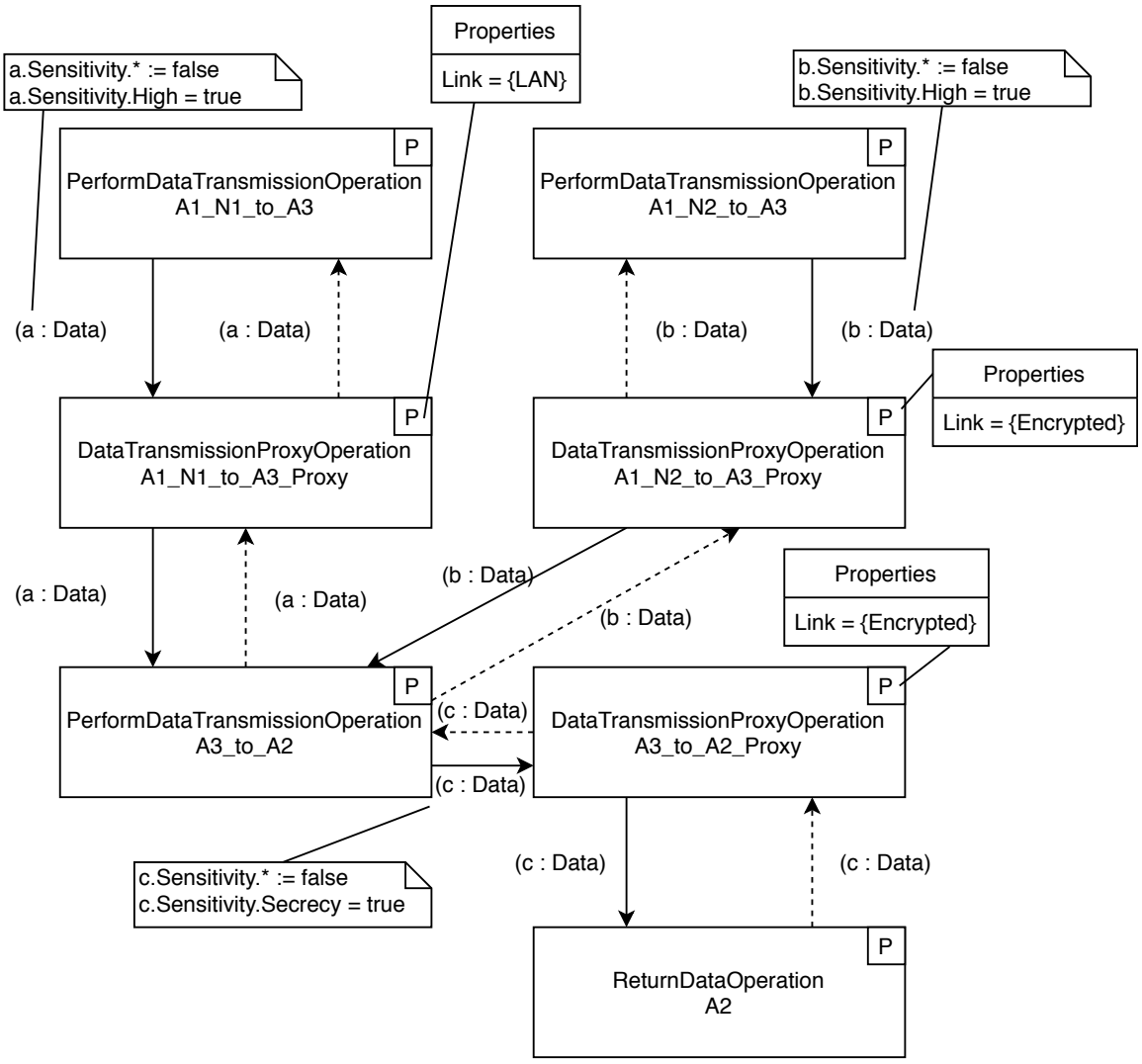

Abbildung 5.6.: Ein vereinfachter Auszug des Operationsmodells, welches zu Abbildung 5.2 korrespondiert.

```
1 ATT = 'default',          ACT = ['delete'],
2 OP = A1_N2_to_A3_Proxy, D = CallParameter,
3 DS = High,               L = Encrypted ;
4
5 ATT = 'default',          ACT = ['delete'],
6 OP = A1_N2_to_A3_Proxy, D = ReturnValue,
7 DS = High,               L = Encrypted ;
```

Algorithmus 5.5.: Die Ausgabe der Secure-Links-Analyse für das Beispiel in Abbildung 5.6.

Analyse auf die DCP-Analyse. Die Analyse funktioniert offensichtlich auch, wenn anstatt der Erzeugung eines neuen Datums, die Charakteristiken des bestehenden Datums geändert werden und es weiter verwendet wird.

5.3. Modelltransformation

Für die Modelltransformation des UMLsec-Eingabemodells auf das DCP-Eingabemodell sind in Abschnitt 5.2 bereits einige grundlegende Annahmen an die Struktur des Operationsmodells definiert. Diese können als notwendige Bedingungen an das DCP-Eingabemodell betrachtet werden. Zusätzlich soll die Modelltransformation die Semantik des UMLsec-Eingabemodells möglichst genau erhalten. Im Folgenden wird zunächst die Modelltransformation von UMLsec nach DCP erklärt und anschließend die Modelltransformation von DCP nach UMLsec beschrieben.

5.3.1. UMLsec zu Data-Centric Palladio

Zur besseren Nachvollziehbarkeit wird die Modelltransformation anhand der entstehenden einzelnen Modelle des PCM strukturiert. Im Folgenden wird zunächst die Erstellung des Komponentenverzeichnisses beschrieben. Anschließend werden das Systemmodell, die Ausführungsumgebung und die Allokation erstellt. Die Erstellung des Nutzungsmodells vervollständigt dann das PCM-Modell. Abschließend wird die Erstellung des Datenflussmodells erläutert.

5.3.1.1. Komponentenverzeichnis

Die Transformation erzeugt zunächst aus dem UMLsec-Eingabemodell das Komponentenverzeichnis des DCP-Eingabemodells. Dazu wird zunächst ein Datentyp (composite data type) *Data* definiert. Da die UMLsec-Modelle keine Informationen über aufgerufene Methoden oder verwendete Datentypen bieten, wird dieser Datentyp für alle definierten Schnittstellen und Datenflüsse verwendet. Für jedes Artefakt des UMLsec-Eingabemodells wird eine einfache Komponente (basic component) und eine Schnittstelle (operation interface) erstellt. Die Schnittstelle verfügt über eine Methodensignatur (operation signature) mit einem Parameter und einem Rückgabewert von jeweils dem Typ *Data*. Zur Verwendung mit den separat definierten Datenflüssen werden die Methodensignaturen der Schnittstellen mit dem Stereotyp «`OperationSignatureDataRefinement`» annotiert. Dies dient zur Spezifikation der zwischen Komponenten kommunizierten Daten im Datenflussmodell [12]. Die Komponenten stellen die jeweils zu ihnen korrespondierenden Schnittstellen bereit (operation provided role). Für jede Abhängigkeit, die ein Artefakt A_{UMLsec} des UMLsec-Eingabemodells zu einem Artefakt B_{UMLsec} besitzt, definiert die zu dem Artefakt A_{UMLsec} korrespondierende Komponente A_{DCP} eine Abhängigkeit (operation required role) auf die Schnittstelle, die von der Komponente B_{DCP}, welche zu dem Artefakt B_{UMLsec}

korrespondiert, realisiert wird. Die UMLsec-Stereotypen der Sicherheitseigenschaften, die an den Abhängigkeiten annotiert sind, werden im DCP-Eingabemodell als Teil des Datenflussmodells modelliert und werden daher an dieser Stelle keinem Modellelement des Komponentenverzeichnisses zugewiesen.

Schließlich wird das Verhalten der Komponenten in Form von jeweils einer RDSEFF (resource demanding service effect specification) definiert. Da das UMLsec-Modell keinerlei Aussagen über das Verhalten macht, reicht es hier, die externen Aufrufe (external call action) oder internen Aktionen (internal action) zu modellieren, die benötigt werden, um die Datenflussmodelle zu spezifizieren. Für jede Schnittstelle, von der die Komponente abhängt, wird ein externer Aufruf in der RDSEFF modelliert. Hat eine Komponente keine externen Abhängigkeiten wird nur eine interne Aktion modelliert. An die externen Aufrufe und internen Aktionen wird jeweils der DCP-Stereotyp «`DataProcessingSpecification`» annotiert, der es erlaubt, im Datenflussmodell die durch diese Aufrufe und Aktionen beschriebenen Datenflüsse zu modellieren. Ein Beispiel eines Komponentenverzeichnisses ist in Abbildung 5.7 dargestellt. Dabei dient das in Abbildung 5.2 vorgestellte Beispiel als das korrespondierende UMLsec-Eingabemodell.

Abschließend ist zu erwähnen, dass eine technische Limitation der bestehenden Implementation der Transformation des DCP-Eingabemodells in das Operationsmodell vorliegt. Aufgrund dieser Limitation ist es notwendig, für jede Komponente eine eigene Schnittstelle anzulegen. Sobald diese Limitation gelöst ist, kann an dieser Stelle auch nur eine einzige Schnittstelle erzeugt werden.

5.3.1.2. Systemmodell, Ausführungsumgebung und Allokation

Die Erzeugung des Systemmodells, der Ausführungsumgebung und des Verteilungsmodells ist ein Transformationsschritt, der gemeinsam ausgeführt werden muss. Da die voneinander abhängigen Komponenteninstanzen (assembly context) auf unterschiedlichen Rechnerknoten (resource container) alloziert sind (allocation context), muss für eine Verbindung zwischen zwei Komponenteninstanzen (assembly connector) ebenfalls eine Netzwerkressource (linking resource) zwischen den Rechnerknoten bestehen. Um diese Transformation verständlich darzustellen, werden zunächst bestimmte Modifikationen an einer Kopie des UMLsec-Eingabemodells vorgenommen. Diese sind als flüchtige Modifikationen zu verstehen, die nur zur besseren Nachvollziehbarkeit dieser Transformation vorgenommen werden. Der Fokus liegt dabei auf der Syntax der entstehenden Modelle. Die Modifikationen an dem UMLsec-Modell werden an dem Beispiel in Abbildung 5.8 veranschaulicht.

Das Ziel der Änderungen an dem UML-Modell ist es, eine Darstellung zu schaffen, an der die Transformation zu dem Systemmodell und der Ausführungsumgebung einfacher ersichtlich ist. Daher ist eine Darstellung sinnvoll, die konzeptionell näher an den eigentlichen Softwareinstanzen ist, die auf konkrete Rechnerknoten installiert werden. Daher werden in einem ersten Schritt die Artefakte durch Objekte ersetzt, die konzeptionell Instanzen der Artefakte darstellen. Diese Objekte sind ähnlich den Komponenteninstanzen aus DCP zu

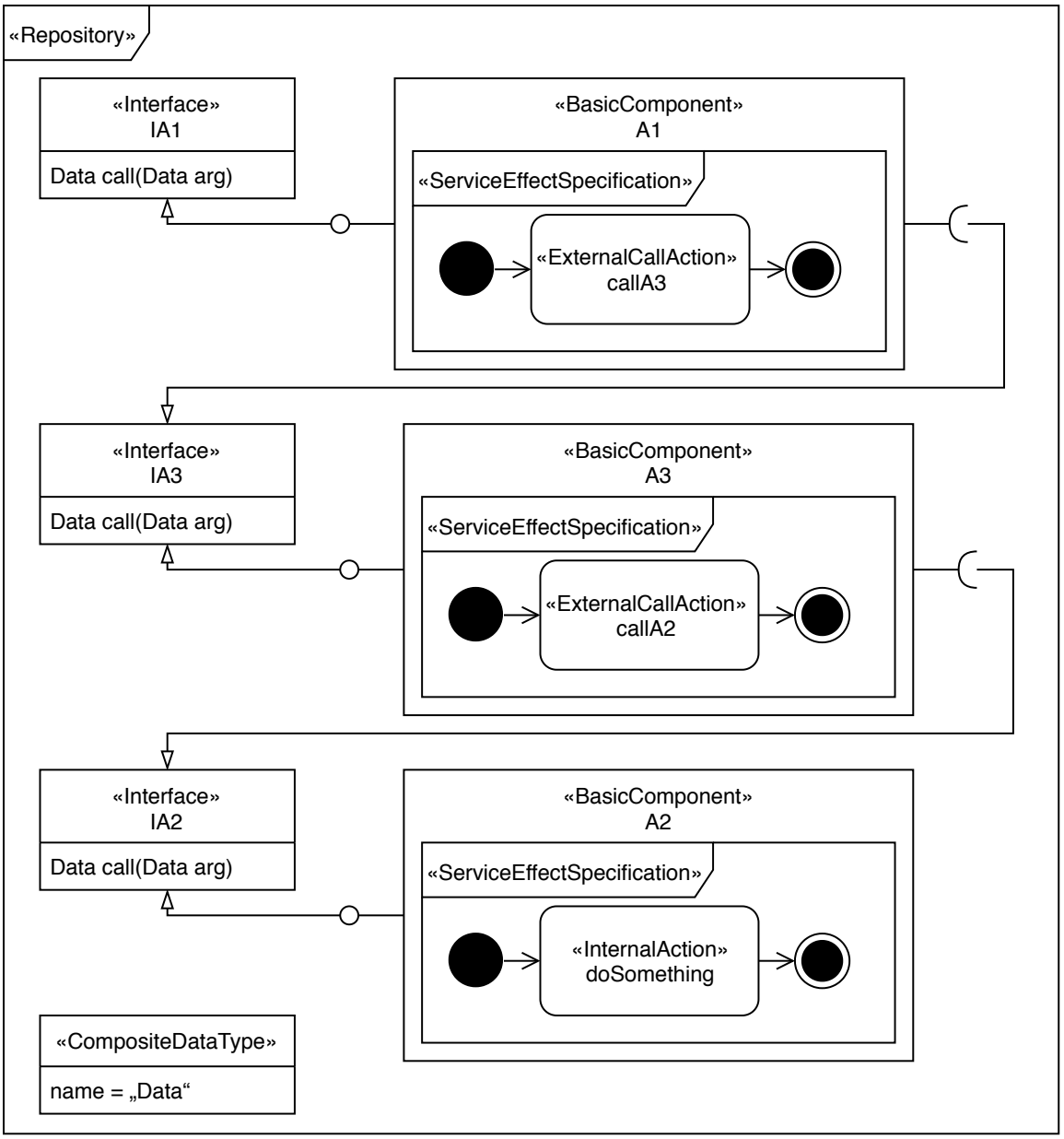

Abbildung 5.7.: Das durch die Modelltransformation entstandene Komponentenverzeichnis, welches zu Abbildung 5.2 korrespondiert.

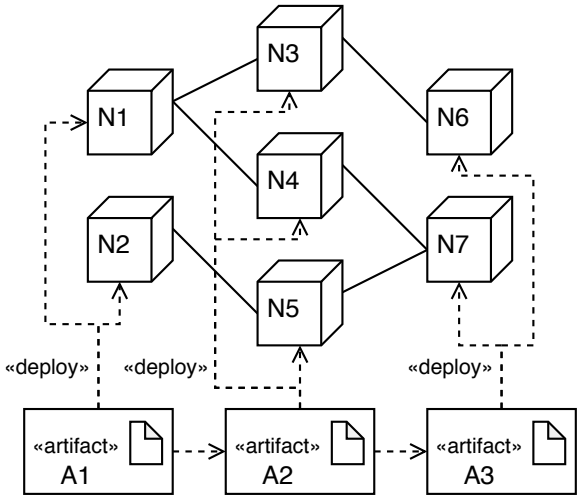

Abbildung 5.8.: Ein UML-Verteilungsdiagramm, das als Beispiel für die Erzeugung des Systemmodells, der Ausführungsumgebung und der Allokation dient.

verstehen. Für jedes Artefakt des Modells wird für jede Verteilungsbeziehung des Artefakts ein Objekt erstellt und auf den Rechnerknoten, der das Ziel der Verteilungsbeziehung ist, verteilt. Als nächstes müssen die Abhängigkeitsbeziehungen der Artefakte durch die erstellten Objekte repräsentiert werden. Sei dazu o_A ein neu erzeugtes Objekt, das ursprünglich aus dem Artefakt A erzeugt wird, und sei ein Artefakt B gegeben, zu dem A eine Abhängigkeit hat. Dann wird eine Abhängigkeit von o_A zu jedem Objekt o_B^i, das aus Artefakt B erzeugt wird und das von o_A aus erreichbar ist, erstellt. Die Definition von Erreichbarkeit ist in diesem Kontext folgendermaßen gegeben: Ein Objekt o kann ein Objekt p genau dann erreichen, wenn der Rechnerknoten, auf dem o alloziert ist, über einen Kommunikationspfad zu dem Rechnerknoten verfügt, auf dem p alloziert ist. Ein Beispiel für ein Modell nach diesen Änderungen ist in Abbildung 5.9 dargestellt. Das hervorgehobene Objekt stellt einen Problemfall dar, der durch die nächste Änderung behoben wird.

Mit der zuvor erstellten Darstellung sind bereits die für das DCP-Eingabemodell notwendigen Komponenteninstanzen und ihre Allokation auf Rechnerknoten sichtbar. Jedes Objekt, das aus einem Artefakt A erzeugt wurde, stellt eine Komponenteninstanz der Komponente dar, die zu Artefakt A korrespondiert. Ein Problem, das nach diesem Schritt auftreten kann, ist, dass ein Artefakt A eine Abhängigkeit auf ein Artefakt B hat und für ein Objekt o_A des Artefakts A mehrere Objekte o_B^i des Artefakts B erreichbar sind. In diesem Fall wird eine Abhängigkeit zwischen jeweils o_A und o_B^i erstellt. In DCP kann eine benötigte Rolle (required role) einer Komponenteninstanz jedoch nur von genau einer Komponenteninstanz bereitgestellt werden (provided role). Genauer ausgedrückt muss für alle paarweise verschiedenen Komponenteninstanzkonnektoren (assembly connector) c_1 und c_2 gelten, dass wenn die konsumierende Komponenteninstanz (requiring assembly context) von c_1 und c_2 identisch ist, sich die benötigte Rolle von c_1 und c_2 unterscheiden muss. Da jedoch die Objekte o_B alle aus dem Artefakt B erzeugt werden, ist dies bisher nicht der Fall. Abhängigkeitsbeziehungen dieser Art müssen also aufgelöst werden. Dies geschieht dadurch,

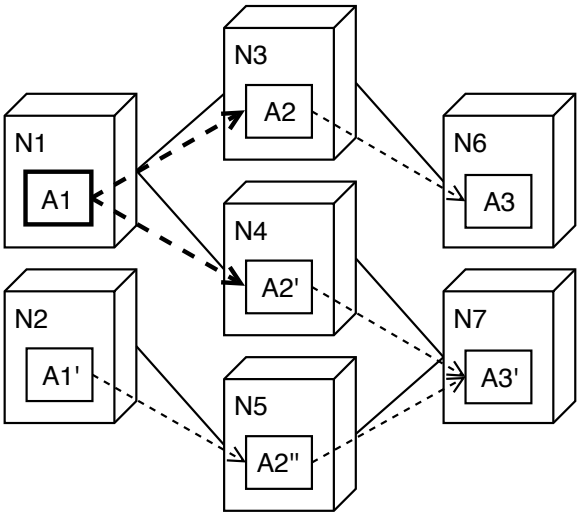

Abbildung 5.9.: Ein modifiziertes UML-Verteilungsdiagramm basierend auf Abbildung 5.8.

dass aus dem Artefakt A ein weiteres Objekt o'_A erzeugt wird und auf dem Rechnerknoten alloziert wird, auf dem o_A alloziert ist. Die Quelle einer der Abhängigkeitsbeziehungen zwischen o_A und einem o^i_B wird anschließend zu o'_A verändert. Dies ist in Abbildung 5.10 dargestellt. Die Änderung zu dem vorherigen Modell ist hervorgehoben.

Aus diesem Zwischenmodell ist es nun einfach möglich die korrespondierenden DCP-Modellelemente zu extrahieren. Für jedes Objekt o wird eine Komponenteninstanz erzeugt. Sie ist eine Instanz der Komponente, die zu dem Artefakt, aus dem o erzeugt wurde, korrespondiert. Die Komponenteninstanzkonnektoren sind in diesem Zwischenmodell durch die Abhängigkeiten zwischen den erzeugten Objekten gegeben und können daraus direkt erzeugt werden. Schließlich müssen noch die vom System bereitgestellten Rollen (operation provided role) definiert werden. Für jedes Objekt o im Zwischenmodell, das nicht das Ziel einer Abhängigkeit eines anderen Objekts ist, wird eine vom System bereitgestellte Rolle erzeugt und ein Konnektor (provided delegation connector), der diese Rolle mit der Komponenteninstanz, die zu o korrespondiert, verbindet. Dies entspricht im UMLsec-Eingabemodell den Artefakten, von denen kein anderes Artefakt abhängig ist. Sie dienen als Einsprungpunkt für die Nutzungsszenarien und bilden damit den Start des Datenflusses innerhalb des Systems.

Die Rechnerknoten und Kommunikationspfade wurden bei der Erstellung des Zwischenmodells nicht verändert. Daher lässt sich die Ausführungsumgebung auch direkt aus dem UMLsec-Eingabemodell extrahieren. Die Rechnerknoten des UMLsec-Eingabemodells (node) können direkt auf die Rechnerknoten des DCP-Eingabemodells (resource container) abgebildet werden. Analog können die Kommunikationspfade direkt auf Netzwerkressourcen abgebildet werden. Für jeden Rechnerknoten des UMLsec-Eingabemodells wird daher ein Rechnerknoten im DCP-Eingabemodell erstellt. Für jeden Kommunikationspfad, der zwei Rechnerknoten x und y verbindet, wird im DCP-Modell eine Netzwerkressource erstellt, die die zu x und y korrespondierenden Rechnerknoten miteinander verbindet. Schließlich werden die Netzwerkressourcen, deren korrespondierende Kommunikationspfade über

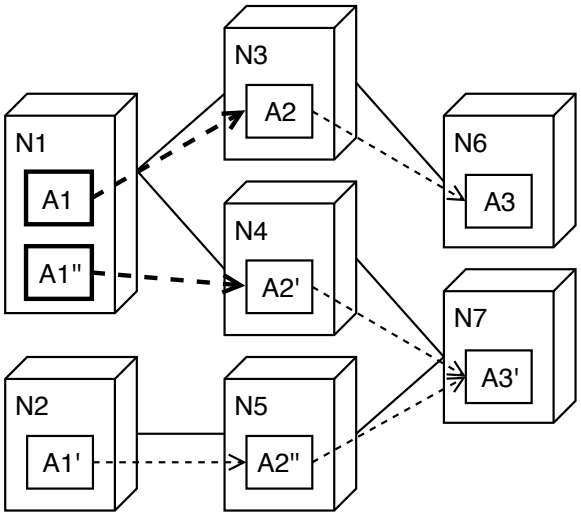

Abbildung 5.10.: Die finale Modifikation des UML-Verteilungsdiagramm aus Abbildung 5.8. In diesem Modell können notwendige DCP-Modellelemente leicht erkannt werden.

einen UML-Stereotyp $s \in \{$«`Internet`», «`LAN`», «`wire`», «`encrypted`»$\}$ verfügen, mit dem DCP-Stereotyp «`Characterizable`» annotiert. Dies erlaubt es, im Datenflussmodell Charakteristiken zu spezifizieren und der Netzwerkressource zuzuweisen. Wie auch schon bei den UML-Stereotypen der Sicherheitseigenschaften werden diese als Teil des Datenflussmodells modelliert und daher an dieser Stelle auf kein Element der Ausführungsumgebung abgebildet.

Abschließend kann die Allokation der Komponenteninstanzen aus dem Zwischenmodell extrahiert werden. Für jedes Objekt *o*, das auf einem Rechnerknoten *n* alloziert ist, wird eine Allokation der zu *o* korrespondierenden Komponenteninstanz auf den zu *n* korrespondierenden Rechnerknoten (allocation context) erstellt. Für das zu Beginn des Kapitels beschriebene Beispiel (siehe Abbildung 5.2) sind das Systemmodell, die Ausführungsumgebung und das Verteilungsmodell in Abbildung 5.11 dargestellt.

5.3.1.3. Nutzungsmodell

Das Nutzungsmodell hat kein Gegenstück in dem UMLsec-Eingabemodell. Es wird in DCP als Ausgangspunkt für die Datenflüsse verwendet und der Vollständigkeit halber spezifiziert. Nach der Erzeugung des Systemmodells existieren bereits einige vom System bereitgestellte Rollen. Für jede dieser Rollen *r* wird ein Nutzungsszenario (usage scenario) definiert, das einen Systemaufruf (entry level system call) an *r* durchführt. Wie auch schon an einem externen Aufruf und einer internen Aktion einer RDSEFF wird an diesem Systemaufruf der Stereotyp «`DataProcessingSpecification`» annotiert, mit dem es im Datenflussmodell möglich ist, den Datenfluss des Systemaufrufs zu spezifizieren. Die Systemaufrufe der Nutzungsszenarien dienen dabei als Datenquellen für den initialen Datenfluss. In Abbildung 5.12 ist ein Beispiel für das erstellte Nutzungsmodell dargestellt.

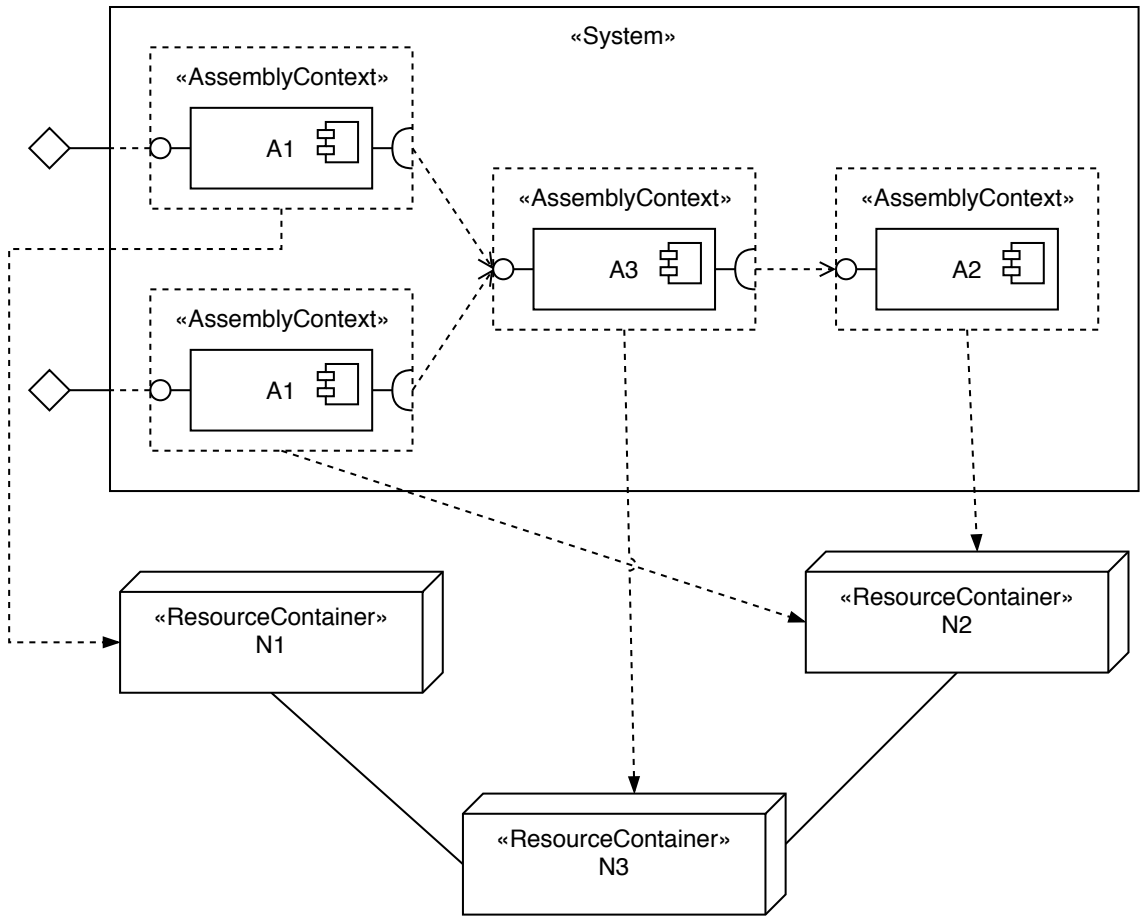

Abbildung 5.11.: Das Systemmodell, die Ausführungsumgebung und das Verteilungsmodell, welche zu Abbildung 5.2 korrespondieren.

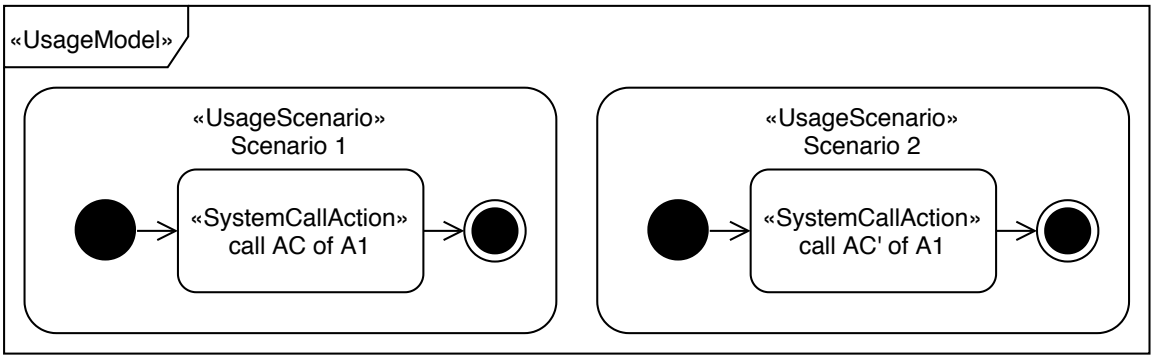

Abbildung 5.12.: Das Nutzungsmodell, das als Teil der Modelltransformation für das Beispiel aus Abbildung 5.2 erzeugt wurde.

Tabelle 5.4.: Die in der Secure-Links-Analyse verwendeten Charakteristiktypen und Werte.

Charakteristiktyp	Mögliche Werte
Link	Internet, LAN, Wire, Encrypted
Sensitivity	Secrecy, Integrity, High

5.3.1.4. Datenflussmodell

Zur Erstellung des Datenflussmodells müssen zunächst die Charakteristiktypen erstellt werden, die zur Spezifikation von Sicherheitseigenschaften der Daten und Art der physischen Netzwerkressource verwendet werden. Diese sind unabhängig vom UMLsec-Eingabemodell, müssen aber aktuell noch manuell als Teil des Modells spezifiziert werden. In einer zukünftigen Implementation ist es denkbar, die Charakteristiktypen für verschiedene Analysen als Teil der Anwendung auszuliefern, wodurch sie nicht vom Nutzer manuell spezifiziert werden müssen. Die Charakteristiken sind in Tabelle 5.4 dargestellt.

Um die Datenflüsse für das UMLsec-Eingabemodell umzusetzen, müssen zunächst einige Konzepte des Datenflussmodells eingeführt werden. Datenverarbeitungscontainer (data processing container), die an die externen Aufrufe, internen Aktionen oder Systemaufrufe annotiert werden können, legen das Verhalten des Datenflusses dieser Aktionen fest. Charakteristikcontainer, die hier an Netzwerkressourcen annotiert werden, spezifizieren die Art der Netzwerkressource mit einer Charakteristik vom Typ *Link*. Die Verfeinerung der Methodensignaturen der Komponentenschnittstellen (operation signature data refinement) dient dazu, die Methodensignaturen um Datenspezifikationen zu erweitern. Letzteres wird für die Secure-Links-Analyse nicht benötigt und ist nur notwendig, um ein gültiges Datenflussmodell zu erzeugen. Deshalb wird zunächst für jede Schnittstelle des Komponentenverzeichnisses eine Methodensignaturverfeinerung im Datenflussmodell angelegt. Diese verfügt über ein parameterbasiertes Datum (parameter based data) und ein rückgabebasiertes Datum (return based data) vom Typ *Data*.

Anschließend werden die Charakteristikcontainer angelegt und an die entsprechenden Netzwerkressourcen annotiert. Für jeden Kommunikationspfad des UMLsec-Eingabemodells existiert genau eine korrespondierende Netzwerkressource. Daher wird für jeden Kommunikationspfad p, der über einen Stereotyp

$$s \in \{\texttt{«Internet»}, \texttt{«LAN»}, \texttt{«wire»}, \texttt{«encrypted»}\}$$

verfügt, ein Charakteristikcontainer angelegt, der den zu s gleichnamigen Wert der *Link*-Charakteristik enthält. Sobald dieser Charakteristikcontainer erstellt ist, kann er an die zu p korrespondierende Netzwerkressource annotiert werden. Dies ist in Abbildung 5.13 dargestellt.

Schließlich werden die Datenflüsse definiert. Wie bereits in Abbildung 5.3 gezeigt, ist es für die Analyse unerheblich, ob es sich dabei um eine zusammenhängende oder getrennte Betrachtung der einzelnen relevanten Teilmodelle handelt. Ferner ist es nicht möglich aus dem UMLsec-Modell eine sinnvolle Datenverarbeitungsspezifikation zu extrahieren.

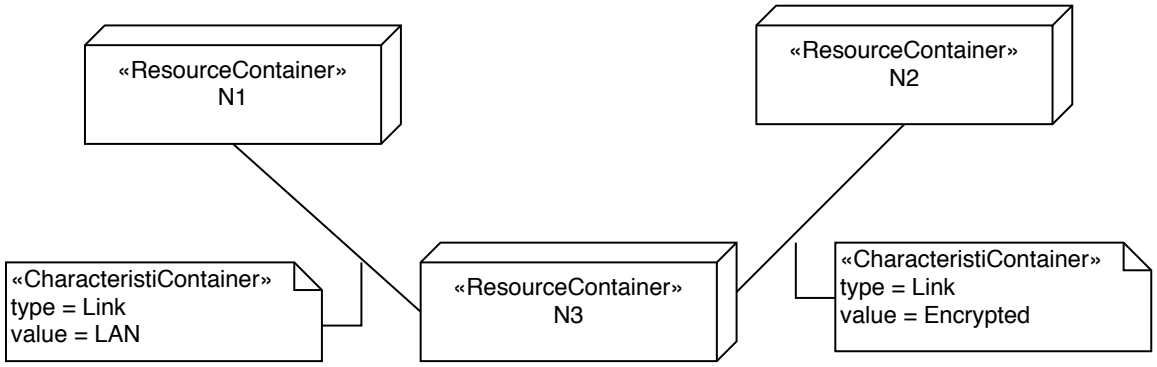

Abbildung 5.13.: Die Charakteristiken der Netzwerkressourcen der Ausführungsumgebung, die aus dem UMLsec-Beispiel aus Abbildung 5.2 erzeugt wurde.

Daher wird der Datenfluss im Folgenden so definiert, dass eine Komponente vor jedem externen Aufruf ein neues Datum mit der geforderten Sicherheitseigenschaft anlegt, an das Ziel des Aufrufs überträgt und von dort das übertragene Datum wieder zurückerhält. Das erhaltene Datum ist nicht mehr benötigt und wird verworfen. Wenn die Komponente auch eine Schnittstelle für andere Komponenten bereitstellt, wird das erhaltene Datum unverändert an den vorherigen Datenverarbeitungsvorgang zurückgegeben. Dies gilt sowohl für die Datenflüsse, die das Verhalten der Komponenten beschreiben, als auch für die Datenflüsse, die das Nutzungsverhalten beschreiben. Es ist anzumerken, dass die unveränderte Rückgabe des erhaltenen Datums von zentraler Bedeutung für die Secure-Links-Analyse ist, da so die Sicherheitseigenschaft sowohl an Eingabeparametern, als auch an Rückgabevariablen geprüft wird.

Die Transformation geschieht folgendermaßen. Zunächst werden die Datenflüsse der Nutzungsszenarien erzeugt. Da diese Szenarien keine Repräsentation im UMLsec-Eingabemodell haben, wird nur das notwendige Minimum modelliert, um ein gültiges Datenflussmodell darzustellen. Für jeden Datenverarbeitungscontainer, der das Verhalten eines Systemaufrufs beschreibt, wird ein Datenverarbeitungsvorgang angelegt, der ein Datum vom Typ *Data* ohne weitere Charakteristiken erzeugt. Dieses Datum wird an den Parameter der Methodensignaturverfeinerung der Schnittstelle übertragen, die von dem Systemaufruf aufgerufen wird. Die Rückgabevariable dieses Aufrufs wird verworfen.

Die Datenflüsse von externen Aufrufen werden ähnlich modelliert. Für jeden Datenverarbeitungscontainer, der das Verhalten eines externen Aufrufs beschreibt, wird ein Datenverarbeitungsvorgang angelegt, der ein Datum vom Typ *Data* erzeugt. Diesem Datum werden als Teil des Datenverarbeitungsvorgangs Charakteristiken zugewiesen, welche aus dem UMLsec-Eingabemodell extrahiert werden können. Nach der Definition dieser Transformation existiert pro Abhängigkeit eines Artefakts A_{UMLsec} genau ein externer Aufruf in der RDSEFF der zu A_{UMLsec} korrespondierenden Komponente A_{DCP}. Es kann also von dem Datenverarbeitungscontainer, über den externen Aufruf, an dem er annotiert wird, zurück zu der UML-Abhängigkeit traversiert werden, aus der dieser externe Aufruf erzeugt wurde. Die an dieser Abhängigkeit annotierten UML-Stereotypen $s \in \{Secrecy, Integrity, High\}$ werden in Form der gleichnamigen Werte der Charakteristik vom Typ *Sensitivity* an das

erzeugte Datum annotiert. Dieses Datum wird an den Parameter der Methodensignaturverfeinerung der Schnittstelle übertragen, die das Ziel des externen Aufrufs ist. Die Rückgabevariable dieses Aufrufs wird verworfen und das parameterbasierte Eingabedatum, das dieser Datenverarbeitungscontainer von dem vorherigen Datenverarbeitungscontainer *prev* erhalten hat, fließt an *prev* zurück.

Eine vereinfachte Form dieser Datenflüsse ist für interne Aktionen modelliert. Eine Komponente, deren RDSEFF nur über eine interne Aktion und keine externen Aufrufe verfügt, repräsentiert ein Artefakt, das keine Abhängigkeit hat, sondern nur eine Schnittstelle für andere Artefakte zur Verfügung stellt. Daher ist es für den Datenfluss einer solchen Komponente ausreichend, das erhaltene Datum an den vorherigen Datenverarbeitungsvorgang zurückzugeben. Für jeden Datenverarbeitungscontainer, der das Verhalten einer internen Aktion beschreibt, wird ein Datenverarbeitungsvorgang angelegt, der das erhaltene Datum an den vorherigen Datenverarbeitungscontainer zurückgibt. Ein Beispiel der für die Secure-Links-Analyse relevanten Datenflüsse ist in Abbildung 5.14 dargestellt.

Abschließend ist anzumerken, dass die Secure-Links-Analyse in DCP auch bei zusammenhängenden Datenflüssen angewendet werden kann. Die Modellierung der einzelnen zu betrachtenden Szenarien als getrennte Datenflüsse dient nur dazu möglichst wenig neue Informationen, die im UMLsec-Eingabemodell nicht vorhanden sind, in das DCP-Eingabemodell einzuführen. Die Modellierung als getrennte Datenflüsse benötigt weniger Annahmen über den Datenfluss, der aus dem UMLsec-Modell nicht ersichtlich ist, als die Modellierung als ein zusammenhängender Datenfluss, bei dem an verschiedenen Stellen die Charakteristiken des Datums modifiziert werden und zusätzliche Annahmen über die Schnittstelle des aufgerufenen Artefakts getroffen werden müssen.

5.3.2. Data-Centric Palladio zu UMLsec

Die Modelltransformation eines DCP-Modells zu einem UMLsec-Modell, das als Eingabe an die Secure-Links-Analyse dienen kann, ist weniger komplex als die umgekehrte Transformationsrichtung. Dies ist hauptsächlich darauf zurückzuführen, dass das UMLsec-Eingabemodell aus weniger notwendigen Modellelementen besteht und eine weniger strikte Semantik als das DCP-Eingabemodell aufweist. Für die Transformation ist es sinnvoll, zunächst die Modellelemente aus DCP zu UMLsec zu transformieren, die eine bijektive Relation zu ihren korrespondieren Elementen aufweisen. Dies ist der Fall für Komponenten und Artefakte, sowie DCP-Rechnerknoten und UML-Rechnerknoten. Die Kommunikationspfade werden aus den Netzwerkressourcen generiert, aber es liegt hier keine bijektive Relation vor. Anschließend werden aus den Allokationen von und den Verbindungen zwischen Komponenteninstanzen die Verteilungsbeziehungen von und Abhängigkeiten zwischen Artefakten abgeleitet. Schließlich muss das Datenflussmodell analysiert werden, um die Stereotypen, die an die Abhängigkeiten und Kommunikationspfade annotiert werden, zu extrahieren. Diese Transformation ist im Folgenden beschrieben.

Für jede Komponente des Komponentenverzeichnisses wird ein dazu korrespondierendes Artefakt erzeugt und für jeden DCP-Rechnerknoten wird ein dazu korrespondierender

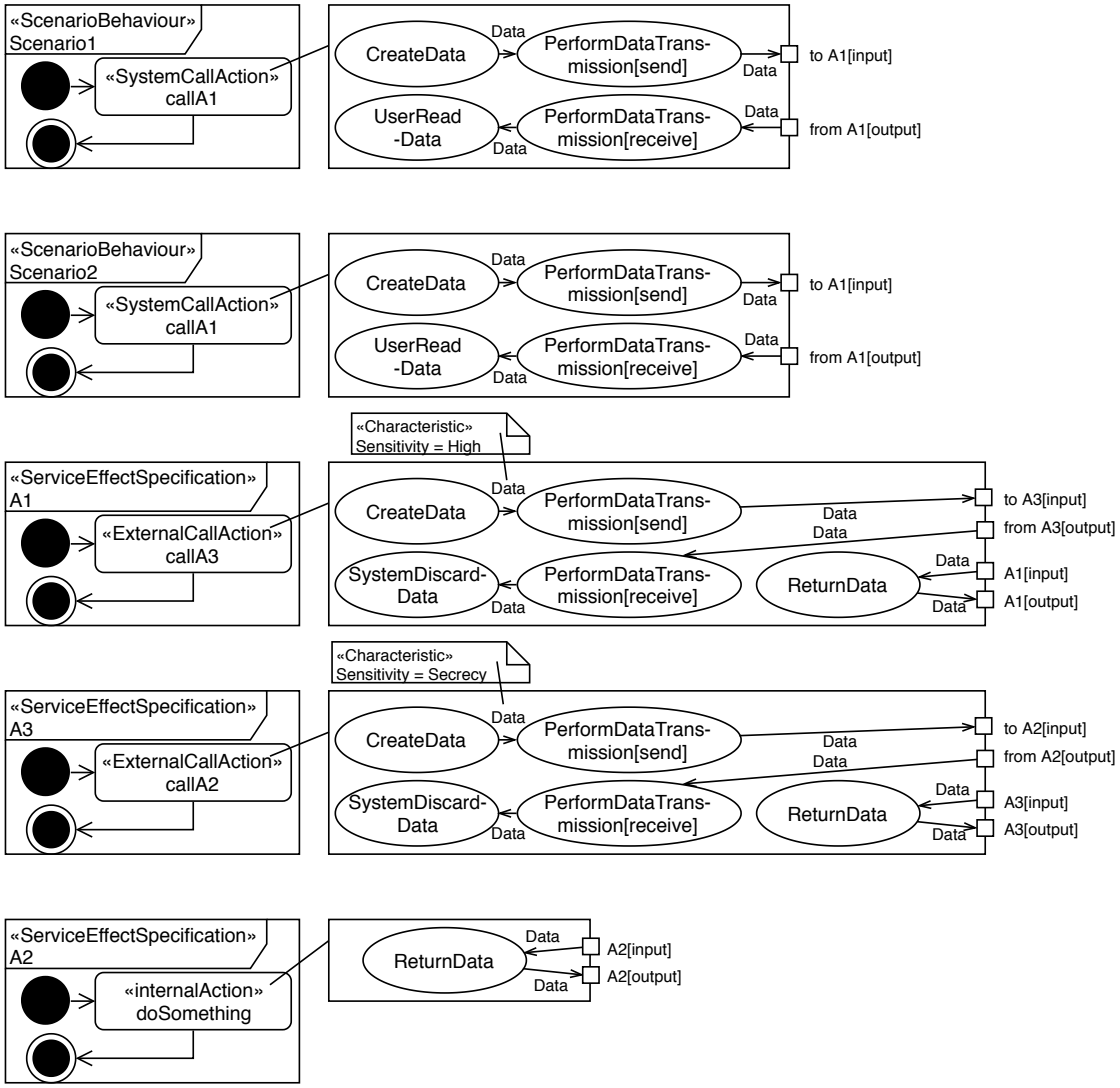

Abbildung 5.14.: Der definierte Datenfluss für das Beispiel aus Abbildung 5.2.

UML-Rechnerknoten erzeugt. Für jede Netzwerkressource i sei R_i die Menge von DCP-Rechnerknoten, die durch i verbunden sind ($|R_i| \geq 2$). Dann wird für jede Netzwerkressource i eine Menge von Kommunikationspfaden so erzeugt, dass für die UML-Rechnerknoten, für die gilt, dass ihr jeweils korrespondierender DCP-Rechnerknoten ein Element von R_i ist, eine vollvermaschte Topologie entsteht. Aus einer Netzwerkressource, die z. B. vier DCP-Rechnerknoten A, B, C und D miteinander verbindet, entstehen also sechs Kommunikationspfade, die jeweils die Rechnerknoten (a, b), (a, c), (a, d), (b, c), (b, d) und (c, d) miteinander verbinden. Dabei sind a, b, c und d als die zu jeweils A, B, C und D korrespondierenden UML-Rechnerknoten zu verstehen. Verfügt die Netzwerkressource i über eine annotierte Charakteristik des Typs *Link*, wird der UMLsec-Stereotyp, der zu dem Wert dieser Charakteristik korrespondiert, an alle aus i erzeugten Kommunikationspfade annotiert.

Um die Abhängigkeiten zwischen den Artefakten zu erzeugen, muss das Systemmodell des DCP-Eingabemodells betrachtet werden. Für jeden Komponenteninstanzkonnektor, der zwei Komponenteninstanzen A_n und B_m miteinander verbindet, die Instanzen der Komponenten A_{DCP} bzw. B_{DCP} sind, wird eine Abhängigkeitsbeziehung zwischen den zu A_{DCP} und B_{DCP} korrespondierenden Artefakten A_{UMLsec} und B_{UMLsec} eingefügt, sofern noch keine Abhängigkeitsbeziehung zwischen A_{UMLsec} und B_{UMLsec} besteht. Sei dabei A_n die konsumierende Komponenteninstanz (requiring assembly context) und B_m die bereitstellende Komponenteninstanz (providing assembly context). Dann ist die Quelle der erzeugten Abhängigkeitsbeziehungen A_{UMLsec} und das Ziel ist B_{UMLsec}. Als Nächstes sind die Sicherheitsanforderungen an die Daten zu ermitteln, die zwischen den Komponenten A_{DCP} und B_{DCP} fließen, da diese Sicherheitsanforderungen als Stereotyp an der Abhängigkeitsbeziehung annotiert werden. Dazu wird zunächst die benötigte Rolle des Komponenteninstanzkonnektor gespeichert. Diese ist im Folgenden mit r bezeichnet. Anschließend wird über alle Datenverarbeitungscontainer iteriert, die an einen externen Aufruf annotiert sind, für den gilt, dass die Rolle des externen Service gleich r ist und dass er ein Bearbeitungsschritt einer SEFF von A_{DCP} ist. Dies entspricht also den Datenverarbeitungscontainern, die an die externen Aufrufe annotiert sind, welche von Komponente A_{DCP} an Komponente B_{DCP} ausgeführt werden.

Aus jedem dieser Datenverarbeitungscontainer werden alle enthaltene Datenverarbeitungsvorgänge PDT_i des Typs *PerformDataTransmission* extrahiert und in einer gemeinsamen Liste L gespeichert. Diese Datenverarbeitungsvorgänge repräsentieren jeweils die Übertragung eines Datums von der Komponente A_{DCP} zu der Komponente B und den Erhalt eines Datums von der Komponente B_{DCP}. Für jeden Datenverarbeitungsvorgang PDT_i der Liste L werden nun die Charakteristiken der ausgehenden und eingehenden Daten ermittelt. Verfügt ein Datum über Charakteristiken vom Typ *Sensitivity* wird diese Charakteristik einer Zwischenmenge C_i von Charakteristiken hinzugefügt. Es existiert also für jeden Datenverarbeitungsvorgang PDT_i eine Menge C_i. Da in UMLsec nicht zwischen Sicherheitsanforderungen einzelner Operationen oder Variablen unterschieden wird, muss hier eine Überabschätzung für den Stereotyp der Abhängigkeitsbeziehungen getroffen werden. Dazu wird die Vereinigung $C = \bigcup C_i$ gebildet. Die Menge C entspricht also der vereinigten Menge der Charakteristiken vom Typ *Sensitivity* aller Daten, die ausgehende oder eingehende Daten eines der Datenverarbeitungsvorgänge PDT_i sind.

```
1 ?- member(OP, [OperationList]),
2      S = [OP|_],
3      operationState(OP, ST),
4      preCallState(S, OP, ST, 'Sensitivity', V).
5
6 ?- member(OP, [OperationList]),
7      S = [OP|_],
8      operationReturnValue(OP, R),
9      returnValue(S, R, 'Sensitivity', V).
```

Algorithmus 5.6.: Ein Beispiel für die Prolog-Abfragen, die die Charakteristiken der ausgehenden und eingehenden Daten einer Operation ermitteln.

Die Ermittlung der Charakteristiken der eingehenden und ausgehenden Daten eines Datenverarbeitungsvorgangs PDT_i kann durch die Verwendung des generierten Prolog-Programms geschehen. Ein Beispiel für solche Prolog-Abfragen ist in Algorithmus 5.6 dargestellt. Dabei ist *OperationList* eine Liste der Namen der Stellvertreteroperationen (DataTransmissionProxyOperation) der Übertragungsoperationen (PerformDataTransmissionOperation), die aus PDT_i generiert werden.

Es handelt sich hierbei um dieselben Prädikate, die bereits bei der Definition der Analyse eingeführt und erklärt wurden (siehe Abschnitt 5.2). Die erste Abfrage listet alle Parameter und ihre Charakteristiken, während die zweite Abfrage dasselbe für die Rückgabevariablen tut.

Abschließend bleiben noch die Verteilungsbeziehungen zu bestimmen. Dazu wird für jeden Allokationskontext (allocation context) eine Verteilungsbeziehung wie folgt erzeugt, sofern nicht bereits eine solche Verteilungsbeziehung existiert: Sei A_i die Komponenteninstanz, die durch den Allokationskontext auf den DCP-Rechnerknoten N verteilt wird. Dann ist das Ziel der Verteilungsbeziehung genau der UML-Rechnerknoten, der zu dem DCP-Rechnerknoten N korrespondiert und die Quelle der Verteilungsbeziehung ist das Artefakt, dass zu der Komponente korrespondiert, von der A_i eine Instanz ist.

Einen Sonderfall stellt die Definition des Angreifers dar. In UMLsec ist dieser explizit in dem Eingabemodell dargestellt, während in DCP die Angreiferdefinitionen Teil des generierten Prolog-Programms sind und somit unabhängig vom Eingabemodell geändert werden können. Dies bedeutet aber auch, dass die Information des betrachteten Angreifers immer vollständig verloren geht, wenn ein UMLsec-Eingabemodell nach DCP transformiert wird. Daher kann an dieser Stelle die Definition der Angreifermächtigkeit nicht aus dem DCP-Modell, das zu UMLsec transformiert wird, abgeleitet werden. Da für Secure Links nur zwei Angreifer definiert sind, der Standardangreifer und der Insiderangreifer, ist der letzte Schritt der Modelltransformation von DCP nach UMLsec die Duplikation des erstellten UMLsec-Modells und das Setzen des Standardangreifers für die Angreifermächtigkeit in dem einen Modell, während in dem anderen der UMLsec-Modelle als Angreifermächtigkeit

der Insiderangreifer gewählt wird. Die Transformation von DCP nach UMLsec erzeugt also zwei UMLsec-Modelle für die Secure-Links-Analyse.

6. Secure Dependencies

Bei Secure Dependencies [1, Seite 59f] handelt es sich um eine strukturelle Sicherheitsanalyse aus UMLsec. Sie operiert auf einem Klassendiagramm, in dem Klassen, Schnittstellen, Abhängigkeiten von Klassen zu Schnittstellen und Realisierungsbeziehungen modelliert sind. Die Analyse überprüft, ob die von einer Klasse spezifizierten Sicherheitsanforderungen an eine Schnittstelle, von der die Klasse abhängt, von allen Klassen, die diese Schnittstelle realisieren, erfüllt werden. Im Folgenden wird zunächst diese Analyse, zusammen mit den Eingabemodellen, auf denen sie operiert, erklärt. Anschließend erfolgt die Beschreibung der Analyse in DCP. Abschließend wird die Modelltransformation zwischen den UMLsec- und DCP-Eingabemodellen beschrieben.

Im Folgenden ist die Syntax und Semantik der Eingabemodelle beschrieben. Die Secure-Dependencies-Sicherheitsanalyse erhält als Eingabemodell ein UML-Klassendiagramm, welches aus mindestens einem Subsystem besteht, das mit dem «`secure dependencies`» Stereotyp annotiert ist. Dieser Stereotyp dient lediglich als Marker für die Subsysteme, auf denen die Secure-Dependencies-Analyse durchgeführt wird. Bei den für die Analyse relevanten Modellelementen handelt es sich um Klassen, Abhängigkeitsbeziehungen, Realisierungsbeziehungen und Schnittstellen. Dabei sind Klassen und Abhängigkeiten um UMLsec-Stereotypen erweitert. Für Abhängigkeiten werden die Stereotypen «`secrecy`», «`integrity`» und «`high`» verwendet, welche der in Abschnitt 2.3 definierten Semantik entsprechen. Für Klassen wird der Stereotyp «`critical`» verwendet, dessen Semantik nachfolgend erklärt ist.

Die Sicherheitsanforderungen, die eine Klasse *K* an eine Implementierung einer Schnittstelle oder den Aufrufer einer Schnittstelle stellt, werden durch die Annotation des Stereotyps «`critical`» an der Klasse *K* definiert. Der Stereotyp «`critical`» verfügt dafür über die Eigenschaftsdefinitionen `{secrecy}`, `{integrity}` und `{high}`. Ist die Sicherheitsanforderung, dass die Rückgabewerte einer Methode *m* einer Schnittstelle für die Verwendung in Klasse *K* vertraulich sein müssen, wird der Name der Methode dem Eigenschaftswert der Eigenschaftsdefinition `{secrecy}` des Stereotyps «`critical`» der Klasse *K* hinzugefügt. In Abbildung 6.1 ist dies an der Klasse *KeyGenerator* zu erkennen. Diese Klasse hat eine Abhängigkeit zu der Schnittstelle *RandomNumber* und verfügt über den Stereotyp «`critical`» mit dem Eigenschaftswert *{random()}* für die Eigenschaftsdefinition `{high}`. Dies bedeutet also, dass die Klasse *KeyGenerator* fordert, dass für die Rückgabewerte der Methode *random()* Vertraulichkeit und Integrität (high) garantiert sind.

Eine Klasse *L*, die eine Schnittstelle realisiert, kann ebenfalls mit dem Stereotyp «`critical`» annotiert werden. Im Falle einer Methode entspricht dies jedoch nicht der Sicherheitsanforderung, die an diese Methode gestellt wird, sondern beschreibt die Sicherheitsgarantien

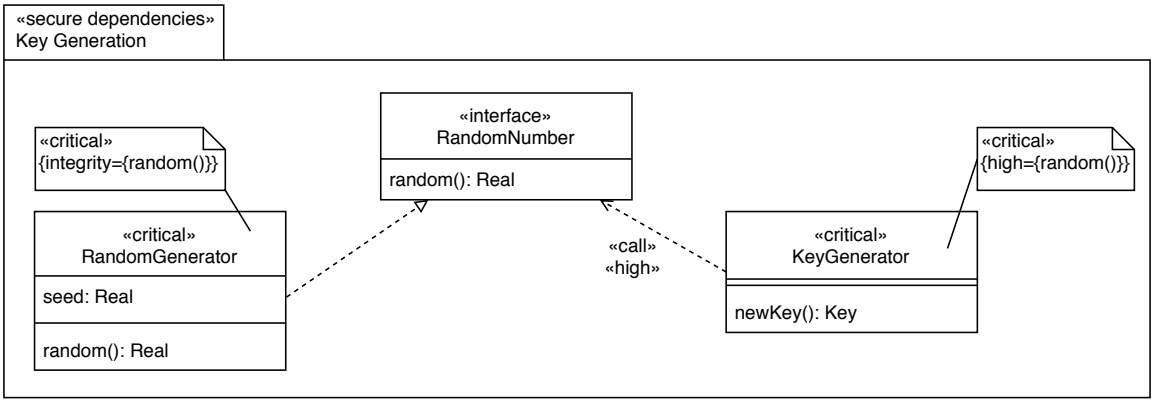

Abbildung 6.1.: Ein minimales Beispiel für ein Eingabemodell an die Secure-Dependencies-Analyse. Basierend auf [1, Seite 61].

des Rückgabewertes. Wenn z. B. *L* für die Rückgabewerte einer Methode einer Schnittstelle garantiert, dass die Vertraulichkeit gewahrt ist, wird der Name der Methode der Eigenschaftsdefinition `{Vertraulichkeit}` hinzugefügt. In Abbildung 6.1 ist dies an der Klasse *RandomGenerator* zu erkennen. Dieser realisiert die Schnittstelle *RandomNumber* und garantiert die Integrität der zurückgegebenen Werte der Methode *random()*. Daher verfügt diese Klasse über den Stereotyp «`critical`» mit dem Eigenschaftswert *{random()}* für die Eigenschaftsdefinition `{integrity}`.

Analog dazu können auch Sicherheitsanforderungen und -garantien für Parameter eines Methodenaufrufs definiert werden. In diesem Fall definiert die Klasse, die die Schnittstelle konsumiert, die Sicherheitseigenschaften der übergebenen Parameter und die Klasse, die die Schnittstelle realisiert, definiert die Sicherheitsanforderungen an die erwarteten Parameter. Dies wird ebenfalls über den Stereotyp «`critical`» annotiert und folgt dem beschriebenen Vorgehen für Rückgabewerte von Methoden. Die Semantik des Stereotyps und seinen Eigenschaftsdefinitionen muss daher aus dem Kontext ermittelt werden.

Schließlich verfügt eine Abhängigkeit einer Klasse K zu einer Schnittstelle I über Stereotypen $s \in \{$«`secrecy`», «`integrity`», «`high`»$\}$ genau dann, wenn die gleichnamige Eigenschaftsdefinition des Stereotyps «`critical`» von K Methodennamen enthält, die in I deklariert sind. Beispielsweise ist in Abbildung 6.1 an der Abhängigkeit von *KeyGenerator* zu *RandomNumber* der Stereotyp «`high`» annotiert, weil der Eigenschaftswert der Eigenschaftsdefinition `{high}` des Stereotyps «`critical`» der Klasse *KeyGenerator* die Methode *random()* enthält, welche in *RandomNumber* deklariert ist.

Im Folgenden wird die Analyse formal definiert. Zunächst ist zu erwähnen, dass es sich um eine rein statische Analyse handelt, die ohne eine Definition einer Angreifermächtigkeit auskommt. Die Einschränkung, die durch den Stereotyp «`secure dependencies`» realisiert wird, besagt, dass für jede «`call`» oder «`send`» Abhängigkeit von einem Objekt oder Subsystem C zu einer Schnittstelle I eines Objekts oder Subsystems D folgende Bedingungen erfüllt sein müssen. Nachrichtenname bezeichnet dabei eine UML-Nachricht,

wie in Kapitel 2 eingeführt, und kann entweder der Name einer Methode sein oder der Name eines Parameters einer Methode.

- Für jeden Nachrichtennamen *n* in *I* ist *n* genau dann Teil des Werts der Eigenschaftsdefinition `{secrecy}` (resp. `{integrity}`, resp. `{high}`) der Klasse *D*, wenn es auch Teil des Werts der gleichen Eigenschaftsdefinition in *C* ist.
- Wenn ein Nachrichtenname in *I* Teil des Werts der Eigenschaftsdefinition `{secrecy}` (resp. `{integrity}`, resp. `{high}`) der Klasse *C* ist, dann ist die Abhängigkeit von *C* zu *I* mit dem Stereotyp «`secrecy`» (resp. «`integrity`», resp. «`high`») annotiert.

Nachfolgend sind diese beiden Bedingungen anhand des Beispiels in Abbildung 6.1 dargestellt. Sei zunächst die zweite Bedingung betrachtet. Da *random()* Teil des Werts der Eigenschaftsdefinition `{high}` von *KeyGenerator* ist, ist die Abhängigkeit zwischen *KeyGenerator* und *RandomNumber* mit dem Stereotyp «`high`» annotiert. Diese Bedingung ist also erfüllt. Semantisch bedeutet dies, dass die Klasse *KeyGenerator* für mindestens ein Datum, das über diese Abhängigkeit kommuniziert wird, die Sicherheitsanforderung «`high`» fordert. Zur Ermittlung der ersten Bedingung müssen alle Abhängigkeiten, die mit dem Stereotyp «`call`» oder dem Stereotyp «`send`» annotiert sind, betrachtet werden. Davon existiert nur eine in dem Modell. Es muss also überprüft werden, dass wenn der Nachrichtenname *random()* aus *RandomNumber* in einem Wert einer Eigenschaftsdefinition von *KeyGenerator* vorkommt, er auch in demselben Eigenschaftswert von *RandomGenerator* vorkommt und umgekehrt. Dies ist hier nicht der Fall. Der Nachrichtenname *random()* ist Teil des Werts der Eigenschaftsdefinition `{high}` in *KeyGenerator* und Teil des Werts der Eigenschaftsdefinition `{integrity}` in *RandomGenerator*. Diese Bedingung ist also eingehalten, was dazu führt, dass die Secure-Dependencies-Analyse eine Sicherheitsverletzung erkennt. Das Problem ist, dass *KeyGenerator* beim Aufruf der Methode *random()* Daten erwartet, für die Vertraulichkeit und Integrität (high) garantiert sind, die Implementierung dieser Methode von der Klasse *RandomGenerator* aber nur die Integrität garantiert.

Es ist eine Einschränkung der Analyse und der Eingabemodelle zu erwähnen. Es ist möglich eine Klasse *K* zu haben, die von einer Schnittstelle *I* abhängt, aber auch die Schnittstelle *I* realisiert. In diesem Fall ist die Semantik der UMLsec-Stereotypen und -Eigenschaftsdefinitionen nicht eindeutig. Dieser Fall tritt ebenfalls ein, wenn zwei Methoden *m* und m' von jeweils der Schnittstelle *I* bzw. I' dieselbe Methodensignatur und einen identischen Namen besitzen, sowie eine Klasse *K* eine Abhängigkeitsbeziehung zu *I* und eine Realisierungsbeziehung zu I' hat. Da die Eigenschaftswerte der Eigenschaftsdefinitionen des Stereotyps «`critical`» lediglich Strings sind, kann daher nicht zwischen zwei Methoden gleichen Namens und gleicher Signatur unterschieden werden. Die Semantik dieser Situationen ist in dem Buch zu UMLsec [1] nicht definiert und kann auch nicht aus der Funktionsweise der Referenzimplementationen der Secure-Dependencies-Analyse erschlossen werden. Daher wird für die Spezifikation der Eingabemodelle und der Modelltransformationen davon ausgegangen, dass alle Methodennamen und Parameternamen des Eingabemodells eindeutig sind und diese Situationen nicht vorkommen können. Die Bedeutung für die Ausdrucksmächtigkeit ist in Kapitel 7 diskutiert.

In Abbildung 6.2 wird ein komplexeres Eingabemodell für die Secure-Dependencies-Analyse dargestellt, an dem die nachfolgende DCP-Analyse und die Modelltransformationen

sinnvoller erklärt werden können als an dem Minimalbeispiel in Abbildung 6.1. Das Modell besteht aus den drei Klassen *C1*, *C2*, *C3* und den beiden Schnittstellen *I1* und *I2*. Die Klasse *C2* realisiert beide Schnittstellen und garantiert, dass die Rückgabewerte der Methoden *m1()* und *m2()* der Schnittstelle *I1* vertraulich sind. Die Klasse *C1* ist von beiden Schnittstellen abhängig und fordert, dass für die Rückgabewerte der Methoden *m1()* und *m2()* Vertraulichkeit und für den Rückgabewert der Methode *m3(d : Data)* hohe Sensitivität garantiert ist. Weiter garantiert die Klasse *C1*, dass der Parameter *d* von hoher Sensitivität ist. Schließlich stellt Klasse *C3* die Schnittstelle *I2* bereit und hat eine Abhängigkeit auf die Schnittstelle *I1*. Die Klasse *C3* garantiert, dass die Rückgabedaten der Methode *m3(d : Data)* Daten hoher Sensitivität sind und fordert, dass die Rückgabewerte der Methode *m1()* vertraulich sind, sowie dass der Parameter *d* der Methode *m3(d : Data)* ein Datum hoher Sensitivität ist. An dieser Klasse ist zu erkennen, dass der Stereotyp «`critical`» sowohl zur Spezifizierung der Sicherheitsanforderungen, als auch zur Definition der Sicherheitseigenschaften der Daten verwendet wird.

Während der Analyse wird jedes Paar von Klassen, die über eine Abhängigkeitsbeziehung bzw. eine Realisierungsbeziehung zu derselben Schnittstelle verfügen, auf die zwei definierten Bedingungen überprüft. Die zu überprüfenden Paare sind nachfolgend dargestellt.

1. Die Klasse *C1* mit einer Abhängigkeit zu *I1* und die Klasse *C2* mit einer Realisierungsbeziehung zu *I1*. Es sind die Methoden *m1()* und *m2()* zu überprüfen.
2. Die Klasse *C1* mit einer Abhängigkeit zu *I2* und die Klasse *C2* mit einer Realisierungsbeziehung zu *I2*. Es sind die Methode *m3(d : Data)* und der Parameter *d* zu überprüfen.
3. Die Klasse *C1* mit einer Abhängigkeit zu *I2* und die Klasse *C3* mit einer Realisierungsbeziehung zu *I2*. Es sind die Methode *m3(d : Data)* und der Parameter *d* zu überprüfen.
4. Die Klasse *C3* mit einer Abhängigkeit zu *I1* und die Klasse *C2* mit einer Realisierungsbeziehung zu *I1*. Es sind die Methoden *m1()* und *m2()* zu überprüfen.

Für das erste Paar gilt, dass sowohl in *C1* als auch in *C2* die Methoden *m1()* und *m2()* die Werte der Eigenschaftsdefinition `{secrecy}` sind. Damit ist die erste Bedingung erfüllt. Ferner gilt, dass die Abhängigkeit über den Stereotyp *secrecy* verfügt, womit die zweite Bedingung erfüllt ist. Analog sind die beiden Bedingungen für das dritte Paar erfüllt, da in sowohl *C1* als auch *C3* die Methode *m3(d : Data)* und der Parameter *d* Werte der Eigenschaftsdefinition `{high}` sind. Anders ist es jedoch für das zweite und vierte Paar. Für das Zweite ist zu erkennen, dass für *C1* die Methode *m3(d : Data)* und der Parameter *d* Werte der Eigenschaftsdefinition `{high}` sind, aber dies nicht für *C2* gilt. Daher ist die erste Bedingung nicht erfüllt und die Analyse erkennt an dieser Stelle ein Sicherheitsproblem. Die Semantik des Problems ist, dass *C1* Daten von hoher Sensitivität erfordert, *C2* aber keinerlei Sicherheitseigenschaften bezüglich der Rückgabewerte von der Methode *m3(d : Data)* garantiert. Schließlich enthält in dem vierten Paar die Eigenschaftsdefinition `{secrecy}` von *C2* die Methoden *m1()* und *m2()*. Die Klasse *C3* hat als Werte für diese Eigenschaftsdefinition aber nur die Methode *m1()*. Daher schlägt an dieser Stelle die Analyse ebenfalls fehl. Die Semantik hiervon ist nicht definiert und aus dem Kontext unklar. Da *C3* keine Sicherheitsanforderungen an *m2()* stellt, ist anzunehmen, dass das Verhalten des

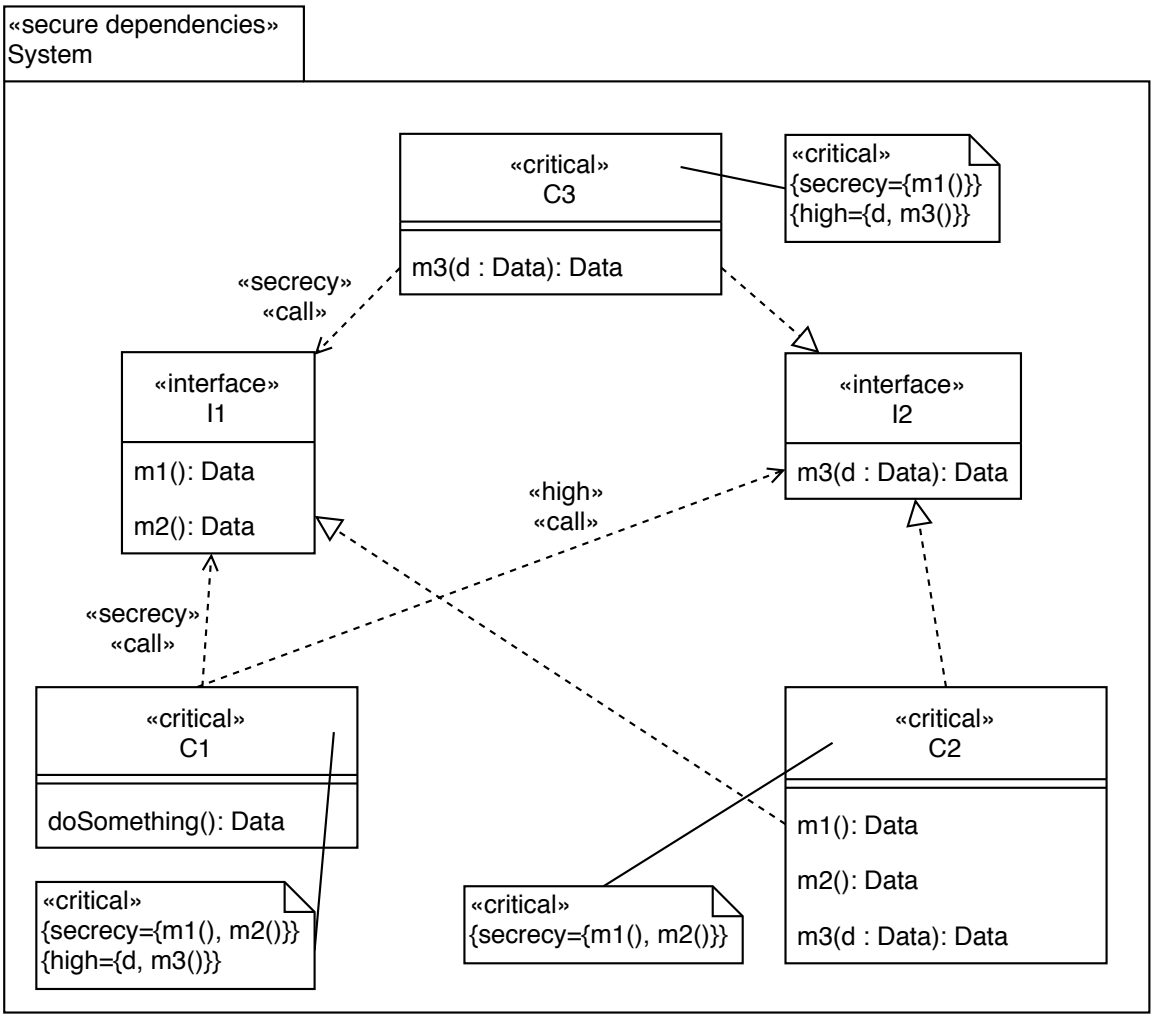

Abbildung 6.2.: Ein komplexeres Beispiel für ein Eingabemodell an die Secure-Dependencies-Analyse.

Systems nicht davon beeinflusst wird, dass *C2* an dieser Stelle mehr Sicherheitsgarantien erfüllt, als von *C3* benötigt. Im Folgenden wird davon ausgegangen, dass dies dennoch das gewünschte Verhalten der Analyse ist.

Des Weiteren ist zu erkennen, dass, wie bereits in der Secure-Links-Analyse, die Analyse eine Komposition der Analyse der einzelnen zu überprüfenden Klassen-Paaren ist. Das Beispielmodell aus Abbildung 6.2 kann also in ein Modell bestehend aus vier separate Subsystemen transformiert werden, ohne dass das Ergebnis der Analyse davon beeinflusst wird (siehe Abbildung 6.3). Diese Eigenschaft wird wieder dazu verwendet, separate Datenflüsse zu modellieren, um möglichst wenig zusätzliche Informationen in das DCP-Eingabemodell einzuführen. Im Folgenden werden die Gründe für die Auswahl der Secure-Dependencies-Analyse beschrieben und anschließend die DCP-Analyse, sowie die Modelltransformationen, erklärt.

6.1. Auswahlkriterien

Es existiert keine funktionsfähige Referenzimplementation der aktuellen Definition der Secure-Dependencies-Analyse. Daher ist Auswahlkriterium 1 nicht erfüllt. Die Definition der Secure-Dependencies-Analyse ist jedoch ausführlich genug, sodass eine Betrachtung der Analyse trotzdem möglich ist. Tatsächlich konnten keine Fälle identifiziert werden, in denen die Ergebnisse für ein gegebenes Modell nicht durch die Definition der Analyse des Buchs [1, Seite 59f] ermittelt werden können. Da die Analyse lediglich eine Iteration über alle Abhängigkeiten zwischen einem Subsystem oder Objekt zu einer Schnittstelle eines anderen Subsystems oder Objekts darstellt und für jede dieser Abhängigkeiten zwei wohldefinierte Bedingungen überprüft, sind die erwarteten Analyseergebnisse leicht ersichtlich. Auswahlkriterien 2 ist damit erfüllt.

Schließlich ist Auswahlkriterium 3 zu betrachten. Das Eingabemodell von der Secure-Dependency-Analyse ist ein Klassendiagramm. Es kann davon ausgegangen werden, dass die meisten Softwareentwickler mit Klassendiagrammen vertraut sind. In diesem Klassendiagramm gibt es bis auf die Stereotypen von UMLsec keine Modellelemente bei denen die Semantik unklar ist. Daher ist zu erwarten, dass die strukturellen Eigenschaften des Diagramms verständlich für beliebige Softwareentwickler sind. Die an Abhängigkeiten verwendeten Stereotypen und Eigenschaftsdefinitionen sind dieselben, die auch in Secure Links verwendet werden. Dabei handelt es sich um einfach nachzuvollziehende Konzepte, die zwar potentiell für Entwickler ohne kryptographische Vorkenntnisse eine ausführlichere Erklärung benötigen, aber bezüglich ihrer Semantik ist anzunehmen, dass eine textuelle Erklärung ausreichend ist, um die Intention der Analyse beliebigen Entwicklern verständlich zu machen. Wie bereits in Kapitel 5 erwähnt, ist eine detaillierte Betrachtung der Anwenderfreundlichkeit von kryptographischen Analysen außerhalb des Bearbeitungsumfangs dieser Arbeit. Unter der Annahme, dass die Einschätzung der Verständlichkeit der Sicherheitseigenschaften stimmt, handelt es sich bei Secure Dependencies also ebenfalls um eine Analyse, die für Anwender ohne kryptographische Vorkenntnisse zu bedienen ist.

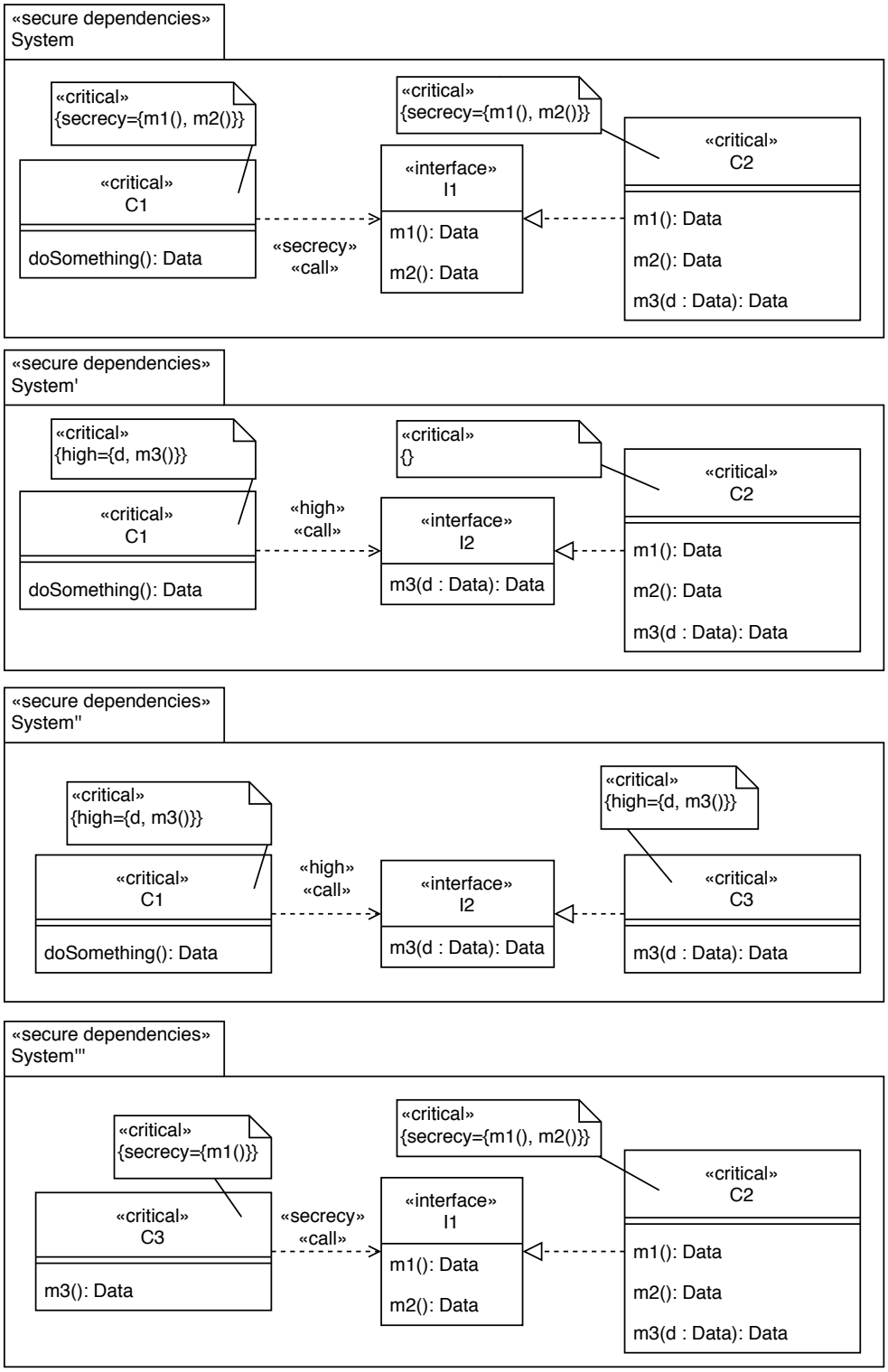

Abbildung 6.3.: Das Secure-Dependencies-Modell aus Abbildung 6.2 unterteilt in vier separate Subsysteme, sodass die Analyse weiterhin dasselbe Ergebnis liefert.

6.2. DCP-Analyse

Im Folgenden wird die Prolog-Abfrage vorgestellt, die Sicherheitsverletzungen nach Definition der Secure-Dependencies-Analyse erkennt. Analog zu Kapitel 5 wird zunächst die erwartete Instanz des Operationsmodells für ein Eingabemodell der Secure-Dependencies-Analyse betrachtet. Für die Secure-Dependencies-Analyse sind lediglich die Datenflüsse zwischen voneinander abhängigen Klassen relevant. In DCP ist dieses Konzept am ehesten durch einen externen Aufruf einer Methode einer Schnittstelle, die von einer anderen Komponente realisiert wird, repräsentiert. Der dazugehörige Datenverarbeitungsvorgang ist als *PerformDataTransmission* bezeichnet. Dieser Datenverarbeitungsvorgang transferiert für jeden Parameter der aufzurufenden Methode ein Datum an den Datenfluss der SEFF der Komponente, die die Schnittstelle realisiert, und erhält die Rückgabevariable von dieser SEFF. Analog zu der Secure-Links-Analyse liegt es nahe, die Sicherheitsgarantien der übertragenen Daten an den Daten selbst zu annotieren. Die Sicherheitsforderungen können jeweils an den Datenverarbeitungsvorgängen *PerformDataTransmission* und den Datenverarbeitungsvorgängen der aufgerufenen SEFF, die die Parameter des Aufrufs verwenden, annotiert werden. An den Datenverarbeitungsvorgängen *PerformDataTransmission* werden die Sicherheitsanforderungen an die Rückgabevariablen der aufgerufenen Methode definiert. An den Datenverarbeitungsvorgängen der aufgerufenen SEFF werden die Sicherheitsanforderungen an die jeweiligen Parameter der Methode annotiert. Die Idee dieser Analyse ist es, über alle Operationen im Operationsmodell zu iterieren, die einen solchen Datenverarbeitungsvorgang *PerformDataTransmission* oder eine aufgerufene SEFF repräsentieren und über annotierte Sicherheitsanforderungen an Rückgabevariablen oder Parameter verfügen. Anschließend kann für jede dieser Operationen überprüft werden, ob die geforderten Sicherheitseigenschaften den Sicherheitsgarantien des Datums entsprechen. Dies entspricht der ersten der beiden Bedingungen der Secure-Dependencies-Analyse. Die zweite Bedingung spezifiziert lediglich, dass wenn eine bestimmte Sicherheitsanforderung an die Rückgabevariable einer Methode gestellt ist, der entsprechende Stereotyp dieser Sicherheitsanforderung an die Abhängigkeitsbeziehung annotiert wird. Es handelt sich damit also mehr um eine syntaktische Bedingung als um eine semantische Bedingung. Ferner gilt, dass die DCP-Analyse Datenflüsse analysiert und nicht die Struktur der Eingabemodelle direkt und dass die Analyse daher eine solche syntaktische Bedingung nicht überprüfen kann. Aus diesen Gründen wird diese Bedingung nicht in der DCP-Analyse umgesetzt. Die Auswirkungen dieser Einschränkung auf die Ausdrucksmächtigkeit werden in Kapitel 7 diskutiert.

Ein Beispiel für eine Operationsmodellinstanz ist in Abbildung 6.4 dargestellt. Die Operation *PerformDataTransmissionOperation* repräsentiert den Aufruf einer Methode *m(a : Data, b : Data) : Data* und transferiert die Daten *a* und *b* an die Operation *ResourceDemandingSEFFOperation*, welche die SEFF repräsentiert, die das Verhalten der Methode *m* der aufgerufenen Komponente beschreibt. Die Daten *a* und *b* werden in weiteren Datenverarbeitungsvorgängen weiterverwendet. Dies ist hier durch die Operationen *SelectionDataOperation1* und *SelectionDataOperation2* dargestellt. Diese Operationen sind mit den Sicherheitsanforderungen an den Parameter *a* bzw. *b* annotiert. Sobald alle Datenabhängigkeiten der Operation *ResourceDemandingSEFFOperation* aufgelöst sind, wird

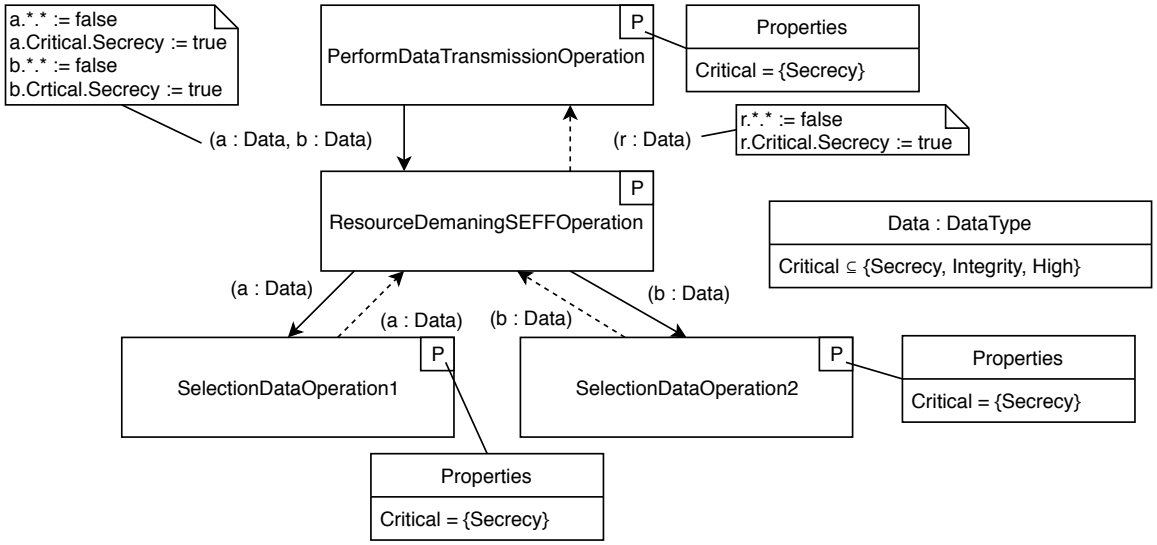

Abbildung 6.4.: Ein Beispiel einer Operationsmodellinstanz zur Verwendung mit der Secure-Dependencies-Analyse.

die Rückgabevariable *r* an die Operation *PerformDataTransmissionOperation* zurückgegeben. An dieser Operation sind die Sicherheitsanforderungen an die Rückgabevariable *r* annotiert. Es ist anzumerken, dass es im Gegensatz zu dem korrespondierenden UMLsec-Modell hier nicht notwendig ist, die Namen der zu überprüfenden Daten oder Methoden zu annotieren. Da ein Datenverarbeitungsvorgang vom Typ *PerformDataTransmission* den Aufruf an eine bestimmte Methode darstellt, ist die aufgerufene Methode, für deren Rückgabevariable die Sicherheitsanforderungen gelten, darüber eindeutig definiert. Selbiges gilt für die Parameter der SEFF, welche das Verhalten einer Methode beschreibt.

Im Folgenden sind die Prolog-Regeln definiert, aus denen die Secure-Dependencies-Sicherheitsanalyse in DCP besteht. Den Einspringpunkt in die Analyse stellt das Prädikat *secureDependencies\4* dar. Es ist in Algorithmus 6.1 dargestellt. Dieses Prädikat wird als Generator verwendet und liefert eine Parameterbelegung für jede im Modell vorhandene Sicherheitsverletzung der Secure-Dependencies-Bedingungen. Die Parameter des Prädikats dienen dazu, dem Nutzer Informationen über den Grund des Sicherheitsproblems zu liefern. Der Parameter *OP* beschreibt die Operation des Operationsmodells, an dem eine Sicherheitsverletzung ermittelt wurde und der Parameter *V* beschreibt die Sicherheitseigenschaft, die für den Fehler verantwortlich ist. Parameter *R* ist die untersuchte Rückgabevariable und *S* ist die Aufrufhierarchie. Diese beiden Parameter dienen nur dazu, die Modellelemente zu ermitteln, an denen die Sicherheitsverletzung auftritt. Wenn z. B. an einer Operation gefordert ist, dass die zurückerhaltenen Daten vertraulich sind, aber die Daten nur die Integrität garantieren, werden zwei Sicherheitsverletzungen zu dieser Operation geliefert, eine mit $V = Secrecy$ und eine weitere mit $V = Integrity$. Nachfolgend ist die Funktionsweise des Prädikats erklärt. Zunächst werden in Zeile 2 und 3 alle Operationen generiert, die über die Eigenschaft *Critical* verfügen. Anschließend wird in Zeile 4 und 5 *S* die Aufrufhierarchie zugewiesen und sichergestellt, dass es sich um eine gültige Aufrufhierarchie handelt. In Zeile 6 wird durch eine Abfrage an das Prädikat *ope-*

```
1 secureDependencies(OP, S, R, V) :-
2     isOperation(OP),
3     hasProperty(OP, 'Critical'),
4     S=[OP|_],
5     stackValid(S),
6     operationReturnValue(OP, R),
7     secureDependenciesViolation(OP, S, R, V).
```

Algorithmus 6.1.: Das Prädikat *secureDependencies\4* der Secure-Dependencies-Analyse.

```
1 secureDependenciesViolation(OP, S, R, V) :-
2     valueSetMember('CriticalLiteral', V),
3     operationProperty(OP, 'Critical', V),
4     lnot(returnValue(S, R, 'Critical', V)).
5
6 secureDependenciesViolation(OP, S, R, V) :-
7     valueSetMember('CriticalLiteral', V),
8     lnot(operationProperty(OP, 'Critical', V)),
9     returnValue(S, R, 'Critical', V).
```

Algorithmus 6.2.: Das Prädikat *secureDependenciesViolation\4* der Secure-Dependencies-Analyse.

rationReturnValue\2 eine Rückgabevariable der Operation *R* zugewiesen. Schließlich wird durch das Prädikat *secureDependenciesViolation\4* überprüft, ob für die Rückgabevariable *R* der Operation *OP* unter der Aufrufhierarchie *S* eine Sicherheitsverletzung der Secure-Dependencies-Bedingung vorliegt.

Das Prädikat *secureDependenciesViolation\4* ist in Algorithmus 6.2 dargestellt. Es erhält als Eingabe eine Operation *OP*, eine Aufrufhierarchie *S* und eine Rückgabevariable *R* der Operation *OP*. Falls eine Sicherheitsverletzung vorliegt, weist das Prädikat *V* den Wert der verantwortlichen Sicherheitseigenschaft zu. In Zeile 2 wird *V* ein möglicher Aufzählungswert der Eigenschaft *Critical* zugewiesen und dadurch nacheinander jeweils die in Zeile 3 und 4 definierte Bedingung für die Werte *Secrecy*, *Integrity*, und *High* überprüft. Zeile 3 ist genau dann wahr, wenn der Wert der aktuell betrachteten Sicherheitseigenschaft *V* als Eigenschaftswert von der *Critical*-Eigenschaft der Operation *OP* vorhanden ist. Zeile 4 ist genau dann wahr, wenn der Wert der aktuell betrachteten Sicherheitseigenschaft *V* als Attributswert von dem *Critical*-Attribut der Rückgabevariable *R* nicht vorhanden ist. Gemäß der definierten Bedingung stellt dies eine Verletzung dar, da gefordert ist, dass wenn eine der Sicherheitseigenschaften in der konsumierenden Klasse vorhanden ist, diese Sicherheitseigenschaft auch in der bereitstellenden Klasse vorhanden sein muss und umgekehrt. Zeile 6 beschreibt dabei die umgekehrte Richtung, in der eine Sicherheitseigenschaft auf dem betrachteten Datum vorhanden ist, aber auf der Operation fehlt. Das Prädikat *lnot\1* ist Teil der Prolog-Schnittstelle von DCP und stellt das logische Nicht dar.

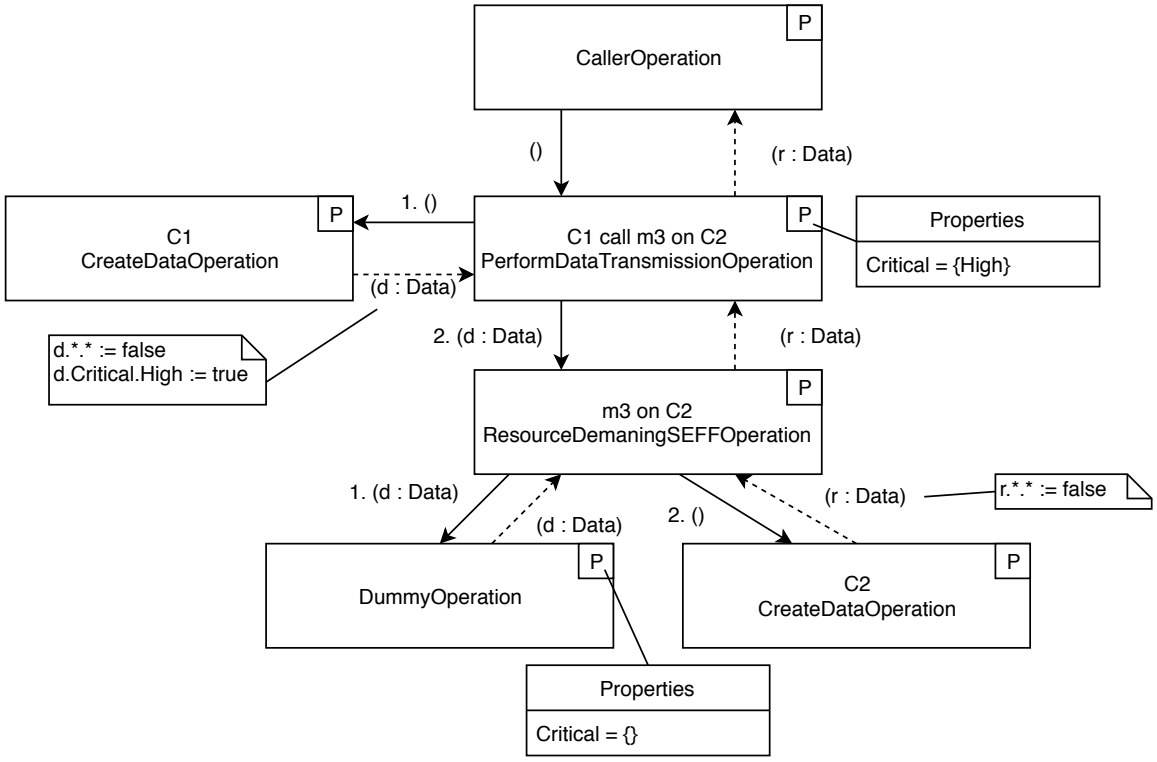

Abbildung 6.5.: Ein vereinfachter Auszug der Operationsmodellinstanz, die zu dem Beispiel aus Abbildung 6.2 korrespondiert.

Der Ablauf der Analyse wird im Folgenden anhand des in Abbildung 6.2 gegebenen Beispiels dargestellt. Hierfür wird der Kürze halber nur ein vereinfachter Auszug aus der korrespondierenden Operationsmodellinstanz verwendet. Das vollständige Modell ist als Datensatz verfügbar [42]. Das vereinfachte Operationsmodell ist in Abbildung 6.5 dargestellt. Es handelt sich dabei um den Auszug, in dem die Abhängigkeit zwischen $C1$ und $I2$ und die Realisierungsbeziehung zwischen $I2$ und $C2$ Gegenstand der Analyse sind. Die Schnittstelle definiert die Methode $m3(d : Data) : Data$. Die Klasse $C1$ fordert, dass für den Rückgabewert von $m3$ hohe Sensitivität garantiert ist. Weiter garantiert C, dass für den Parameter d hohe Sensitivität vorliegt. Die Klasse $C2$ verfügt über keine Sicherheitsanforderungen oder -garantien. Daher wird im Operationsmodell in Abbildung 6.5 das Datum d mit dem Attributswert *High* erzeugt und die *PerformDataTransmissionOperation* von $C1$ verfügt über den Eigenschaftswert *High*.

Die Analyse läuft folgendermaßen ab. Zunächst wird in dem Prädikat *secureDependencies\4* der Parameter *OP* gebunden, indem *OP* eine Operation zugewiesen wird, die über die Eigenschaft *Critical* verfügt (siehe Algorithmus 6.1, Zeile 2 und 3). Sei im Folgenden zunächst die Operation *PerformDataTransmissionOperation* betrachtet. Anschließend wird für eine Rückgabevariable von dieser Operation die Bedingung überprüft. Die Operation verfügt nur über die Rückgabevariable r. Für diese wird die Bedingung, die in dem Prädikat *secureDependenciesViolation\4* definiert ist, geprüft. Dabei wird zunächst $V = Secrecy$ zugewiesen. Da r über keinen Attributswert *Secrecy* des Attributs *Critical* verfügt und die

Operation *PerformDataTransmissionOperation* auch nicht den Eigenschaftswert *Secrecy* der Eigenschaft *Critical* besitzt, schlägt für $V = Secrecy$ dieses Prädikat fehl. Anschließend wird dasselbe für $V = Integrity$ und $V = High$ geprüft. Für $V = Integrity$ schlägt die Regel ebenfalls fehl. Für $V = High$ dagegen gilt, dass die Operation über den Eigenschaftswert *High* verfügt, die Rückgabevariable *r* dagegen nicht. Damit ist eine Parameterbelegung gefunden, in der eine Sicherheitsverletzung der Secure-Dependencies-Bedingung vorliegt. Analog funktioniert die Überprüfung bei der Operation *DummyOperation*. Diese Operation dient dazu zu überprüfen, ob der Parameter *d* den Sicherheitsanforderungen entspricht. Für $V = High$ liegt ebenfalls eine Sicherheitsverletzung vor, da *d* über den Attributswert *High* verfügt, die Operation jedoch keine Sicherheitsgarantien erwartet. Die Secure-Dependencies-Analyse erkennt für dieses Beispiel also zwei Sicherheitsverletzungen.

An dieser Stelle ist zu erkennen, dass sowohl für die Überprüfung der Rückgabewerte eines Methodenaufrufs, als auch für die Parameter an diesen Methodenaufruf, im Operationsmodell Rückgabevariablen geprüft werden. Dies ist auf eine Limitierung des Operationsmodells zurückzuführen. Parameter einer Methode werden im Operationsmodell als Zustandsvariablen der Operation *ResourceDemandingSEFFOperation* modelliert, die die SEFF repräsentiert, die die entsprechende Methode beschreibt. Diese Zustandsvariablen werden nicht als Parameter an weiter aufgerufene Operationen gegeben, sondern sind global verfügbar. Das bedeutet z. B., dass der dargestellte Parameter *d* der Operation *DummyOperation* in einer Abfrage an das Operationsmodell nur erreicht werden kann, wenn von der Operation *DummyOperation* zu der Operation *ResourceDemandingSEFF-Operation* traversiert wird und von dieser Operation die Zustandsvariablen betrachtet werden. Dies ist generell möglich, da die Aufrufhierarchie bekannt ist. Es ist jedoch für diese Analyse nicht anwendbar. Da die Operation *ResourceDemandingSEFFOperation* über mehrere Parameter und damit Zustandsvariablen verfügen kann, ist eine Zuordnung von Parametern zu Sicherheitsforderungen notwendig. Da sich Zustandsvariablen im Operationsmodell nur durch ihren eindeutigen Namen unterscheiden, ist diese Zuordnung allerdings nicht typsicher möglich. Deswegen wird im Kontext der Secure-Dependencies-Analyse folgende Konstruktion verwendet: Für jeden Parameter, den eine Operation, die eine SEFF repräsentiert, erhält, wird ein Datenverarbeitungsvorgang explizit modelliert, der die Sicherheitsanforderungen an diesen Parameter festlegt. In dem Beispiel ist dies die Operation *DummyOperation*. Für die Modellierung in DCP wird dafür ein Datenverarbeitungsvorgang vom Typ *SelectData* verwendet, der eine Kopie des Parameters erstellt und zurückgibt. Für die umgekehrte Richtung existiert dieses Problem nicht, da die Rückgabevariablen des Methodenaufrufs immer ein Teil der entsprechenden Operation *PerformDataTransmissionOperation* sind. Abschließend ist zu erwähnen, dass die Einführung dieser Operationen, die nur zur Überprüfung der Sicherheitseigenschaften von Parametern dienen, nur für die Modelle relevant ist, die aus entsprechenden UMLsec-Modellen transformiert wurden. In Modellen, in denen komplexe Datenflüsse beschrieben sind, kann die Eigenschaft *Critical* an bereits existierende Datenverarbeitungsvorgänge annotiert werden, die das entsprechende Datum, das überprüft werden soll, verwenden.

6.3. Modelltransformation

Im Folgenden sind die Modelltransformationen des UMLsec-Eingabemodells auf das DCP-Eingabemodell beschrieben. Dabei sollen die Modelltransformationen die Semantik des Eingabemodells erhalten und ein DCP-Eingabemodell erzeugen, das eine Operationsmodellinstanz erzeugt, sodass die beschriebene DCP-Analyse möglich ist. Zunächst wird die Transformation von UMLsec zu DCP erklärt und anschließend die Transformation von DCP zu UMLsec beschrieben.

6.3.1. UMLsec zu Data-Centric Palladio

Analog zu der Beschreibung der Modelltransformation für die Secure-Links-Analyse wird im Folgenden die Beschreibung der Transformation anhand der entstehenden einzelnen Modelle des PCM strukturiert. Dabei wird zunächst auf das Komponentenverzeichnis eingegangen, und anschließend auf das Systemmodell, die Ausführungsumgebung und die Allokation. Abschließend werden die Datenflüsse und Charakteristiken definiert.

6.3.1.1. Komponentenverzeichnis

Die Transformation erzeugt zunächst aus dem UMLsec-Eingabemodell das Komponentenverzeichnis des DCP-Eingabemodells. Dazu ist zunächst zu erwähnen, dass die Semantik von Klassen nicht direkt in DCP abbildbar ist. Daher werden für die Zwecke dieser Analyse Klassen eines UMLsec-Eingabemodells auf Komponenten in DCP abgebildet. Diese verfügen ebenfalls über die Möglichkeit, Abhängigkeitsbeziehungen zu Schnittstellen zu definieren und stellen daher für die Umsetzung der Analyse in DCP ein sinnvolles Substitut für Klassen dar.

Zunächst wird für jede Schnittstelle I_{UMLsec} des UMLsec-Eingabemodells eine Schnittstelle (operation interface) I_{DCP} in DCP erstellt. Für jede Methode von I_{UMLsec} wird eine Methode (operation signature) in I_{DCP} erzeugt. Dabei wird die Methodensignatur in DCP nachgebildet, also jeder Parameter der UMLsec-Methode erzeugt einen Parameter äquivalenten Typs in DCP und wenn I_{UMLsec} über einen Rückgabewert verfügt, erhält I_{DCP} ebenfalls einen Rückgabewert äquivalenten Typs. Gegebenenfalls müssen hierfür zusätzliche Datentypen (composite data type) in dem Komponentenverzeichnis des DCP-Eingabemodells erzeugt werden, sofern es sich bei den in UMLsec spezifizierten Datentypen nicht um Datentypen handelt, die als primitive Datentypen in DCP vorhanden sind. Abschließend wird jede Schnittstelle mit dem DCP-Stereotyp *OperationSignatureRefinement* annotiert, um die zwischen Komponenten kommunizierten Daten im Datenflussmodell spezifizieren zu können.

Weiter wird für jede Klasse C_{UMLsec} des UMLsec-Eingabemodells eine Komponente C_{DCP} im DCP-Eingabemodell erzeugt. Für jede Abhängigkeit von C_{UMLsec} zu einer Schnittstelle I_{UMLsec} wird in C_{DCP} eine benötigte Rolle (operation required role) erzeugt, die als benötigtes Interface I_{DCP} spezifiziert. Dabei korrespondiert die Schnittstelle I_{DCP} zu der

Schnittstelle I_{UMLsec}. Für jede Schnittstelle I_{UMLsec}, die von C_{UMLsec} realisiert wird, wird in C_{DCP} eine bereitgestellte Rolle (operation provided role) definiert, die als bereitgestellte Schnittstelle I_{DCP} hat. Des Weiteren wird für jede Methode dieser Schnittstelle I_{DCP} eine RDSEFF definiert, die das Verhalten dieser Methode beschreibt. Diese RDSEFF besteht nur aus einer internen Aktion (internal action), welche mit dem DCP-Stereotyp «`DataProcessingSpezification`» annotiert ist, um bei der Erstellung der Datenflüsse den Datenflusscontainer dieser Aktion zu spezifizieren.

Um die DCP-Analyse durchzuführen wird ein Datenfluss benötigt, der für eine Komponente *C* jede Methode jeder Schnittstelle, von der *C* abhängt, einmal aufruft. Es kann an dieser Stelle weder ein semantisch sinnvoller Datenfluss, noch ein semantisch sinnvolles Nutzungsprofil aus dem UMLsec-Modell extrahiert werden, da diese Informationen im rein strukturellen UMLsec-Modell nicht verfügbar ist. Daher wird eine zusätzliche Schnittstelle *Check* in DCP eingeführt. Diese Schnittstelle verfügt über eine Methode *check()* ohne Parameter und mit einem Rückgabewert eines beliebigen Datentyps. Jede Komponente, die über mindestens eine benötigte Rolle verfügt, erhält zusätzlich eine bereitgestellte Rolle, die als bereitgestellte Schnittstelle die Schnittstelle *Check* spezifiziert. Anschließend wird für jede Komponente *C*, die die Schnittstelle *Check* bereitstellt, eine RDSEFF erzeugt, die das Verhalten der Methode *check()* definiert. Diese RDSEFF verfügt über jeweils einen externen Aufruf (external call action) für jede Methode jeder Schnittstelle, die von *C* benötigt wird. Die externen Aufrufe werden mit dem DCP-Stereotyp «`DataProcessingSpezification`» annotiert, um später Datenflusscontainer für diese Aufrufe zu spezifizieren. Die Komponenten, die diese Schnittstelle bereitstellen, stellen durch die Methode *check()* den Einspringpunkt der Datenflüsse innerhalb des Systems dar und werden von den später definierten Nutzungsszenarien aufgerufen. Ein Beispiel eines Komponentenverzeichnisses ist in Abbildung 6.6 dargestellt. Dabei dient das in Abbildung 6.2 vorgestellte Beispiel als das korrespondierende UMLsec-Eingabemodell. Aus Platzgründen sind die Aktivitätsdiagramme der SEFFs nicht dargestellt. Stattdessen ist in Abbildung 6.7 jeweils ein Beispiel dargestellt für eine SEFF, die die Methode *check()* beschreibt und eine SEFF, die eine Methode beschreibt, die bereits als Teil einer Schnittstelle im korrespondierenden UMLsec-Eingabemodell gegeben ist.

Abschließend ist zu erwähnen, dass bei dieser Transformation Informationen über von Klassen implementierte Methoden, die nicht über eine Schnittstelle zur Verfügung gestellt werden, verloren gehen. Dies ist auf den Abstraktionsunterschied zwischen Klassen und Komponenten zurückzuführen. Da diese Methoden für die Analyse jedoch irrelevant sind, wird an dieser Stelle akzeptiert, dass diese Informationen während der Transformation verloren gehen.

6.3.1.2. Systemmodell, Ausführungsumgebung und Allokation

Um ein Systemmodell aus dem UMLsec-Eingabemodell zu erzeugen, wird analog zu Secure Links zunächst ein Zwischenmodell erzeugt, anhand dessen die Komponenteninstanzen (assembly context) und Komponenteninstanzkonnektoren (assembly connector) direkt abgelesen werden können. Aus jeder Klasse des UMLsec-Eingabemodells wird eine DCP-

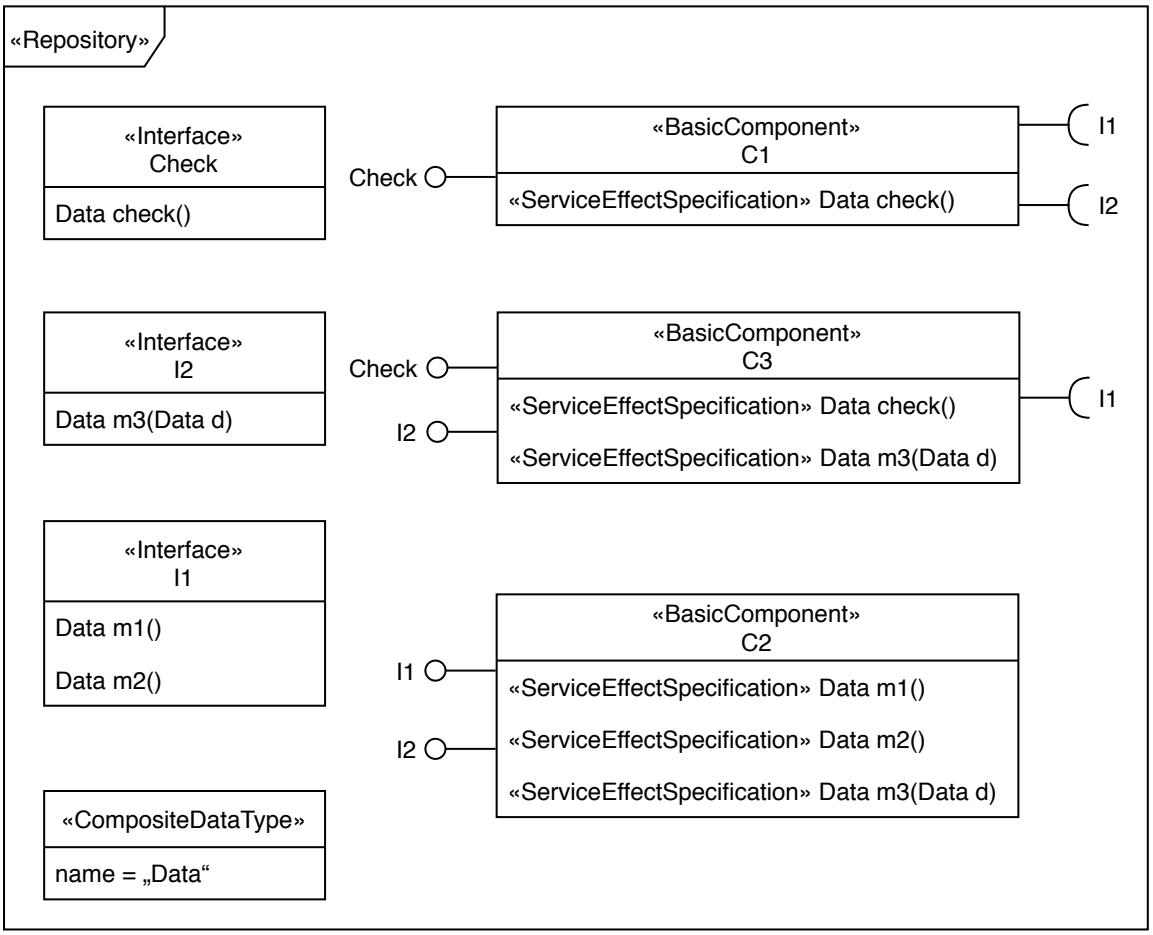

Abbildung 6.6.: Das durch die Modelltransformation entstandene Komponentenverzeichnis, welches zu Abbildung 6.2 korrespondiert.

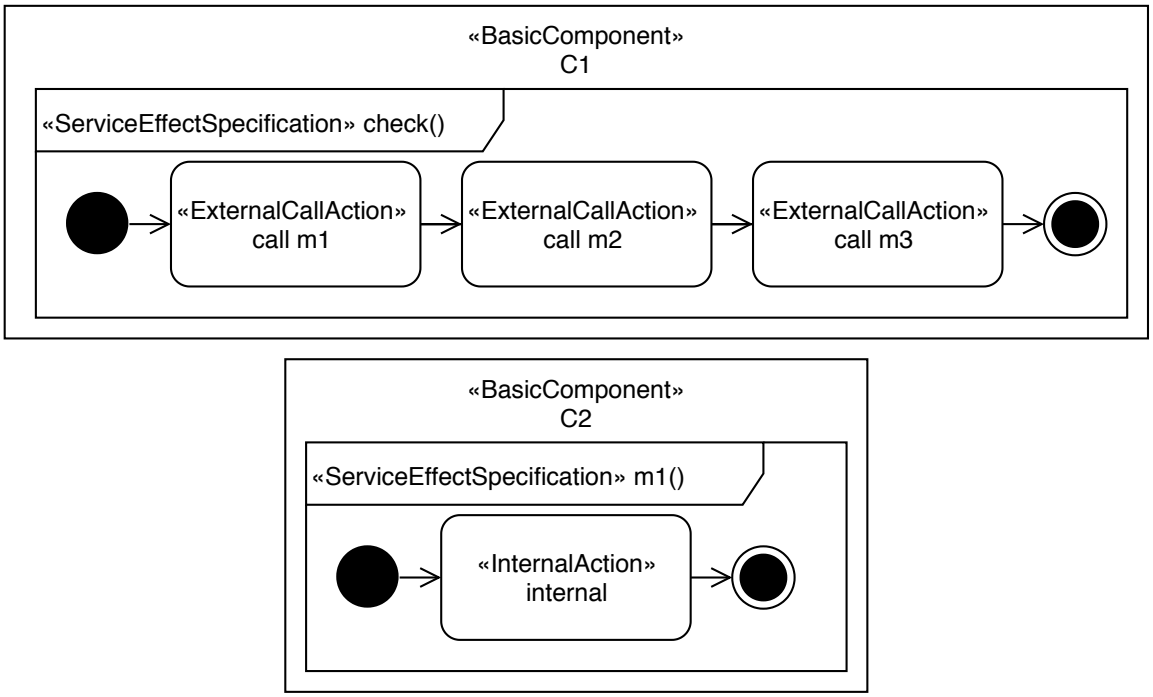

Abbildung 6.7.: Beispiele für die SEFFs der Methode *check()* und *m1()*.

Komponenteninstanz erzeugt, deren Komponente für jede Abhängigkeitsbeziehung an eine Schnittstelle I_{UMLsec} eine benötigte Rolle mit der Schnittstelle I_{DCP} und für jede Realisierungsbeziehung an die Schnittstelle J_{UMLsec} eine bereitgestellte Rolle mit der Schnittstelle J_{DCP} definiert. Zusätzlich erhält jede Komponente, die über mindestens eine benötigte Rolle verfügt, eine bereitgestellte Rolle mit der Schnittstelle *Check*. Dies entspricht dem Vorgehen der Erzeugung des Komponentenverzeichnisses, mit dem Unterschied, dass an dieser Stelle auf Instanzebene operiert wird und nur die Darstellung der Komponenteninstanzen relevant ist. Die Komponenteninstanzkonnektoren werden entsprechend der Abhängigkeitsbeziehungen und Realisierungsbeziehungen erstellt. Für jedes Paar von Klassen C_{UMLsec} und D_{UMLsec}, wovon C_{UMLsec} eine Realisierungsbeziehung und D_{UMLsec} eine Abhängigkeitsbeziehung an die Schnittstelle I_{UMLsec} besitzen, wird ein Komponenteninstanzkonnektor zwischen der bereitgestellten Rolle der Komponenteninstanz C_{DCP} für Schnittstelle I_{DCP} und der benötigten Rolle der Komponenteninstanz D_{DCP} für I_{DCP} erzeugt. Dies ist in Abbildung 6.8 dargestellt. Es kann hierbei der bereits bekannte Fall eintreten, dass eine benötigte Rolle einer Komponenteninstanz einen Konnektor zu mehreren bereitgestellten Rollen verschiedener anderer Komponenteninstanzen besitzt. Wie bereits in Kapitel 5 beschrieben, handelt es sich hierbei um eine Verletzung einer Invariante des Systemmodells. Deswegen wird in einem nächsten Transformationsschritt für jede Komponenteninstanz C, die mehrere Konnektoren für eine ihrer benötigten Rollen besitzt, für jeden zusätzlichen Konnektor k_i eine weitere Komponenteninstanz C_i erzeugt. Jeder der zusätzlichen Konnektoren k_i wird anschließend geändert, sodass er die benötigte Rolle von C_i als Ziel hat. Anschließend wird für jede weitere benötigte Rolle einer der neuen Komponenteninstanzen C_i ein neuer Konnektor erzeugt, der die gleiche Quelle hat, wie der

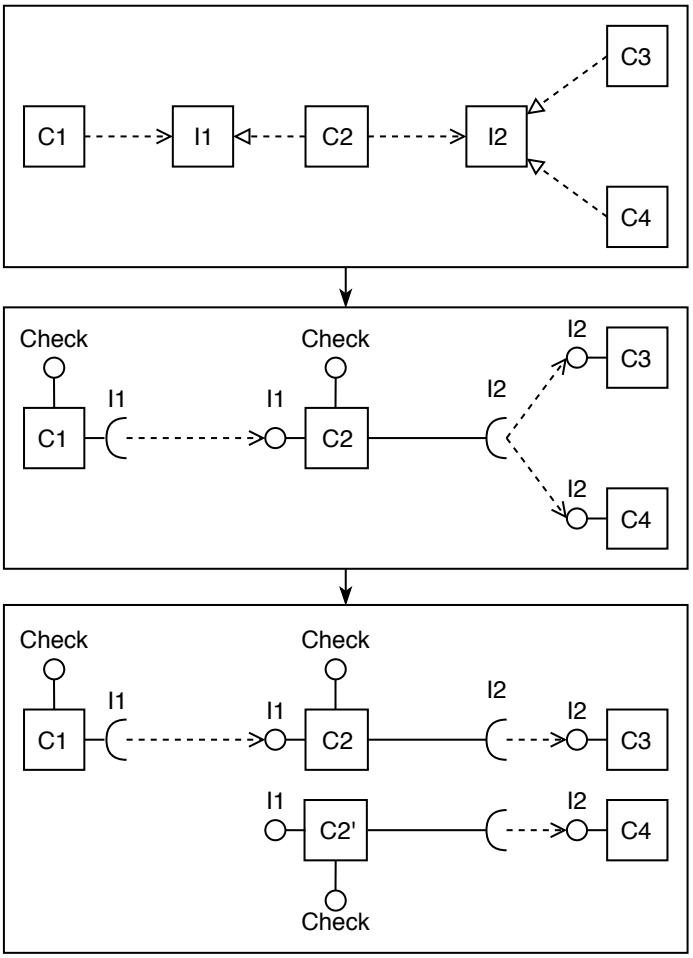

Abbildung 6.8.: Beispiel für die Erzeugung des Zwischenmodells, das zur Erzeugung des Systemmodells verwendet wird.

Konnektor der diese benötigte Rolle für Komponenteninstanz C mit einer bereitgestellten Rolle verbindet. Wenn die Komponenteninstanzen C_i über bereitgestellte Rollen verfügen, werden diese nicht mit anderen Komponenteninstanzen verbunden. Dieser Transformationsschritt ist ebenfalls in Abbildung 6.8 dargestellt. Diese Darstellung entspricht nun den zu erzeugenden Komponenteninstanzen und Komponenteninstanzkonnektoren. Schließlich wird für jede Komponenteninstanz, die die Schnittstelle *Check* zur Verfügung stellt, eine bereitgestellte Rolle (operation provided role) im Systemmodell und ein Konnektor (provided delegation connector), der diese Rolle mit der entsprechenden Komponenteninstanz verbindet, erstellt. Diese dienen als Einspringpunkt für die Nutzungsszenarien und bilden damit die Startpunkte des Datenflusses innerhalb des Systems.

Da diese Analyse keinerlei Aussage über Allokation oder Ausführungsumgebung macht, wird nur ein einziger Rechnerknoten im DCP-Modell erzeugt, auf den jede Komponenteninstanz alloziert wird. Ein Beispiel für Systemmodell, Ausführungsumgebung und Allokation ist in Abbildung 6.9 dargestellt. Dieses Modell korrespondiert zu dem in Abbildung 6.2 vorgestellten Beispiel.

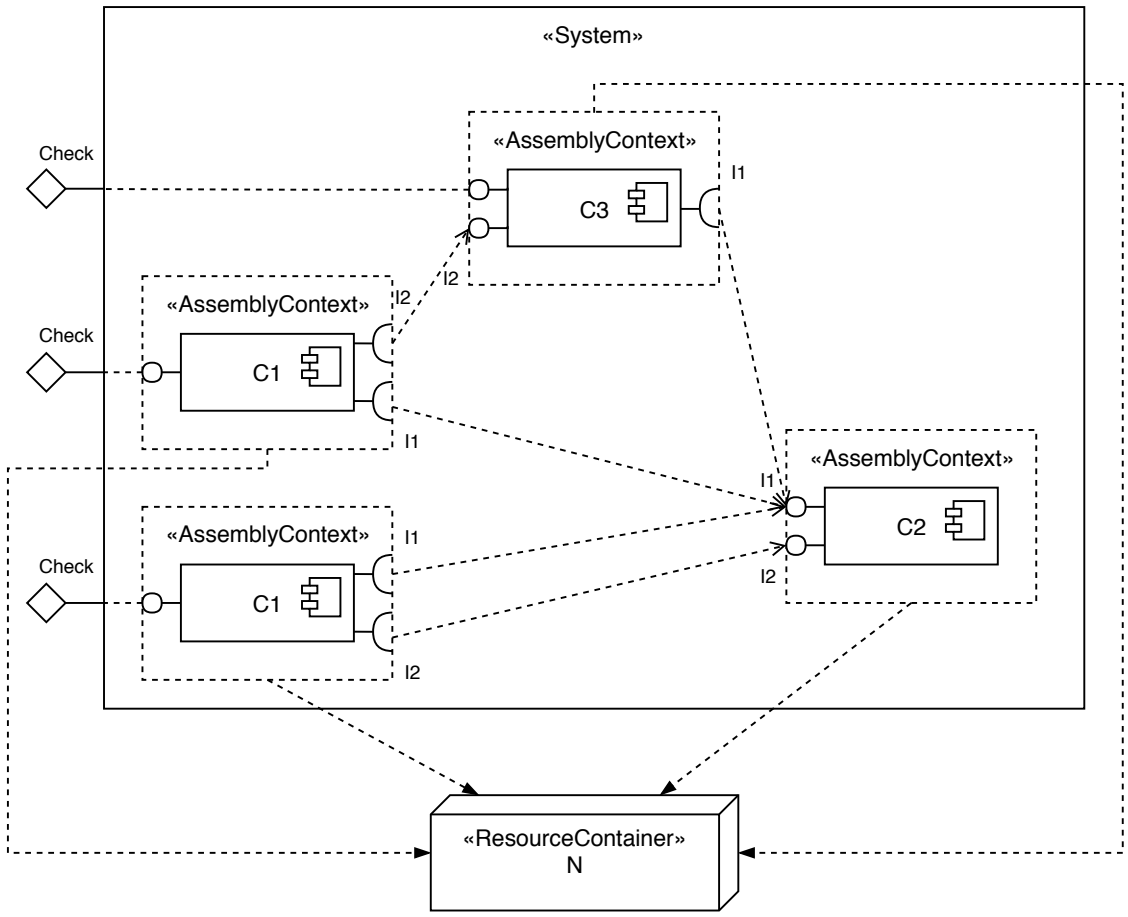

Abbildung 6.9.: Das Systemmodell, die Ausführungsumgebung und das Verteilungsmodell, welche zu Abbildung 6.2 korrespondieren.

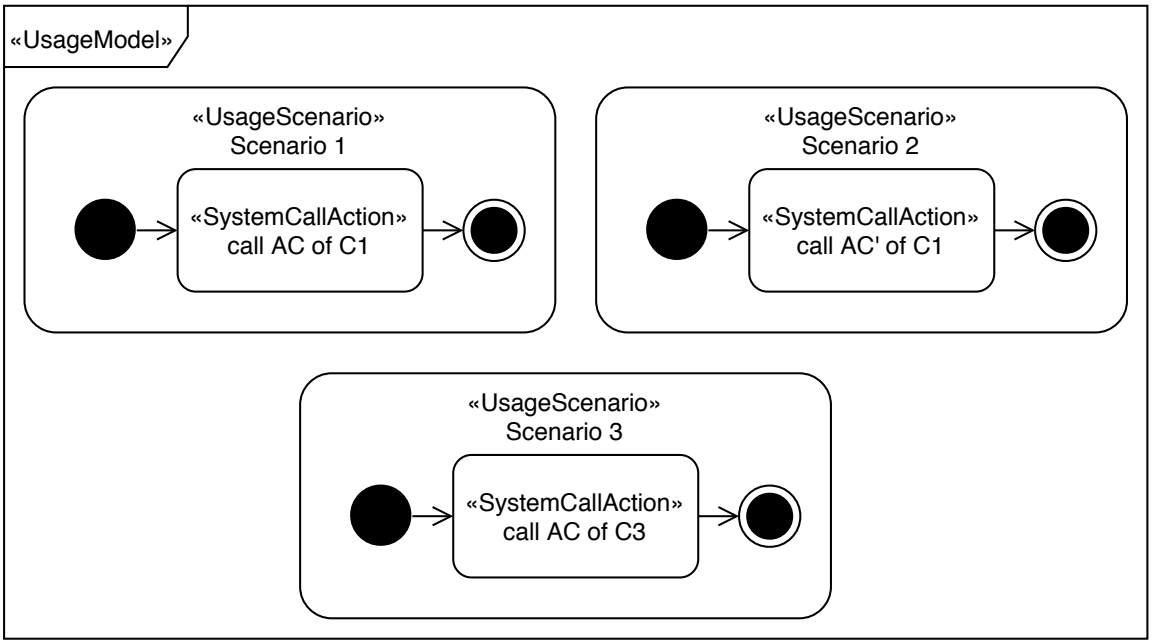

Abbildung 6.10.: Das Nutzungsmodell, das für das Beispiel aus Abbildung 6.2 erzeugt wird.

6.3.1.3. Nutzungsmodell

Das Nutzungsmodell hat kein Gegenstück in dem UMLsec-Eingabemodell der Secure-Dependencies-Analyse und wird im Folgenden als Ausgangspunkt für Datenflüsse spezifiziert. Die Erstellung des Nutzungsmodells erfolgt analog zur Modelltransformation für die Secure-Links-Analyse. Während der Erzeugung des Systemmodells wird für jede Komponente, die die Schnittstelle *Check* zur Verfügung stellt, eine vom System bereitgestellte Rolle erzeugt. Für jede dieser Rollen r wird ein Nutzungsszenario (usage scenario) definiert, das einen Systemaufruf (entry level system call) an r durchführt. Diese Systemaufrufe werden zur späteren Spezifizierung des Datenflusses mit dem DCP-Stereotyp «`DataProcessingSpecification`» annotiert. In Abbildung 6.10 ist ein Beispiel für ein erstelltes Nutzungsmodell dargestellt.

6.3.1.4. Datenflussmodell

Zunächst werden die notwendigen Charakteristiken betrachtet. Es wird der Charakteristiktyp *Critical* erstellt, der über die Werte *Secrecy*, *Integrity* und *High* verfügt. Die Semantik dieser Werte entspricht der Definition in Kapitel 2. Analog zur UMLsec-Analyse wird dieser Charakteristiktyp sowohl dazu verwendet, die Sicherheitsgarantien, als auch die Sicherheitsanforderungen an ein Datum zu spezifizieren.

Anschließend wird für jede Methode jeder Schnittstelle des Komponentenverzeichnisses eine Methodensignaturverfeinerung erstellt. Wie schon bei der Secure-Links-Analyse wird diese im Kontext der Secure-Dependencies-Analyse nur benötigt, um ein gültiges

Datenflussmodell zu erstellen und nicht um eine tatsächliche Verfeinerung der Daten zu spezifizieren. Für jeden Parameter einer Methode, für die eine Verfeinerung erstellt wird, wird ein parameterbasiertes Datum (parameter based data) erzeugt. Wenn die Methode über einen Rückgabewert verfügt, wird zusätzlich ein rückgabebasiertes Datum (return based data) erzeugt. Die Typen dieser Daten entsprechen den Typen der Parameter bzw. Rückgabewerte.

Nachdem Charakteristiktypen und Methodensignaturverfeinerungen erstellt sind, können nun die Datenverarbeitungscontainer erzeugt werden. Für die Transformation eines UMLsec-Eingabemodells für die Secure-Dependencies-Analyse kann zwischen drei Arten von Datenverarbeitungscontainern unterschieden werden:

- Container, die den Datenfluss eines Systemaufrufs eines Nutzungsszenarios beschreiben,
- Container, die den Datenfluss eines externen Aufrufs beschreiben, welche im Kontext dieser Transformation nur in den Verhaltensbeschreibungen der Methode *check()* auftreten, und
- Container, die eine interne Aktion beschreiben, welche im Kontext dieser Transformation nur in den Verhaltensbeschreibungen der Methoden auftreten, die aus den in UMLsec spezifizierten Methoden generiert werden.

Zunächst ist die Erzeugung der Datenverarbeitungscontainer der Systemaufrufe betrachtet. Diese sind nur Startpunkte für den Datenfluss und verfügen über keine für die Analyse wichtigen Eigenschaften. Daher verfügen sie lediglich über einen Datenverarbeitungsvorgang vom Typ *PerformDataTransmission*, der die Rückgabedaten der aufgerufenen Methode *check()* erhält. Das erhaltene Datum wird anschließend verworfen.

Container, die einen externen Aufruf beschreiben, verfügen dagegen über mehrere Datenverarbeitungsvorgänge und sind relevant für die durchzuführende Analyse, da sie zum einen die Sicherheitseigenschaften der übermittelten Parameter bestimmen und zum anderen die Sicherheitsanforderungen an die Rückgabevariable des Aufrufs festlegen. Für jeden externen Aufruf einer SEFF, die die Methode *check()* beschreibt, wird folgende Konstruktion durchgeführt: Für jeden Parameter der aufgerufenen Methode wird ein Datum erzeugt, indem ein Datenverarbeitungsvorgang des Typs *CreateData* erstellt wird. Das Datum, das von einem solchen Datenverarbeitungsvorgang erzeugt wird, werden direkt die entsprechenden Charakteristiken zugewiesen. Die Charakteristiken hängen von den Werten der Eigenschaftsdefinitionen des Stereotyps *Critical* ab, der an der Klasse annotiert ist, die zu der aktuell betrachteten Komponente korrespondiert. Dabei ist die aktuell betrachtete Komponente definiert als die Komponente, die die SEFF enthält, die über den externen Aufruf verfügt, dessen Datenverarbeitungscontainer aktuell erzeugt wird. Spezifiziert z. B. eine Klasse, dass für einen Parameter d_{UMLsec} einer Methode $m(d_{UMLsec} : Data) : Data$ Vertraulichkeit garantiert ist, wird in dem zu diesem Aufruf und zu dieser Klasse korrespondierenden Datenverarbeitungscontainer das Datum d_{DCP} mit der Charakteristik *Secrecy* erzeugt. Nachdem alle Datenverarbeitungsvorgänge des Typs *CreateData* erzeugt sind und Daten mit den entsprechenden Charakteristiken erzeugen, wird ein Datenverarbeitungsvorgang des Typs *PerformDataTransmission* erzeugt. Dieser transferiert jedes erzeugte Datum an die entsprechenden Parameter der Methodensignaturverfeinerungen, die die

Methode verfeinert, die das Ziel des externen Aufrufs ist, dessen Datenverarbeitungscontainer aktuell erstellt wird. Wenn die Methode über eine Rückgabevariable verfügt, wird definiert, dass dieser Datenverarbeitungsvorgang die Rückgabevariable des Methodenaufrufs zurückgibt. Wenn vorhanden, wird diese Rückgabevariable anschließend verworfen, da sie für die Analyse nicht weiter gebraucht wird und der Datenfluss terminiert sein muss, um ein gültiges Modell darzustellen. Die Sicherheitsanforderungen, die eine Klasse an den Rückgabewert eines Methodenaufrufs $m() : Data$ stellt, werden über die Annotation von Charakteristiken an dem Datenverarbeitungsvorgang des Typs *PerformDataTransmission* modelliert. Fordert eine Klasse z. B. die Integrität des Rückgabewertes der Methode $m()$, wird ein Charakteristikcontainer in dem Datenverarbeitungsvorgang des Typs *PerformDataTransmission* erzeugt, der den Wert *Integrity* für die Charakteristik *Critical* besitzt. Diese Charakteristik wird als Teil der Transformation in das Operationsmodell auf die entsprechende Operation abgebildet, wodurch die beschriebene Analyse möglich wird.

Schließlich sind noch die Datenverarbeitungscontainer zu betrachten, die interne Aktionen beschreiben. Diese bestimmen die Sicherheitsanforderungen an die Parameter der Methode, deren Datenfluss sie repräsentieren. Des Weiteren legen sie die Sicherheitseigenschaften der Rückgabevariable fest. Für jede interne Aktion wird folgende Konstruktion durchgeführt: Es ist in der Beschreibung der Analyse bereits die Einschränkung erklärt, dass lediglich Rückgabevariablen für die Analyse betrachtet werden. Daher wird für jeden eingehenden Parameter ein Datenverarbeitungsvorgang vom Typ *SelectData* erzeugt, der den übergebenen Parameter kopiert und zurückgibt. Diese Kopie wird anschließend verworfen. Dadurch ist es möglich, die Charakteristiken, die die Sicherheitsforderungen an diesen Parameter beschreiben, an diesen Datenverarbeitungsvorgang zu annotieren. Wenn z. B. für einen Parameter d die Sicherheitseigenschaft Vertraulichkeit gefordert ist, wird in dem Datenverarbeitungsvorgang des Typs *SelectData*, der die Kopie von d erstellt, ein Charakteristikcontainer erzeugt, der den Wert *Secrecy* für die Charakteristik *Critical* besitzt. Wenn die Methode über eine Rückgabevariable verfügt, wird anschließend ein Datenverarbeitungsvorgang vom Typ *CreateData* erstellt, der ein neues Datum erzeugt und diesem Charakteristiken zuweist. Die Charakteristiken können, wie bereits im Kontext der externen Aufrufe erklärt, aus den Eigenschaftswerten des UML-Stereotyps *Critical* der korrespondierenden Klasse ermittelt werden. Dieses erzeugte Datum wird anschließend unter Verwendung eines Datenverarbeitungsvorgangs des Typs *ReturnData* zurückgegeben.

In Abbildung 6.11 ist ein Beispiel für jeweils einen dieser Datenflüsse dargestellt. Es handelt sich dabei um einen Auszug aus dem vollständigen Datenflussmodell, das zu Abbildung 6.2 korrespondiert. Auf die Darstellung des vollständigen Datenflussmodells wird aus Platzgründen an dieser Stelle verzichtet.

Schließlich ist festzuhalten, dass getrennte Datenflüssen definiert werden, in denen Daten verworfen werden, nachdem sie für die Analyse keine Relevanz mehr haben. Dies ist für die Transformation gewählt, da dadurch möglichst wenig zusätzliche Informationen in das Modell eingeführt werden, welche im ursprünglichen UMLsec-Eingabemodell nicht vorhanden sind. Für die definierte Analyse können an beliebigen Datenverarbeitungsvorgängen Charakteristiken des Typs *Critical* annotiert werden. Die Analyse überprüft

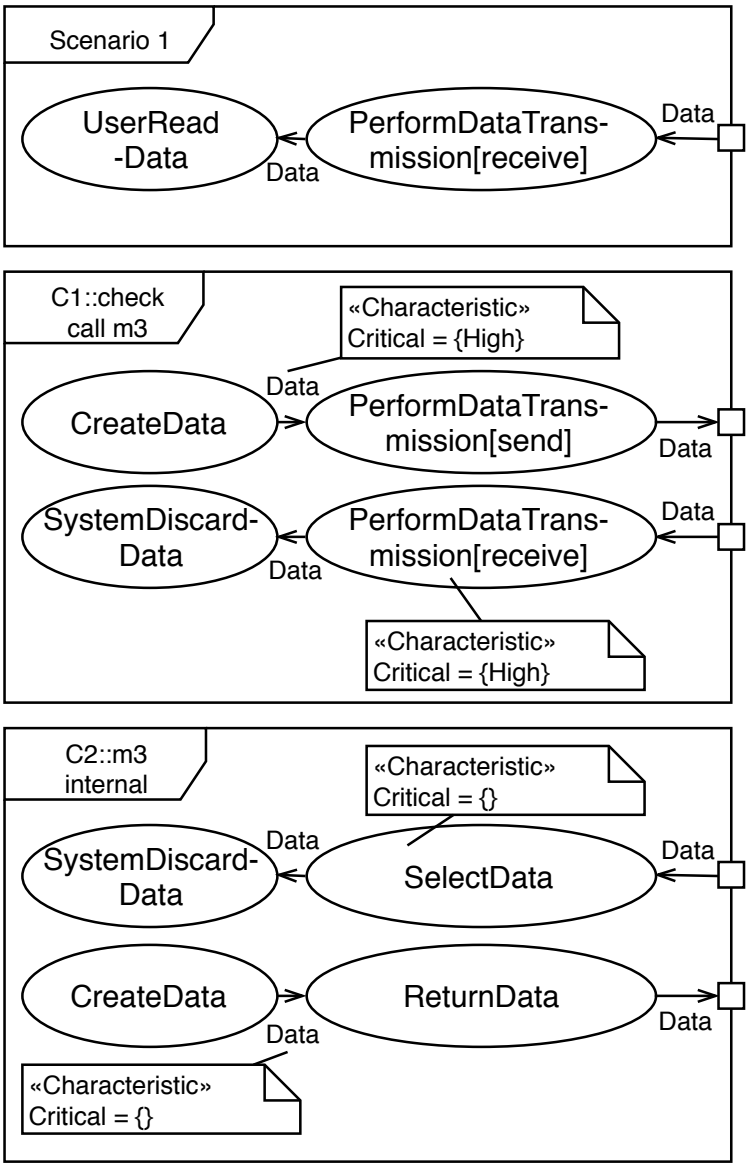

Abbildung 6.11.: Ein Auszug des definierten Datenflusses für das Beispiel aus Abbildung 6.2.

schließlich, ob die von dem Datenverarbeitungsvorgang zurückgegebenen Daten über dieselben Werte der Charakteristik *Critical* verfügen, wie es von dem Datenverarbeitungsvorgang vorgegeben ist. Es ist ersichtlich, dass damit die Secure-Dependencies-Analyse auch in komplexeren Datenflussmodellen angewandt werden kann und nicht nur in den minimalen Datenflüssen, die durch diese Transformation erzeugt werden.

6.3.2. Data-Centric Palladio zu UMLsec

Die Modelltransformation eines DCP-Modells zu einem UMLsec-Modell, das als Eingabemodell an die Secure-Dependencies-Analyse dienen kann, ist im Folgenden beschrieben. Es ist anzumerken, dass die folgende Modelltransformation auf Eingabemodelle beschränkt ist, die durch die bereits beschriebene Modelltransformation von UMLsec zu DCP entstanden sind. Dies ist deshalb notwendig, da im Allgemeinen die Abbildung einer DCP-Komponente auf eine Klasse in UMLsec semantisch nicht korrekt ist. Dies ist auf den Abstraktionsunterschied der Eingabemodelle der beiden Ansätze zurückzuführen. Weiter ist zu erwähnen, dass es sich nicht um eine Umkehrabbildung der beschriebenen Modelltransformation handelt. Dennoch existieren einige DCP-Modellelemente, die eine bijektive Relation zu UMLsec-Modellelementen im Kontext dieser beiden Modelltransformationen besitzen. Dazu gehören DCP-Komponenten, welche auf Klassen in UMLsec abgebildet werden, Schnittstellen, und benötigte bzw. bereitgestellte Rollen, welche auf Abhängigkeitsbeziehungen bzw. Realisierungsbeziehungen abgebildet werden. In einem ersten Schritt wird also für jede Schnittstelle in DCP eine Schnittstelle in UMLsec erzeugt, die über eine Methodendefinition gleicher Signatur pro Methode der DCP-Schnittstelle verfügt. Gegebenenfalls müssen in dem UMLsec-Modell zusätzliche Datentypen erzeugt werden, um die DCP-Methodensignatur in UMLsec abbilden zu können.

Anschließend wird für jede DCP-Komponente C_{DCP} eine Klasse in UMLsec erzeugt. Für jede Schnittstelle I_{DCP}, die die Komponente C_{DCP} bereitstellt, wird eine Realisierungsbeziehung in UMLsec von der Klasse C_{UMLsec} zu I_{UMLsec} erzeugt. Analog wird für jede Schnittstelle, von der die Komponente abhängt, eine Abhängigkeitsbeziehung zu der korrespondierenden Schnittstelle in UMLsec erstellt. Für jede RDSEFF der Komponente wird eine Methodendefinition in der entsprechenden Klasse in dem UMLsec-Modell erstellt. Die Methodensignatur wird dabei übernommen.

An dieser Stelle ist zu erkennen, wieso es sich nicht um eine Umkehrabbildung handelt. Da die Transformation von UMLsec nach DCP die neue Schnittstelle *Check* einführt, wird diese bei dieser Transformation in dem entstehenden UMLsec-Modell erstellt. Dies hat keinerlei Auswirkungen auf die Analyse, da keine Komponente von dieser Schnittstelle abhängt. Des Weiteren gehen die Informationen über Attribute von Klassen, sowie Methoden von Klassen, die nicht eine Methode einer Schnittstelle realisieren, während der Transformation von UMLsec zu DCP verloren. Diese können daher nicht wiederhergestellt werden.

Abschließend müssen die Eigenschaftswerte der Eigenschaftsdefinitionen des Stereotyps «`Critical`» aus dem DCP-Modell extrahiert werden. Dazu wird zunächst jede Klasse in

dem UMLsec-Modell mit diesem Stereotyp annotiert und erhält für die Sicherheitseigenschaften `secrecy`, `integrity` und `high` als Wert die leere Menge. Für jeden externen Aufruf einer RDSEFF einer Komponente C_{DCP} wird in dem entsprechenden Datenverarbeitungscontainer der Datenverarbeitungsvorgang *PerformDataTransmission* betrachtet. Dieser verfügt über einen Charakteristikcontainer, der die geforderten Sicherheitseigenschaften an den Rückgabewert der aufgerufenen Methode beschreibt. Der Name der Methode wird dem Wert der Eigenschaftsdefinition der geforderten Sicherheitseigenschaft hinzugefügt. Wenn z. B. die Werte *Secrecy* und *Integrity* für den Rückgabewert der Methode *m*() gesetzt sind, wird der String *m*() den Wertemengen der Eigenschaftsdefinition `secrecy` und `integrity` hinzugefügt. Analog kann für interne Aktionen vorgegangen werden. Für jede interne Aktion werden in dem dazugehörigen Datenverarbeitungscontainer die Datenverarbeitungsvorgänge vom Typ *SelectData* ermittelt. Wie bereits beschrieben, werden diese zur Überprüfung der Parameter verwendet. In der Klasse, die zu der Komponente korrespondiert, dessen RDSEFF aktuell betrachtet wird, werden die Sicherheitseigenschaften aus dem Charakteristikcontainer dieses Datenverarbeitungsvorgangs extrahiert und analog in Eigenschaftswerte für die entsprechenden Eigenschaftsdefinitionen des Stereotypen «`Critical`» transformiert.

Die Sicherheitseigenschaften der Daten müssen nun auch noch zu den Wertemengen des Stereotypen «`Critical`» hinzugefügt werden. Dazu müssen die Charakteristiken der Daten betrachtet werden, die von dem Datenverarbeitungsvorgängen des Typs *PerformDataTransmission* bzw. *SelectData* zurückgegeben werden. Analog zu der Secure-Links-Analyse können diese mittels einer Prolog-Abfrage an das aus dem DCP-Modell entstehende Prolog-Programm ermittelt werden. Gemäß der Konstruktion der Datenflussmodelle sind gerade die Datenverarbeitungsvorgänge für die Analyse relevant, die über die Charakteristik *Critical* verfügen. Daher kann die Abfrage alle Operationen mit der DCP-Eigenschaftsdefinition *Critical* generieren und ermitteln, welche Daten von einer solchen Operation zurückgegeben werden. Die Prolog-Abfrage ist in Algorithmus 6.3 dargestellt. Die für die Daten, die von einer Operation zurückgegeben werden, gesetzten Charakteristiken werden anschließend vereinigt und in die entsprechenden UML-Eigenschaftswerte der entsprechenden UML-Eigenschaftsdefinitionen überführt. Schließlich bleibt lediglich die Klasse zu ermitteln, zu deren Stereotyp diese Werte hinzuzufügen sind. Dazu wird für die betrachtete Operation der korrespondierende Datenverarbeitungsvorgang ermittelt. Dies ist eindeutig möglich. Ausgehend von diesem kann die zugehörige Komponente bestimmt werden, indem über den umschließenden Datenverarbeitungscontainer zu der Aktion, an der dieser Container annotiert ist, und zu der SEFF, die die Aktion enthält, traversiert wird. Die Werte werden der Klasse hinzugefügt, die zu der ermittelten Komponente korrespondiert.

```
1 ?- isOperation(OP),
2      hasProperty(OP, 'Critical'),
3      S = [OP|_],
4      stackValid(S),
5      operationReturnValue(OP, R),
6      returnValue(S, R, 'Critical', V).
```

Algorithmus 6.3.: Die Abfrage zur Ermittlung der Charakteristik der relevanten Daten zur Transformation in UML-Eigenschaftswerte.

7. Betrachtung der Ausdrucksmächtigkeit

In diesem Kapitel werden zunächst in Abschnitt 7.1 die für die Ausdrucksmächtigkeit relevanten Erkenntnisse der betrachteten Analysen zusammengefasst. Anschließend werden Analysen erklärt, die als Teil der Masterarbeit betrachtet wurden, aber sich nicht für eine vollständige Umsetzung in DCP eignen. Anschließend wird in Abschnitt 7.2 die Bedeutung dieser Erkenntnisse für die Ausdrucksmächtigkeit von DCP und UMLsec dargestellt.

7.1. Analysen

Im Folgenden sind in Unterabschnitt 7.1.1 und Unterabschnitt 7.1.2 zunächst die Erkenntnisse von Secure Links und Secure Dependencies zusammengefasst. Anschließend wird in Unterabschnitt 7.1.3 und Unterabschnitt 7.1.4 auf die Analysen Fair Exchange und sicherer Informationsfluss (no down-flow / no up-flow) eingegangen, welche sich im Kontext dieser Arbeit nicht für eine Umsetzung in DCP eignen, aber trotzdem im Rahmen der Vorauswahl untersucht wurden.

7.1.1. Secure Links

Die Umsetzung von Secure Links in DCP zeigt, dass die DCP-Analyse, im Gegensatz zu der UMLsec-Analyse, dazu in der Lage ist, die Sicherheitsanforderungen auf der Basis von Datenklassen zu betrachten und nicht nur pauschal auf der Basis von Abhängigkeiten zwischen Komponenten. Dies erlaubt die getrennte Betrachtung von den Daten, die über eine physische Netzwerkverbindung gesendet werden. Insbesondere können dadurch also die Aufrufparameter von den Rückgabevariablen unterschieden und getrennt untersucht werden. Dies ist z. B. in Fällen relevant, in denen eine dedizierte Komponente zur Verschlüsselung von Daten verwendet wird. Diese Daten können in diesem Beispiel als Aufrufparameter übergeben werden und sind als vertraulich zu behandeln. Die Rückgaben sind die verschlüsselten Daten, bei denen Vertraulichkeit nicht mehr gefordert ist. Ebenso ist der umgekehrte Fall denkbar, in dem die Rückgabevariable höhere Sicherheitsanforderungen als die Parameter erfordert, z. B. eine dedizierte Komponente zur Erzeugung von kryptographisch sicheren Zufallszahlen. In diesem Fall seien die Aufrufparameter die Parameter an die Zufallszahlengeneration, z. B. das Intervall, in dem die Zufallszahl liegen soll. Es ist sinnvoll, dies als vertraulich zu behandeln. Die Rückgabe der Komponente ist die generierte Zufallszahl. Da diese in diesem Beispiel zur Verwendung mit kryptographischen Protokollen genutzt wird, ist für die Rückgabevariable hohe Sensitivität zu

fordern. In diesen Fällen muss von UMLsec eine Überabschätzung getroffen werden und für das erste Beispiel für die Abhängigkeit Vertraulichkeit und für das zweite Beispiel hohe Sensitivität gefordert sein. Dadurch ist die Anzahl von falsch-positiven Ergebnissen höher, wodurch es für einen Entwickler schwieriger ist, die Ergebnisse der Sicherheitsanalyse zu interpretieren.

Ein weiterer Vorteil, der durch die Betrachtung der Sicherheitsanforderungen an den Daten selbst entsteht, ist, dass die Charakteristiken von Daten an beliebigen Stellen im Datenfluss geändert werden können. Dadurch ist es möglich, dass dieselbe Komponente den gleichen Aufruf an eine andere Komponente durchführt, aber als Parameter jeweils Daten mit unterschiedlichen Charakteristiken verwendet. Dies kann z. B. dadurch vorkommen, dass ein Teil der Daten von der Nutzereingabe abhängig ist und je nach eingegebenen Daten eine unterschiedliche Vertraulichkeit anzusetzen ist. Analog kann dies für die Rückgabevariablen sinnvoll sein, z. B. sei eine Komponente gegeben, die Profilinformationen eines Nutzers zurückgibt, dessen Bezeichner als Parameter an die Komponente übergeben wird. In diesem Fall sind die zurückgegebenen Daten abhängig von den Sichtbarkeitseinstellungen des Nutzerprofils. So existieren also Rückgabewerte, für die Vertraulichkeit gefordert und andere, für die dies nicht gefordert ist. Ersteres stellen private Nutzerprofile dar und zweiteres öffentliche Nutzerprofile.

Damit sind also Fälle identifiziert, die von der DCP-Analyse dargestellt werden können, aber nicht von der UMLsec-Analyse. Für jeden der Fälle ist ein Anwendungsbeispiel skizziert, das zeigt, dass es sich dabei nicht nur um eine rein theoretische Betrachtung handelt, sondern dieser Fall auch in der Praxis relevant ist. Es sind keine Fälle bekannt, die von der UMLsec-Analyse dargestellt werden können, aber nicht von der DCP-Analyse. Die Umsetzung der Secure-Links-Analyse in DCP hat gezeigt, dass diese Analyse von DCP durchgeführt werden kann.

7.1.2. Secure Dependencies

Die in DCP umgesetzte Secure-Dependencies-Analyse ist in der Lage, Verletzungen an der zweiten Bedingung der Secure-Dependencies-Analyse zu entdecken. Dabei handelt es sich um die Bedingung die besagt, dass die an einer Rückgabevariable oder einem Parameter erwarteten Sicherheitseigenschaften den tatsächlichen Sicherheitseigenschaften entsprechen. Die Spezifikation der Sicherheitseigenschaften und -anforderungen kann in DCP typsicher erfolgen, da diese jeweils an den entsprechenden Daten bzw. den Datenverarbeitungsvorgängen, die diese Daten verwenden, annotiert werden. Dies ist in UMLsec nicht der Fall. Hier wird ein Vergleich von Strings durchgeführt. Während dies nicht direkt die Ausdrucksmächtigkeit der Analyse beeinflusst, ist diese Typsicherheit in der Praxis ein signifikanter Vorteil der DCP-Analyse und deshalb an dieser Stelle erwähnt. Ein Vorteil der dadurch ist entsteht, ist insbesondere die Tatsache, dass in DCP eine eindeutige Semantik für den Fall vorliegt, in denen eine Komponente eine Schnittstelle bereitstellt, die sie selbst auch benötigt.

Auf der anderen Seite wird die erste Bedingung der Secure-Dependencies-Analyse in DCP überhaupt nicht überprüft. Bei der ersten Bedingung handelt es sich um die Einschränkung, dass wenn eine Sicherheitseigenschaft einer Rückgabevariable einer Methode einer Schnittstelle gefordert ist, die Abhängigkeit zu dieser Schnittstelle mit den Stereotypen annotiert sein muss, die zu den geforderten Sicherheitseigenschaften der Rückgabevariable korrespondieren. Es handelt sich hierbei um eine syntaktische Bedingung, die keine zusätzlichen Erkenntnisse über die Sicherheitseigenschaften der Rückgabevariablen zulässt, die nicht durch das Überprüfen der zweiten Bedingung gewonnen werden können. Für die Semantik der Analyse ist die Untersuchung der zweiten Bedingung also ausreichend. Unter dieser Betrachtung können keine Szenarien identifiziert werden, die von der UMLsec-Analyse modellierbar und analysierbar sind, von der DCP-Analyse allerdings nicht. Ferner ist durch die Umsetzung der Secure-Dependencies-Analyse in DCP gezeigt, dass diese Analyse von DCP durchgeführt werden kann.

Analog zu der Secure-Links-Analyse kann durch die Betrachtung der einzelnen Datenklassen eine detailliertere Analyse gegenüber der UMLsec-Analyse erreicht werden. Die Secure-Dependencies-Analyse in UMLsec erlaubt jedoch bereits die Spezifikation von Sicherheitsanforderungen an Rückgabevariablen und Parametern. Daher ist die Betrachtung der Datenklassen in DCP nur dann von Vorteil, wenn eine Klasse oder Komponente unterschiedliche Sicherheitsanforderungen an verschiedene Aufrufe derselben Methode stellt. In DCP kann dies modelliert werden, da jeder Methodenaufruf über einen eigenen Datenverarbeitungscontainer verfügen kann, in dem die Sicherheitsanforderungen spezifiziert werden. In UMLsec ist dieses Szenario nicht darstellbar. Stattdessen muss in UMLsec eine Überabschätzung für alle Methodenaufrufe der Klasse getroffen werden. Damit ist in DCP ein detaillierteres Analyseergebnis für die Secure-Dependencies-Analyse möglich.

7.1.3. Fair Exchange

Bei der Fair-Exchange-Analyse [1, Seite 53–55] handelt es sich um eine Analyse, die auf Aktivitätsdiagrammen durchgeführt wird. Sie hat als Untersuchungsgegenstand die Kontrollflüsse eines Systems und trifft im Gegensatz zu z. B. Secure Links oder Secure Dependencies keine Aussage über die Sicherheitseigenschaften von Daten. Die Analyse stellt sicher, dass in einer Transaktion keine der beiden beteiligten Parteien betrügen kann. Wenn ein Kunde eines Onlineshops z. B. eine Bestellung bezahlt hat, ist sicherzustellen, dass der Kunde am Ende der Transaktion die Bestellung entweder erhält oder im Falle einer Nichtzustellung sein Geld zurückbekommt. Für die Analyse werden drei Eigenschaftswerte des Stereotyps «`fair exchange`» definiert. Der Eigenschaftswert `{start}` beschreibt Zustände, in denen eine Transaktion für ein Gut beginnt und enthält als Wert Paare der Form (*Gut*, *Zustand*). Der Eigenschaftswert `{stop}` beschreibt den Zustand, in dem eine Transaktion für ein Gut beendet ist und enthält Tupel analog zu dem Eigenschaftswert `{start}`. Schließlich ist die Angreifermächtigkeit in dem Eigenschaftswert `{adversary}` definiert. Beschreibt das Modell nur die Transaktion für ein Gut, kann auf die Nennung des Guts in den Werten von den Eigenschaftswerten `{start}` und `{stop}` verzichtet werden. Die Analyse überprüft schließlich nach der Ausführung der zu dem Modell gehörenden

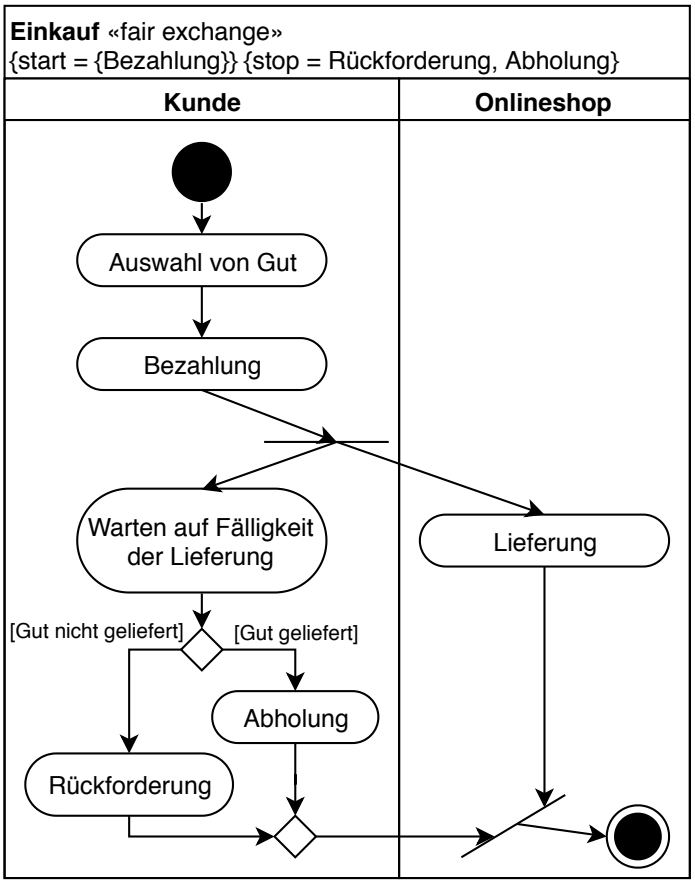

Abbildung 7.1.: Ein Beispiel eines Fair-Exchange-Modells.

UML-Maschine, ob in einer Ausführung für ein Gut gleich viele `{start}`- und `{stop}`-Zustände existieren. Ein Beispiel dafür ist in Abbildung 7.1 dargestellt.

Diese Analyse ist mit DCP nicht darstellbar, da der Angreifer in DCP nicht mit einer Ausführung des Modells interagieren kann. Stattdessen werden die statischen Eigenschaften des Datenflusses überprüft und daraus Rückschlüsse auf Operationen gezogen, in denen ein Angriff eines Angreifers einer bestimmten Mächtigkeit möglich ist. In UMLsec dagegen ist der Angreifer als eigene UML-Maschine modelliert, die zur Ausführungszeit nicht-deterministische Änderungen an den Warteschlangen der UML-Maschine des zu überprüfenden Systems vornehmen kann. Ein Angreifer in UMLsec kann also z. B. die Nachrichten einer Warteschlange des Systems löschen, wodurch verhindert werden kann, dass das System für einen gegebenen `{start}`-Zustand den korrespondierenden `{stop}`-Zustand erreicht. Eine Möglichkeit so etwas doch in DCP zu modellieren, wäre das explizite Modellieren des Fehlerfalls, in dem ein Datenfluss, der von einem `{start}`-Zustand ausgeht, frühzeitig terminiert wird, sodass eine Verletzung der Bedingung des Stereotyps «`fair exchange`» vorliegt. Da dies aber bedeutet, diesen Fehlerfall explizit als Teil des Verhaltens der Komponente zu definieren und dies semantisch falsch ist, eignet sich diese Problemumgehung nicht, um eine allgemein verwendbare Analyse zu definieren, die auf

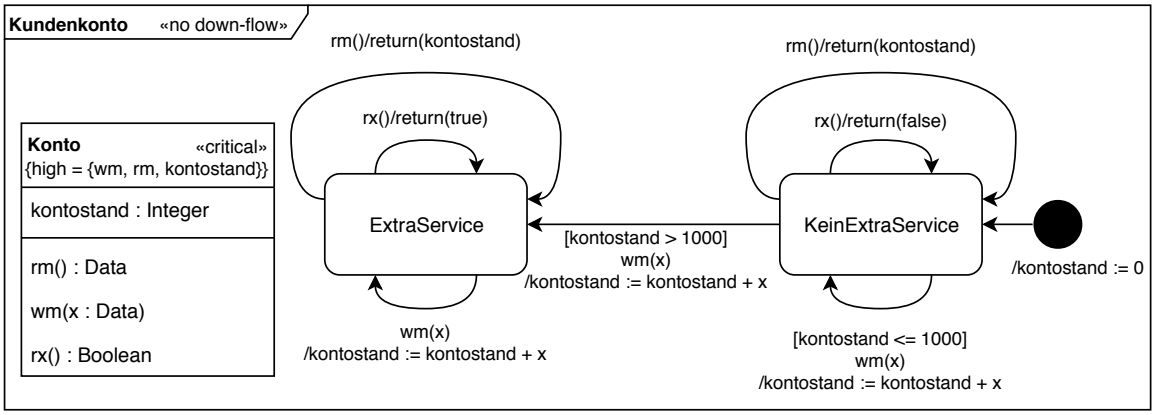

Abbildung 7.2.: Ein Beispiel eines Modells zur Verwendung mit der Analyse des sicheren Informationsflusses.

semantisch korrekten Eingabemodellen operiert. Aus diesen Gründen ist keine sinnvolle Umsetzung dieser Analyse in DCP möglich.

7.1.4. Sicherer Informationsfluss

In dieser Analyse [1, Seite 64f] wird untersucht, ob ein indirekter Informationsfluss sensitiver Informationen zu nicht vertrauenswürdigen Komponenten stattfindet (no down-flow) oder ob nicht vertrauenswürdige Komponenten sensitive Informationen beeinflussen (no up-flow). Diese Analyse wird auf Zustandsdiagrammen ausgeführt. Dazu wird für bestimmte Attribute und Methoden einer Klasse die Sicherheitseigenschaft `{high}` gefordert. In der Analyse wird dann überprüft, ob es möglich ist, Informationen über diese Attribute oder Rückgabewerte dieser Methoden durch Verwendung von Methoden ohne die Sicherheitseigenschaft `{high}` zu erhalten (no down-flow). Zusätzlich wird geprüft, ob Attribute oder Eingabeparameter von Methoden, die mit der Sicherheitseigenschaft `{high}` annotiert sind, durch Methoden ohne diese Annotation beeinflusst werden (no up-flow). Dabei handelt es sich bei der Sicherheitseigenschaft `{high}` um einen Eigenschaftswert des Stereotyps «`critical`», der für diese Analyse an der betrachteten Klasse angebracht ist. Ein Beispiel für ein solches Modell ist in Abbildung 7.2 dargestellt. Es handelt sich dabei um ein Kundenkonto einer Bank, die für Kunden, deren Kontostand mehr als 1000 Euro beträgt, zusätzliche Dienstleistungen zur Verfügung stellt. Das Konto ist also entweder in dem Zustand *ExtraService* oder *KeinExtraService*. Die Methode $wm(x)$ zahlt Geld auf das Konto ein und erhöht damit das Attribut *kontostand*, die Methode $rm()$ liest den aktuellen Kontostand und die Methode $rx()$ überprüft, ob es sich bei dem Konto um eines mit zusätzlichen Dienstleistungen handelt. Hier liegt eine Verletzung des Stereotyps «`no down-flow`» vor. Das Attribut *kontostand* ist als Attribut hoher Sensitivität deklariert. Die Methode $rx()$ erlaubt es jedoch, Informationen über den Wert dieses Attributs zu erlangen, da sie genau dann wahr zurückgibt, wenn gilt, dass $kontostand > 1000$.

Diese Analyse ist so in DCP nicht umsetzbar, da die Abstraktionsebene auf der diese Analyse operiert in DCP nicht darstellbar ist. Es gibt in DCP nicht die Möglichkeit, den Objektzustand einer Klasse zu modellieren, sodass eine äquivalente Analyse auf derselben Abstraktionsebene möglich wäre. Es ist jedoch eine ähnliche Analyse auf Architekturebene definierbar. Dabei wird jeder Rechnerknoten mit der Charakteristik Vertrauenswürdigkeit annotiert, welche die Werte vertrauenswürdig (high) oder nicht vertrauenswürdig (low) annehmen kann. Ein Datum, welches auf einem vertrauenswürdigen Rechnerknoten erzeugt wird, erhält die Charakteristik hohe Sensitivität (high), während ein Datum, welches auf einem nicht vertrauenswürdigen Rechnerknoten erzeugt wird, die Charakteristik niedrige Sensitivität (low) erhält. In der Analyse kann nun geprüft werden, ob ein Datenfluss eines Datums hoher Sensitivität zu nicht vertrauenswürdigen Rechnerknoten und eines Datums niedriger Sensitivität zu vertrauenswürdigen Rechnerknoten existiert. Diese Skizze einer Analyse ist jedoch aufgrund des Abstraktionsunterschiedes nicht äquivalent zu der in UMLsec beschriebenen Analyse des sicheren Informationsflusses. Es ist also auf die Umsetzung dieser skizzierten Analyse verzichtet da daraus keine zusätzlichen Informationen bezüglich der Ausdrucksmächtigkeit zwischen DCP und UMLsec gewonnen werden können. Weiter ist für die beschriebene UMLsec-Analyse keine äquivalente Umsetzung möglich, da DCP auf einer Abstraktionsebene operiert, auf der keine Modellierung des notwendigen Verhaltens möglich ist.

7.2. Ergebnisse

Es ist zu erkennen, dass Szenarien existieren, in denen die DCP-Analyse in der Lage ist, differenziertere Ergebnisse zu liefern als die korrespondierende UMLsec-Analyse. Dies zeigt sich an der Secure-Links-Analyse, in der die UMLsec-Analyse generell eine Überabschätzung der tatsächlichen Sicherheitseigenschaften der Daten treffen muss, und an der Secure-Dependencies-Analyse, in der mehrere Aufrufe an dieselbe Methode einer Schnittstelle mit unterschiedlichen Sicherheitseigenschaften erfolgen können. Beides lässt sich auf den Fakt zurückführen, dass in DCP Datenflüsse als Kerngegenstand der Analyse betrachtet werden, wodurch bereits auf einer hohen Abstraktionsebene viele Informationen für die Analysen gewonnen werden können und dadurch das Analyseergebnis wesentlich detaillierter ist als die korrespondierenden Ergebnisse der UMLsec-Analysen.

Auf der anderen Seite existieren jedoch auch Analysen in UMLsec, die von DCP überhaupt nicht dargestellt werden können. Szenarien, wie die Fair-Exchange-Analyse, die darauf basieren, dass ein Angreifer die Ausführung soweit beeinflusst, dass eine Komponente von ihrem definierten Verhalten abweicht, sind in DCP nicht modellierbar oder analysierbar. Ferner existiert mit der Analyse des sicheren Informationsflusses eine UMLsec-Analyse, die auf einer niedrigen Abstraktionsebene operiert, welche von DCP nicht dargestellt werden kann. In diesem Fall muss jedoch zwischen der Intention der Analyse und der konkreten Analysedefinition unterschieden werden. Im Kern ist das Ziel der Analyse sicherzustellen, dass Daten unterschiedlicher Vertraulichkeitsstufe sich nicht gegenseitig beeinflussen. Dies kann in DCP auf Architekturebene modelliert werden. Dabei ist es

jedoch nicht möglich eine zu der konkreten UMLsec-Analysedefinition äquivalente DCP-Analyse zu spezifizieren, da der Detailgrad der UMLsec-Analyse in DCP nicht darstellbar ist.

Zusammenfassend ist festzuhalten, dass sich insbesondere strukturelle Analysen aus UMLsec gut in DCP umsetzen lassen. Beispiele dafür sind Secure Links und Secure Dependencies. Solche Analysen operieren auf einer ausreichend hohen Abstraktionsebene, sodass sich die Modellelemente der Eingabemodelle meist direkt auf die Modellelemente des PCM abbilden lassen. Zusätzlich sind Datenflüsse für diese Modelle in der Regel einfach zu definieren, da die Struktur bereits einen Teil des notwendigen Datenflusses vorgibt. Kontrollflussorientierte Analysen, die auf Zustands- oder Aktivitätsdiagrammen operieren, sind dagegen problematischer in der Umsetzung in DCP. Für die Ausdrucksmächtigkeit bedeutet dies also, dass weder UMLsec strikt mächtiger ist als DCP, noch dass DCP strikt mächtiger ist als UMLsec. Stattdessen ist der erwartete Anwendungsfall von erheblicher Bedeutung für die Wahl des richtigen Werkzeugs. Bei der Entwicklung eines Informationssystem, in der die Erwartung ist, dass die Modelle von Entwicklern erstellt und gepflegt werden, ist nach dieser Einschätzung DCP vorzuziehen, da es detailliertere Aussagen über die Sicherheitsverletzungen erlaubt und auf der Architekturebene operiert, wodurch die meisten Entwickler voraussichtlich vertraut mit den in DCP verwendeten Modellierungskonzepten sind. UMLsec dagegen erlaubt das Modellieren von wesentlich spezifischeren Anwendungsfällen, wie z. B. einer Variante des TLS-Protokolls [1, Seite 80–88], erfordert aber aufgrund der in der Regel niedrigeren Abstraktionsebene auch wesentlich mehr Wissen über die modellierten Sachverhalte und eignet sich daher eher in Situationen, in denen bereits ein sehr detaillierter Entwurf vorliegt.

8. Evaluation

Dieses Kapitel beschreibt die Evaluation der erstellten Modelltransformationen und Sicherheitsanalysen. Da die Evaluationsziele für jede betrachtete Sicherheitsanalyse identisch sind, werden diese Ziele zunächst in Abschnitt 8.1 unabhängig von den Sicherheitsanalysen dargestellt. Anschließend wird die Evaluation der einzelnen Analysen durchgeführt. Dazu werden je Analyse Fragen und Metriken zu den Evaluationszielen definiert, der Versuchsaufbau erläutert und schließlich die Ergebnisse präsentiert und diskutiert. Darauf folgend wird die Validität der Evaluation in Abschnitt 8.4 bewertet. Abschließend werden in Abschnitt 8.5 die notwendigen Annahmen beschrieben, auf denen die erstellten Analysen basieren.

Die Evaluation ist nach dem Vorgehen des Goal-Question-Metric-Ansatzes (GQM) [43] strukturiert. In GQM wird zwischen drei zentralen Elementen unterschieden, die zusammen mehrere hierarchische Strukturen bilden. Die Ziele (Goals) stellen die Wurzelknoten der Hierarchien dar und spezifizieren jeweils ein zu erreichendes Ziel der Untersuchung. Zu jedem Ziel werden anschließend Fragen (Questions) formuliert, durch deren Beantwortung eine Aussage darüber möglich ist, ob das Ziel erreicht wurde. Zu der Beantwortung einer Frage werden Metriken (Metrics) definiert. Dies erlaubt die Erstellung einer objektiven und nachvollziehbaren Antwort auf die zugehörige Frage.

8.1. Evaluationsziele

Das Ziel dieser Masterarbeit ist es, die praktische Ausdrucksmächtigkeit des in UMLsec definierten Ansatzes zur Sicherheitsanalyse von Systemen mit dem in DCP definierten Ansatz zu vergleichen. Die Aussagen über die Ausdrucksmächtigkeit sind einfache und direkte logische Schlüsse, die aus den Erkenntnissen gezogen werden, welche durch die Erstellung der Modelltransformationen und der Analyse-Abfragen gewonnen wurden. Es ist daher wichtig, die Basis, aus der diese Erkenntnisse gewonnen wurden, zu validieren. Daher ist das Ziel der Evaluation die Validierung der Genauigkeit der verschiedenen in DCP erstellten Sicherheitsanalysen. Dazu wird für jede Sicherheitsanalyse ihre Äquivalenz zu der jeweils korrespondierenden Sicherheitsanalyse aus UMLsec überprüft. Im Folgenden werden gemeinsame Evaluationsziele definiert, die von jeder betrachteten Sicherheitsanalyse erfüllt werden sollen.

Um diese Ziele zu definieren, müssen zunächst die einzelnen Bestandteile einer Sicherheitsanalyse betrachtet werden. Jede betrachtete Sicherheitsanalyse besteht aus einem Eingabemodell und einer Analysedefinition. In UMLsec ist das Eingabemodell in UML

definiert und um UMLsec-spezifische Stereotypen erweitert. Dagegen besteht das Eingabemodell in DCP aus einem PCM-Modell und einem Datenflussmodell. Die Analysedefinition der UMLsec-Analysen ist von Jürjens [1, Kapitel 3 und 4] textuell angegeben. In DCP wird als Analysedefinition ein Prolog-Prädikat verwendet, das Fakten abfragt, welche die Struktur des Operationsmodells beschreiben. Für jede betrachtete Analyse sind jeweils zwei Transformationen zwischen diesen beiden Eingabemodellen definiert, sowie die entsprechende Analyseformel in Prolog konzipiert. Dies führt zu den folgenden beiden Evaluationszielen:

Z1 Die in DCP erstellte Prolog-Abfrage entspricht der Semantik der Definition der betrachteten UMLsec-Analyse.

Z2 Die erstellten Modelltransformationen erzeugen bezüglich der Analyse für beliebige UMLsec-Eingabemodelle äquivalente DCP-Eingabemodelle und für beliebige DCP-Eingabemodelle äquivalente UMLsec-Eingabemodelle.

In Z1 wird die definierte Prolog-basierte Analyseformel mit der Semantik der Definition der betrachteten Analyse verglichen. Die UMLsec-Analysen haben einen Rückgabewert von Erfolg, wenn kein Sicherheitsproblem in dem Eingabemodell gefunden werden konnte und einen Rückgabewert von Fehlschlag, wenn mindestens ein Sicherheitsproblem aufgedeckt wurde. Des Weiteren ist im Falle eines Fehlschlags die Ausgabe um die eindeutigen Bezeichner der Modellelemente, die für das Sicherheitsproblem verantwortlich sind, ergänzt. Existiert mehr als ein Sicherheitsproblem in dem betrachteten Modell, wird diese Information für jedes gefundene Sicherheitsproblem ausgegeben.

In Z2 werden die erstellten Modelltransformationen überprüft. Der direkte Vergleich der Äquivalenz von zwei Modellen in verschiedenen Modellierungssprachen und auf unterschiedlichen Abstraktionsebenen, wie es in dieser Arbeit der Fall ist, ist nicht sehr aussagekräftig. Es existiert keine links- und rechtsvollständige Relation zwischen der Gesamtheit der Modellelemente der Eingabemodelle aus UMLsec und DCP. Im Folgenden wird daher die Äquivalenz der Eingabemodelle und damit die erstellte Modelltransformation über die Äquivalenz der Ergebnisse der Sicherheitsanalyse überprüft. Je nach betrachteter Analyse existiert dafür eine UMLsec-Implementation in der Form des CARiSMA-Werkzeugs [41]. Es ist anzumerken, dass kein Anspruch besteht, eine allgemeingültige Transformation beliebiger UML-Modelle in DCP-Modelle zu definieren. Diese Betrachtung beschränkt sich daher auf die Eingabemodelle an die jeweilige Analyse.

8.2. Secure Links

Für die Secure-Links-Sicherheitsanalyse sind im Folgenden die Fragen und Metriken zu den bereits definierten Zielen Z1 und Z2 vorgestellt. Anschließend wird der Versuchsaufbau und insbesondere die Erzeugung der Eingabemodelle erläutert. Abschließend werden die Ergebnisse präsentiert und diskutiert.

8.2.1. Fragen und Metriken

Zunächst sind die Fragen zu Evaluationsziel Z1 formuliert. Hierbei soll überprüft werden, ob die eingeführten Prolog-Prädikate, die zur Analyse der Secure-Links-Bedingungen dienen, der Semantik der UMLsec-Analysedefinition entsprechen. Um eine bessere Aussage über diese Definition treffen zu können, wird sie durch eine einfache Umformung in die Form einer logischen Formel gebracht. Ausgehend von dieser Formel ist eine Argumentation bezüglich der Prolog-Prädikate möglich. Ein Faktor, der dabei zusätzlich betrachtet werden muss, ist die Tatsache, dass in einem Fehlerfall die UMLsec-Analyse dem Nutzer die für das Sicherheitsproblem verantwortlichen Modellelemente mitteilt. Bei der Betrachtung der logischen Formel geht diese Information offensichtlich verloren. Daher muss zusätzlich überprüft werden, ob die Prolog-Prädikate ebenfalls in der Lage dazu sind, die verantwortlichen Modellelemente auszugeben. Daraus ergeben sich folgende Fragestellungen für die Evaluierung. Diese Fragen werden argumentativ beantwortet, daher sind für sie keine Metriken definiert.

F1.1 Erlaubt die Prolog-Formel den Rückschluss auf die Modellelemente, die im Fall eines Sicherheitsproblems für dieses Problem verantwortlich sind?

F1.2 Entspricht die Semantik der aufgestellten Prolog-Formel der Semantik, der in UMLsec definierten Bedingungen, die von einem Modell zu erfüllen sind?

Das Evaluationsziel Z2 dient dazu, zu überprüfen, ob die Modelltransformation zu einem UMLsec-Eingabemodell ein äquivalentes DCP-Eingabemodell liefert. Dabei kommt die in Z1 evaluierte Analyseformel zum Einsatz. Es soll überprüft werden, ob für ein UMLsec-Eingabemodell und ein DCP-Eingabemodell, welche gemäß der definierten Modelltransformation äquivalent zueinander sind, die beiden Analysen ein äquivalentes Ergebnis erzeugen. Hierbei müssen mehrere Modelltransformationsrichtungen berücksichtigt werden. Diese sind in den folgenden Fragen beschrieben.

F2.1 Sei ein UMLsec-Eingabemodell gegeben. Erzeugt die Analyse in DCP mit dem nach DCP transformierten Eingabemodell ein äquivalentes Ergebnis zu der UMLsec-Analyse?

F2.2 Sei ein DCP-Eingabemodell gegeben. Erzeugt die Analyse in UMLsec mit dem nach UMLsec transformierten Eingabemodell ein äquivalentes Ergebnis zu der DCP-Analyse?

F2.3 Sei ein UMLsec-Eingabemodell gegeben. Erzeugt die Analyse in UMLsec, nachdem das Eingabemodell nach DCP transformiert und von DCP zurück nach UMLsec transformiert wurde, das gleiche Ergebnis wie vor der Hin- und Rücktransformation?

F2.4 Sei ein DCP-Eingabemodell gegeben. Erzeugt die Analyse in DCP, nachdem das Eingabemodell nach UMLsec transformiert und von UMLsec zurück nach DCP transformiert wurde, das gleiche Ergebnis wie vor der Hin- und Rücktransformation?

In den Fragen F2.1 und F2.2 werden jeweils einfache Modelltransformationen durchgeführt und die Ergebnisse der DCP-Analyse mit den Ergebnissen der UMLsec-Analyse verglichen. Hierzu ist es, neben den ermittelten Ergebnissen, notwendig zu beschreiben, wann es sich bei den Ergebnissen der beiden Analysen um äquivalente Ergebnisse handelt. Die Fragen F2.3 und F2.4 führen eine Hin- und Rücktransformation (roundtrip transformation) durch und vergleichen daher das Ergebnis der DCP-Analyse (bzw. UMLsec-Analyse) vor der Transformation mit dem Ergebnis der DCP-Analyse (bzw. UMLsec-Analyse) nach der Transformation. Hierbei ist also keine Argumentation bezüglich der Gleichheit der Ergebnisse notwendig. Ferner erlaubt diese Hin- und Rücktransformation die Überprüfung, dass die Analyseergebnisse nicht von den Modelltransformationen beeinflusst werden. Diese Vorgehen werden für mehrere Modelle wiederholt. Für alle vier Fragen wird die Fehlerrate als Metrik verwendet. Zusätzlich wird in Fällen, in denen die Analyseergebnisse nicht mit den erwarteten Analyseergebnissen übereinstimmen, anhand der UMLsec-Implementierung der Analyse die Ursache für diese Diskrepanz ermittelt.

8.2.2. Versuchsaufbau

Da die Fragestellungen F1.1 und F1.2 argumentativ beantwortet werden, ist für die Beantwortung dieser Fragen keine Erklärung eines Versuchsaufbaus notwendig. Anders sieht das bei den Fragen F2.1 bis F2.4 aus. Für diese wird im Folgenden der Versuchsaufbau erklärt.

8.2.2.1. Frage F2.1

Mit der Frage F2.1 wird die Modelltransformation von UMLsec nach DCP untersucht. Im Folgenden wird zunächst beschrieben, wie die Analyseergebnisse der UMLsec-Analyse erzeugt werden und wie daraus die für die DCP-Analyse erwarteten Analyseergebnisse ermittelt werden. Anschließend wird erklärt, wie die DCP-Analyseergebnisse ermittelt werden und wie der Vergleich dieser Ergebnisse mit den erwarteten Ergebnissen durchgeführt wird.

Es sei ein UMLsec-Eingabemodell gegeben. Dann wird zunächst das Analyseergebnis der Secure-Links-Analyse in UMLsec bestimmt. Hierfür wird das CARiSMA-Werkzeug [41] verwendet. Es handelt sich dabei um die aktuellste Implementation der Secure-Links-Analyse aus UMLsec. Die Implementation war, bis auf die in Kapitel 5 genannten und vernachlässigbaren Ausnahmen, konsistent mit der Definition der Secure-Links-Analyse, die in dem Buch von Jürjens [1, Seite 59] gegeben ist. Bei der Analyse kann es zu $|R^{UMLsec}|$ Sicherheitsproblemen $r^{UMLsec} \in R^{UMLsec}$ kommen, die von der Analyse entdeckt werden. Zu jedem Sicherheitsproblem r^{UMLsec} wird der Kommunikationspfad c notiert, der für das Sicherheitsproblem verantwortlich ist. Dies entspricht der Basis für das erwartete Ergebnis der DCP-Analyse für dieses Eingabemodell. Es ist jedoch zu erwähnen, dass in der DCP-Analyse sowohl die Datenflussrichtung über die Eingabeparameter, als auch über die Rückgabevariablen analysiert wird und bei einem Sicherheitsproblem jede Richtung

jeweils als separater Fehler dem Nutzer ausgegeben wird. UMLsec überprüft dagegen nur beide Richtungen als eine Abhängigkeit zwischen Artefakten und meldet dementsprechend beim Auftreten eines Sicherheitsproblems nur einen Fehler an den Nutzer. Daher ist zu erwarten, dass zu jedem Sicherheitsproblem r^{UMLsec} der UMLsec-Analyse zwei Sicherheitsprobleme in der DCP-Analyse gemeldet werden. Bei n Sicherheitsproblemen in der UMLsec-Analyse sind also genau $2n$ Probleme in der DCP-Analyse zu erwarten. Seien mit $r^{DCP} \in R^{DCP}$ die Sicherheitsprobleme bezeichnet, die die DCP-Analyse entdeckt. Für jedes Sicherheitsproblem r^{UMLsec} der UMLsec-Analyse wird also erwartet, dass die DCP-Analyse zwei Probleme r_i^{DCP} und r_j^{DCP} so ausgibt, dass gilt

- l ist die für r_i^{DCP} und r_j^{DCP} verantwortliche Netzwerkressource,
- l korrespondiert zu dem Kommunikationspfad c und
- die verletzte Sicherheitsanforderung ist sowohl bei r^{UMLsec} als auch bei r_i^{DCP} und r_j^{DCP} identisch.

Anders ausgedrückt wird gefordert, dass nicht nur die Anzahl an Sicherheitsproblemen der Erwartung entspricht, sondern auch zu jedem Sicherheitsproblem die richtige verantwortliche Netzwerkressource und Sicherheitsanforderung zugeordnet ist.

Nachdem damit das erwartende Ergebnis bestimmt ist, wird die Modelltransformation gemäß der Spezifikation in Kapitel 5 von dem UMLsec-Eingabemodell zu dem DCP-Eingabemodell durchgeführt. Für das DCP-Modell wird nun die DCP-Variante der Secure-Links-Analyse ausgeführt. Dazu wird das DCP-Eingabemodell mittels einer Modell-zu-Modell-Transformation in das Operationsmodell überführt. Von dem Operationsmodell kann die Modell-zu-Text-Transformation ausgeführt werden, die das Operationsmodell in ein Prolog-Programm übersetzt. Wenn die Angreifermächtigkeit des UMLsec-Eingabemodells der Standardangreifer ist, wird das Prolog-Prädikat *secureLinks\6* mit $ATT = default$ abgefragt. Ist die Angreifermächtigkeit der Insiderangreifer, wird $ATT = insider$ gesetzt. Damit stehen die Ergebnisse der DCP-Analyse zur Verfügung. Dabei können $|R^{DCP}|$ Sicherheitsprobleme r^{DCP} auftreten. Diese werden mit den erwarteten Ergebnissen verglichen. Die erwarteten Ergebnisse sind die Analyse-Ergebnisse, die aus der UMLsec-Analyse ermittelt wurden und für die für jedes Sicherheitsproblem der UMLsec-Analyse zwei Sicherheitsprobleme der DCP-Analyse erwartet werden. Sollten die tatsächlich ermittelten Analyseergebnisse der DCP-Analyse von diesen erwarteten Problemen abweichen liegt ein Fehlerfall vor. Die DCP-Analyse liefert dann nicht äquivalente Ergebnisse zu der UMLsec-Analyse für dieses Eingabemodell. Für die Menge aller betrachteten Eingabemodelle ist die Metrik der Fehlerrate gegeben durch

$$f_{UMLsec \rightarrow DCP} = \frac{\text{Anzahl Fehlerfälle}}{\text{Anzahl Eingabemodelle}}.$$

Des Weiteren wird in Fällen, in denen eine Diskrepanz der erwarteten Ergebnissen zu den tatsächlichen Analyseergebnissen vorliegt, das CARiSMA-Werkzeug verwendet, um zu ermitteln wie diese Diskrepanz zu Stande kommt.

Da zur Bearbeitungszeit dieser Masterarbeit noch keine Werkzeugunterstützung für die komplette Transformationskette von DCP-Eingabemodell zu Prolog-Programms verfügbar

ist und damit also manueller Aufwand zur Erzeugung der Evaluationsergebnisse notwendig ist, erfolgen die Auswertung der Prolog-Prädikate und der Vergleich der Ergebnisse ebenfalls manuell.

8.2.2.2. Frage F2.2

Mit der Frage F2.1 wird die Modelltransformation von DCP nach UMLsec untersucht. Die Beantwortung von Frage F2.2 erfolgt ähnlich zu der Beantwortung von Frage F2.1. Im Folgenden wird zunächst beschrieben, wie die Analyseergebnisse der DCP-Analyse erzeugt werden und wie daraus die für die UMLsec-Analyse erwarteten Analyseergebnisse ermittelt werden. Anschließend wird erklärt, wie die UMLsec-Analyseergebnisse ermittelt werden und wie der Vergleich dieser Ergebnisse mit den erwarteten Ergebnissen durchgeführt wird.

Es ist ein DCP-Eingabemodell gegeben. Mit diesem als Eingabe wird die Transformation zu dem Prolog-Programm ausgeführt. Dies ist bereits in Unterunterabschnitt 8.2.2.1 beschrieben. Anschließend wird das Prädikat *secureLinks\6* abgefragt. Dieses Prädikat erzeugt sowohl die Analyseergebnisse für den Standardangreifer als auch für den Insiderangreifer. Da die Analyse hier wieder Sicherheitsprobleme in Aufruf- und Rückgaberichtung findet, dies von UMLsec aber nicht dargestellt werden kann, werden diese Probleme jeweils zusammengefasst. Dabei gilt, dass jeweils zwei Sicherheitsprobleme zusammengefasst werden, für die die Prolog-Parameterbelegungen des Prädikats *secureLinks\6* sich nur in dem Richtungsparameter D unterscheiden. Es muss also in der ersten Parameterbelegung D = CallParameter und in der zweiten Parameterbelegung D = ReturnValue gelten. Alle anderen Parameter müssen zueinander identisch sein. Diese Ergebnisse werden, jeweils dem entsprechenden Angreifer zugeordnet, notiert.

Anschließend wird die Transformation des DCP-Eingabemodells in das UMLsec-Eingabemodell durchgeführt. Da diese Transformation zwei UMLsec-Modelle erzeugt, eines mit dem Standardangreifer und eines mit dem Insiderangreifer, wird auf beiden dieser Modelle die UMLsec-Analyse durchgeführt. Analog zu Frage F2.1 wird dafür das CARiSMA-Werkzeug verwendet. Seien die von der DCP-Analyse ermittelten und bereits zusammengefassten Sicherheitsprobleme des Standardangreifers $r^{DCP}_{default} \in R^{DCP}_{default}$ und die des Insiderangreifers $r^{DCP}_{insider} \in R^{DCP}_{Insider}$. Es wird erwartet, dass die UMLsec-Analyse, unter der Eingabe des Modells, in dem der Standardangreifer verwendet wird, für jedes Sicherheitsproblem $r^{DCP}_{default}$ ein Sicherheitsproblem $r^{UMLsec}_{default} \in R^{UMLsec}_{default}$ entdeckt, für das gilt, dass der für $r^{UMLsec}_{default}$ verantwortliche Kommunikationspfad zu der für $r^{DCP}_{default}$ verantwortlichen Netzwerkressource korrespondiert und die verletzte Sicherheitsanforderung bei $r^{UMLsec}_{default}$ und $r^{DCP}_{default}$ identisch ist. Weiter wird erwartet, dass $|R^{DCP}_{default}| = |R^{UMLsec}_{default}|$. Analog gilt dies für den Insiderangreifer und die Menge der Sicherheitsprobleme $r^{UMLsec}_{insider} \in R^{UMLsec}_{insider}$, die durch die UMLsec-Analyse des Modells, in dem der Insiderangreifer verwendet wird, entdeckt werden. Die Sicherheitsprobleme $R^{UMLsec}_{insider}$ müssen in Bezug zu $R^{DCP}_{Insider}$ dieselben Bedingungen erfüllen, die $R^{UMLsec}_{default}$ in Bezug zu $R^{DCP}_{default}$ erfüllen muss. Die Definition der

Fehlerrate $f_{DCP \rightarrow UMLsec}$ ist analog zu Frage F2.1 und die Auswertung der Ergebnisse erfolgt wieder manuell.

8.2.2.3. Frage F2.3

Für die Beantwortung von Frage F2.3 ist keine Definition von Äquivalenzklassen von Ergebnissen notwendig, so wie das bei Frage F2.1 und F2.2 benötigt wird. Da hier eine Hin- und Rücktransformation ausgeführt wird und die Ergebnisse vor und nach diesen Transformationen miteinander verglichen werden, wird hier ein exakter Vergleich der Ergebnisse durchgeführt. Dieses Vorgehen ist im Folgenden beschrieben.

Frage F2.3 beginnt diesen Teil der Evaluation ausgehend von dem UMLsec-Modell. Dabei werden zunächst die Ergebnisse r der Analyse mittels des CARiSMA-Werkzeugs ermittelt und notiert. Anschließend werden die Modelltransformation zu dem DCP-Eingabemodell und die Modelltransformation des entstehenden DCP-Modells zurück zu einem UMLsec-Modell durchgeführt. Bei der Transformation von UMLsec nach DCP geht die Information über den Angreifer verloren. Angenommen, das ursprüngliche UMLsec-Modell hätte als Angreifer den Standardangreifer definiert. In DCP ist der Angreifer nicht Teil des Eingabemodells und daher wird bei der Rücktransformation von DCP zu UMLsec jeweils ein Modell für Standardangreifer und Insiderangreifer erzeugt. Da sich diese Modelle sonst in keinem weiteren Modellelement unterscheiden, wird für diese Evaluation der definierte Angreifer des ursprünglichen UMLsec-Modells notiert. Durch die Information über den ursprünglichen Angreifer kann aus den beiden UMLsec-Modellen, die aus dem DCP-Modell transformiert sind, das zu dem ursprünglichen Angreifer korrespondierende Modell ausgewählt werden. Von diesem werden anschließend mittels des CARiSMA-Werkzeugs die Analyseergebnisse r' ermittelt und mit den vorherigen Ergebnissen r verglichen. Die Erwartung ist, dass nach der Hin- und Rücktransformation genau die selben Sicherheitsprobleme entdeckt werden, die vor diesen Transformationen entdeckt wurden. Die Fehlerrate für dieses Experiment ist mit $f_{UMLsec \rightarrow DCP \rightarrow UMLsec}$ bezeichnet.

8.2.2.4. Frage F2.4

Zur Beantwortung von Frage F2.4 wird ein ähnliches Vorgehen angewendet, wie bei Frage F2.3. Die einzigen Unterschiede bestehen darin, dass hier die Hin- und Rücktransformation ausgehend von einem DCP-Eingabemodell durchgeführt wird und dass es nicht notwendig ist, den Angreifertyp zu notieren. Nachdem die Ergebnisse r der Secure-Links-Analyse in DCP ermittelt und notiert wurden, werden aus dem DCP-Modell unter Verwendung der definierten Modelltransformation zwei UMLsec-Modelle erzeugt. Aus jedem dieser beiden Modelle wird anschließend jeweils ein DCP-Modell transformiert. Da sich die beiden UMLsec-Modelle nur im Angreifer unterscheiden ist zu erwarten, dass die entstehenden beiden DCP-Modelle identisch sind. Daher werden für jedes der beiden entstehenden DCP-Modelle die Ergebnisse r' und r'' der Secure-Links-Analyse ermittelt. Anschließend werden die Ergebnisse r, r' und r'' miteinander verglichen. Die Erwartung ist, dass $r = r' = r''$,

also in allen drei Modellen genau die selben Sicherheitsprobleme entdeckt werden. Die Fehlerrate ist mit $f_{DCP \to UMLsec \to DCP}$ bezeichnet.

Zusammenfassend ist also für sowohl Frage F2.3 und F2.4 das zu überprüfende Ereignis, ob die Ausgaben der Analyse vor der Hin- und Rücktransformation identisch zu den Ausgaben nach diesen Transformationen sind. Es müssen lediglich einige kleine Anpassungen an dem Vorgehen vorgenommen werden, da es sich bei den definierten Modelltransformationen nicht um eine Abbildung und ihre Umkehrabbildung handelt. Die Überprüfung der Ergebnisse erfolgt wie auch schon bei Frage F2.1 und F2.2 manuell nach dem hier definierten systematischen Vorgehen. Die Überprüfung der Ergebnisse ist daher objektiv durchführbar.

8.2.3. Modellerstellung

In diesem Abschnitt wird erklärt, wie die Modelle erstellt werden, die für die Versuche zur Beantwortung von Fragestellungen F2.1 bis F2.4 verwendet werden. Es müssen jeweils UMLsec-, als auch DCP-Modelle erzeugt werden. Im Folgenden wird zunächst die Erstellung der UMLsec-Modelle erklärt. Dabei wird zwischen verschiedenen Arten von Modellen unterschieden. Zunächst werden Basisfälle definiert. Basisfälle sind UMLsec-Modelle, die über die minimale Struktur verfügen, sodass die Secure-Links-Analyse möglich ist. Alle Basisfälle zusammen stellen alle mögliche Belegungen von UMLsec-Stereotypen an den entsprechenden Modellelementen dieser feste Modellstruktur dar. Schließlich werden strukturell interessante Fälle betrachtet, anhand derer die korrekte Funktionsweise der Modelltransformation nachvollzogen werden kann. Anschließend wird die Modellerstellung für die DCP-Modelle erklärt. Auch diese werden wieder in Basisfälle und strukturell interessante Fälle eingeteilt, wobei Basisfälle zuerst erklärt werden.

8.2.3.1. UMLsec-Modelle

Im Folgenden wird die Erzeugung der Basisfälle der UMLsec-Eingabemodelle erklärt. Es wurde bereits in Kapitel 5 darauf hingewiesen, dass es möglich ist, ein Eingabemodell an die Secure-Links-Analyse in mehrere einzelne Secure-Links-Diagramme zu zerlegen, sodass das Analyseergebnis des vollständigen Modells identisch ist zu der Vereinigung der Analyseergebnisse der Analysen auf den zerlegten Modellen (siehe Abbildung 5.2 und Abbildung 5.3). Für ein durch die Zerlegung entstandenes, minimales Modell, nachfolgend Basisfall genannt, gilt also, dass es über genau zwei Artefakte und genau eine Abhängigkeit zwischen diesen Artefakten verfügt. Ferner existieren genau zwei Rechnerknoten, die über genau einen Kommunikationspfad miteinander verbunden sind. Die beiden Artefakte sind jeweils auf einen dieser Rechnerknoten verteilt.

Für die Modellerstellung bedeutet das, dass alle möglichen Stereotypbelegungen für den Basisfall geprüft werden müssen. Für den Kommunikationspfad kann kein Stereotyp annotiert sein oder ein Stereotyp

$$r \in \{\texttt{«Internet»}, \texttt{«encrypted»}, \texttt{«LAN»}, \texttt{«wire»}\}.$$

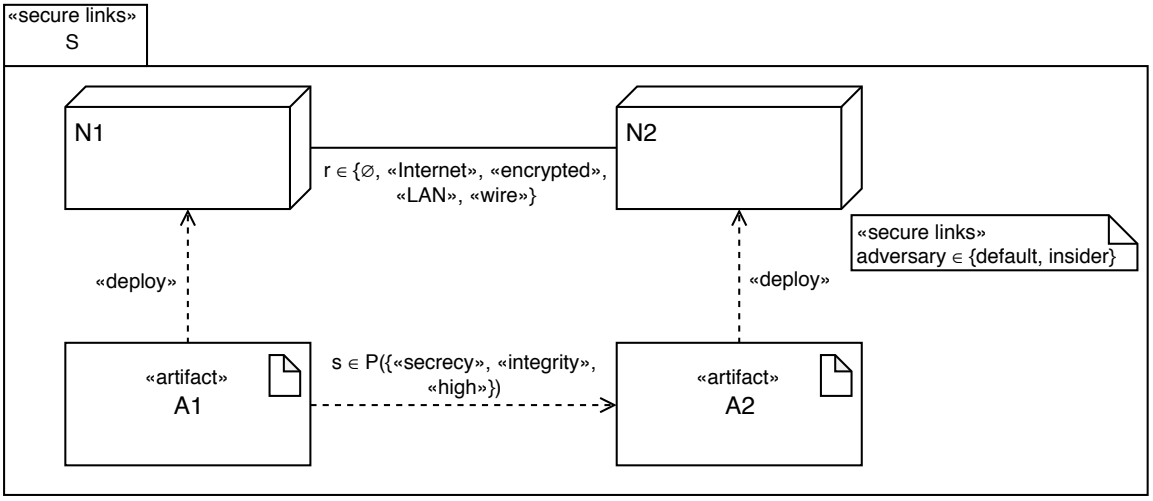

Abbildung 8.1.: Die Struktur der Basisfälle der Secure-Links-Analyse.

Es existieren also fünf verschiedene Möglichkeiten für die Annotation des UMLsec-Stereotyps an einem Kommunikationspfad. Für die Abhängigkeit kann dagegen entweder kein Stereotyp annotiert sein oder mindestens ein Stereotyp aus der Menge

$$\{\text{«secrecy»}, \text{«integrity»}, \text{«high»}\}.$$

Für die Menge s von Stereotypen, die an der Abhängigkeit annotiert sind, gilt also

$$s \in \mathcal{P}(\{\text{«secrecy»}, \text{«integrity»}, \text{«high»}\}).$$

Es können also hier acht mögliche, verschiedene Mengen von Stereotypen an einer Abhängigkeit annotiert werden. Des Weiteren kann einer von zwei definierten Angreifern für das Modell gewählt werden. Es entstehen so also $5 \cdot 8 \cdot 2 = 80$ mögliche Basisfälle. Die Struktur der Basisfälle ist in Abbildung 8.1 dargestellt.

Um auch sicherzustellen, dass die DCP-Analyse für Kompositionen dieser Basisfälle das richtige Verhalten aufweist, muss geprüft werden, dass auch strukturell komplexere UMLsec-Ausgangsmodelle richtig transformiert werden. Dabei ist eine Veränderung von Artefakten, Rechnerknoten, Kommunikationspfaden oder Verteilungsbeziehungen nicht interessant, da diese Modellelemente eine rechtseindeutige Relation zu ihren korrespondierenden Modellelemente in DCP besitzen. Die Abhängigkeiten zwischen Artefakten dagegen legen den Datenfluss in DCP fest und haben in manchen Situationen die Erzeugung einer zusätzlichen Komponenteninstanz als Folge. Es gibt folgende Möglichkeiten für die Abhängigkeiten eines Artefakts.

S1 Ein Artefakt hat keine Abhängigkeiten und ist auch nicht das Ziel von Abhängigkeiten anderer Artefakte.

S2 Ein Artefakt hat n Abhängigkeiten, aber ist nicht das Ziel von Abhängigkeiten anderer Artefakte.

S3 Ein Artefakt hat keine Abhängigkeiten, aber ist das Ziel von m Abhängigkeiten anderer Artefakte.

S4 Ein Artefakt hat n Abhängigkeiten und ist das Ziel von m Abhängigkeiten anderer Artefakte.

Für jeden dieser strukturellen Fälle wird ein Modell erstellt, in dem ein Artefakt existiert, das der geforderten Struktur bezüglich den Abhängigkeiten entspricht. In den Fällen S2 und S3 wird jeweils ein Artefakt eingeführt, dass die strukturelle Bedingung erfüllt und für das $n = 2$ gilt. Der Fall $n = 1$ wird bereits von den Basisfällen abgedeckt. In Fall S4 werden zwei Artefakte, die die strukturelle Bedingung erfüllen, eingefügt. Das eine dieser Artefakte erfüllt die Bedingung für $n = m = 1$, während das andere die Bedingung für $n = m = 2$ erfüllt. Für den vierten Fall ist das Modell in Abbildung 8.2 dargestellt. Es ist zu erkennen, dass mit dem Artefakt $A3$ ein Artefakt existiert, das jeweils eine Abhängigkeit zu den Artefakten $A5$ und $A6$ hat. Ferner ist Artefakt $A3$ das Ziel der Abhängigkeiten von den Artefakten $A1$ und $A2$. Für das Artefakt $A3$ gilt also $n = m = 2$. Des Weiteren ist mit Artefakt $A4$ ein Artefakt definiert, für das $n = m = 1$ gilt.

Schließlich wird ein zusätzliches Modell definiert, das den Fall beschreibt, in dem ein Artefakt a eine Abhängigkeit zu einem Artefakt b hat, Artefakt b aber über mehr Verteilungsbeziehungen verfügt, als Artefakt a. Dies entspricht dem beschriebenen Sonderfall, in dem zusätzliche Komponenteninstanzen in DCP erstellt werden müssen und muss daher auch evaluiert werden. Im Folgenden wird dieses Modell als fünfter struktureller Fall bezeichnet und mit S5 referenziert.

Für die Modelle der strukturellen Fälle werden die UMLsec-Stereotypen, die an Abhängigkeiten und Kommunikationspfaden annotiert sind, so gewählt, dass jedes Modell mindestens über ein Sicherheitsproblem verfügt. Bei dem Angreifer wird sich auf den Standardangreifer beschränkt, da er unterschiedliche Angreiferaktionen für verschiedene Kommunikationspfadtypen durchführen kann und nicht immer entweder alle Angreiferaktionen oder keine zur Verfügung hat, wie das beim Insiderangreifer der Fall ist. Die Definition eines Angreifers ist nur für die UMLsec-Modelle notwendig, da hier der Angreifer ein Teil des Eingabemodells ist.

8.2.3.2. DCP-Modelle

Für die DCP-Modelle gilt ebenfalls das Konzept der Zerlegung in separate Basisfälle, da jede Komponenteninstanz das benötigte Datum unabhängig von anderen Komponenteninstanzen erzeugt und über die entsprechende Netzwerkressource überträgt. Da in DCP der Angreifer nicht Teil des Eingabemodells ist, existieren jedoch nur 40 Basisfälle, die modelliert werden. Analog können die vier interessanten Strukturen in DCP umgesetzt werden.

T1 Eine Komponenteninstanz ist weder die Quelle (requiring assembly context) noch das Ziel (providing assembly context) eines Komponenteninstanzkonnektors.

T2 Eine Komponenteninstanz ist das Ziel von n Komponenteninstanzkonnektoren, aber die Quelle von keinem.

T3 Eine Komponenteninstanz ist die Quelle von n Komponenteninstanzkonnektoren, aber das Ziel von keinem.

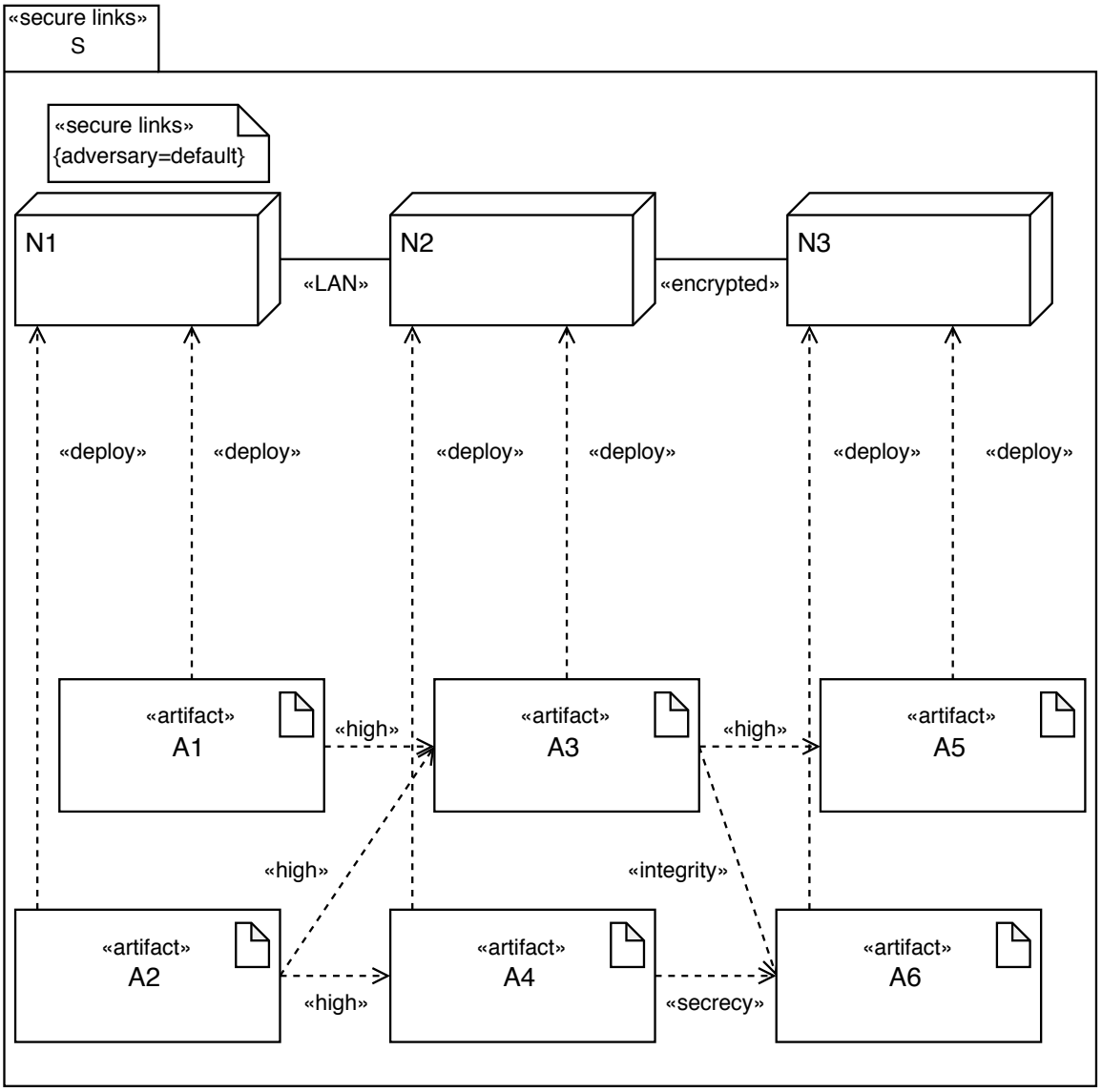

Abbildung 8.2.: Ein UMLsec-Modell der Secure-Links-Analyse, das dazu dient, die Transformation von komplexeren Strukturen zu evaluieren.

T4 Eine Komponenteninstanz ist die Quelle von n und das Ziel von m Komponenteninstanzkonnektoren.

Da die Struktur des Systems in PCM auf der Systemebene über Komponenteninstanzkonnektoren bestimmt wird, ist hier die Betrachtung der Komponenteninstanzen sinnvoller, als die Betrachtung der Komponenten. Die Anzahl an zur Verfügung gestellten Rollen einer Komponente gibt keinen Aufschluss darüber, wie viele andere Komponenten tatsächlich diese Rollen konsumieren. Analog zu den UMLsec-Modellen werden die Modelle so erstellt, dass in den strukturellen Fällen T2 und T3 je eine Komponenteninstanz existiert, für die $n = 2$ gilt und in Fall T4 zwei Komponenteninstanzen existieren, wobei für eine $n = m = 1$ gilt und für die andere $n = m = 2$. Die Charakteristiken werden so erstellt, dass sie zu den Stereotypen der entsprechenden Modelle in UMLsec korrespondieren.

Es werden also insgesamt 85 UMLsec-Modelle erstellt und 44 DCP-Modelle. Die UMLsec-Modelle setzen sich zusammen aus 80 Basisfällen und den fünf strukturellen Fällen S1 bis S5. Bei den DCP-Modellen existieren 40 Basisfälle und vier strukturelle Fälle T1 bis T4. Zur Erstellung der Modelle wird Eclipse verwendet. Für die UMLsec-Modelle wird die Papyrus-Modellierungsumgebung und für die DCP-Modelle die um DCP erweiterte PCM-Modellierungsumgebung eingesetzt. Es existiert ein Datensatz der erstellten Modelle [44]. Abschließend ist zu erwähnen, dass es sich bei den Basisfällen um die minimalen Modelle für die Secure-Links-Analyse in sowohl UMLSec als auch DCP handelt und daher zu erwarten ist, dass die Modelltransformation Modelle erzeugt, die identisch zu bereits erstellten Modellen sind.

8.2.4. Ergebnisse

In diesem Unterabschnitt werden die Ergebnisse der Evaluation präsentiert. In Unterunterabschnitt 8.2.4.1 und Unterunterabschnitt 8.2.4.2 werden jeweils die Fragen F1.1 und F1.2 beantwortet, während in Unterunterabschnitt 8.2.4.3 die Ergebnisse der Experimente, die zur Beantwortung der Fragen F2.1 bis F2.4 verwendet werden, vorgestellt sind. Abschließend werden die Ergebnisse in Unterunterabschnitt 8.2.4.4 zusammengefasst und diskutiert.

8.2.4.1. Rückschluss auf verantwortliche Modellelemente (F1.1)

Für die Beantwortung der Frage F1.1 sei zunächst die Ausgabe des CARiSMA-Werkzeugs für die Secure-Links-Analyse betrachtet. Dieses gibt für jedes gefundene Sicherheitsproblem den Angreifer, den angegriffenen Kommunikationspfad und die für diesen Angreifertyp und Kommunikationspfad erlaubten Angreiferaktionen aus. In der Ausgabe der DCP-Analyse sind ebenfalls der Angreifer und die Angreiferaktionen enthalten. Die Netzwerkressource ist nicht direkt in der Ausgabe enthalten, dafür aber die Operation im Operationsmodell, an der die Sicherheitsverletzung gefunden wird. Im Folgenden wird gezeigt, dass aus der ausgegebenen Operation die dazugehörige Netzwerkressource eindeutig ermittelt werden

kann und damit der Rückschluss auf die verantwortlichen Modellelemente in der DCP-Analyse ebenfalls möglich ist.

In der DCP-Analyse wird bei Abfrage des Prolog-Prädikats *secureLinks\6* zunächst das Prädikat *hasLinkWithDataSensitivity\4* abgefragt, das als Generator fungiert. Es liefert alle Operationen, für die ein Wert der *Link*-Eigenschaft gesetzt ist und die über entweder Eingabeparameter oder Rückgabevariablen verfügen, an denen ein Wert des *Sensitivity*-Attributs gesetzt ist. Die *Link*-Charakteristik, aus der die *Link*-Eigenschaft abgeleitet wird, ist im Eingabemodell nur an Netzwerkressourcen annotiert. Als Teil der Transformation des Eingabemodells in das Operationsmodell wird für eine Netzwerkressource l, die Rechnerknoten n und m verbindet, genau eine Operation (DataTransmissionProxyOperation) für jeden separaten Datenfluss, der von einer Komponenteninstanz auf dem Rechnerknoten n (bzw. m) zu einer Komponenteninstanz auf Rechnerknoten m (bzw. n) fließt, erzeugt. Während also zu einer Netzwerkressource mehrere zu ihr korrespondierende Operationen im Operationsmodell existieren können, existiert zu einer solchen Operation stets nur eine Netzwerkressource. Das Prädikat *hasLinkWithDataSensitivity\4* liefert also nur solche Operationen, die eindeutig zu einer Netzwerkressource zurückverfolgt werden können und als Repräsentation selbiger im Operationsmodell dienen. Die zweite Abfrage des Prädikats *secureLinks\6* ist das Prädikat *secureLinksViolation\4* und ist der eigentliche Kern der Analyselogik. Dieses Prädikat ist abhängig von dem Wert der *Link*-Eigenschaft der zu betrachteten Operation. Diese Variable ist aber durch die vorherige Abfrage an *hasLinkWithDataSensitivity\4* bereits gebunden. Daher kann *secureLinksViolation\4* also nur Sicherheitsprobleme an Operationen erkennen, deren korrespondierende Parameterbelegung durch *hasLinkWithDataSensitivity\4* generiert wird. Damit ist also gezeigt, dass Sicherheitsprobleme nur an Operationen gefunden werden können, die das Konzept der Netzwerkressource im Operationsmodell repräsentieren und dass für diese Operationen jeweils die dazugehörige Netzwerkressource eindeutig ermittelt werden kann. Ein Rückschluss auf die für ein Problem verantwortlichen Modellelemente ist also möglich. Ferner kann dieser Rückschluss automatisiert werden, sodass dem Nutzer automatisch als Analyseergebnis die entsprechende Netzwerkressource angezeigt wird, anstelle der Operation des Operationsmodells, mit dem der Nutzer nicht direkt interagiert.

8.2.4.2. Semantik der Analyseformel (F1.2)

Im Folgenden wird argumentiert, dass die Semantik der Prolog-Prädikate, die die DCP-Analyseformel darstellen, äquivalent ist zu der Semantik den UMLsec-Secure-Links-Bedingungen. Dazu wird die UMLsec-Analyse in ihre zwei hauptsächlichen Bestandteile zerlegt: Die Iteration über die für die Analyse relevanten Modellelemente und die Überprüfung der Secure-Links-Bedingung, sobald eine Menge von zusammengehöriger für die Analyse relevanter Modellelemente gefunden wurde. Anhand dieser beiden Bedingungen kann argumentiert werden, dass die definierten Prolog-Regeln semantisch dieselben Bedingungen darstellen, wie die UMLsec-Secure-Links-Bedingungen und dass die Modellelemente, auf denen die Prolog-Regeln die Bedingungen überprüfen, semantisch äquivalent sind zu den Modellelementen, auf denen die UMLsec-Analyse die Bedingungen überprüft. Es folgt

zunächst eine Wiederholung der Funktionsweise der Secure-Links-Analyse, in der die für die Argumentation relevanten Aspekte hervorgehoben sind. Anschließend wird argumentiert, dass die Prolog-Regeln, dieselben Bedingungen wie die UMLsec-Analyse überprüfen. Schließlich wird gezeigt, dass die Modellelementen auf denen diese Bedingungen jeweils überprüft werden zueinander äquivalent bezüglich ihrer Semantik sind.

Überblick über die UMLsec-Analyse

Der erste Schritt der Analyse ist die Iteration über die vorhandenen Modellelemente, um die Elemente zu identifizieren, die für den zweiten Schritt der Analyse relevant sind. Im zweiten Schritt werden die definierten Bedingungen der Secure-Links-Analyse für die im ersten Schritt identifizierten Modellelemente überprüft. Für die Bedingungsüberprüfung ist es ausreichend, die zusammengehörenden Abhängigkeits-Kommunikationspfad-Tupel zu kennen. Weitere Informationen, wie die ursprünglichen Artefakte oder die Rechnerknoten, auf denen sie verteilt sind, sind für die Bedingungsüberprüfung nicht relevant.

Für jede Abhängigkeit zwischen Artefakten, sofern die Abhängigkeit über einen Stereotyp $s \in \{$«`secrecy`», «`integrity`», «`high`»$\}$ verfügt, werden die korrespondierenden Kommunikationspfade ermittelt. Ein Kommunikationspfad c korrespondiert dann zu einer Abhängigkeit d von einem Artefakt a zu einem Artefakt b, wenn die Rechnerknoten n bzw. m, auf die a bzw. b verteilt sind, durch c verbunden werden. Es entstehen also Tupel der Form $R = \{(d_i, c_j)\}$, wobei d_i eine Abhängigkeit und c_j einen Kommunikationspfad beschreiben. Es ist anzumerken, dass formal noch der Angreifertyp A und das System S notwendig sind. Diese sind aber konstant für den gesamten Ablauf der Analyse und werden an dieser Stelle als global verfügbare Variablen behandelt.

Der zweite Teil der Analyse ist die Überprüfung der Einhaltung der Secure-Links-Bedingungen. Diese Bedingungen werden für jedes Tupel in R geprüft und sind zur besseren Verständlichkeit nachfolgend erneut dargestellt.

1. Wenn d_i über den Stereotyp «`high`» verfügt, dann $\text{threats}_A^S(c_j) = \varnothing$
2. Wenn d_i über den Stereotyp «`secrecy`» verfügt, dann `read` $\notin \text{threats}_A^S(c_j)$
3. Wenn d_i über den Stereotyp «`integrity`» verfügt, dann `insert` $\notin \text{threats}_A^S(c_j)$

Semantik der Bedingungen

Im Folgenden wird zunächst argumentiert, dass die umgesetzten Prolog-Regeln den definierten Bedingungen der Secure-Links-Analyse entsprechen. Im Anschluss wird darauf eingegangen, dass die jeweiligen Modellelemente, auf denen die Bedingungen überprüft werden, im Kontext der Secure-Links-Analyse über eine äquivalente Semantik verfügen.

Da die Prolog-Analyse Parameterbelegungen findet, in denen ein Sicherheitsproblem vorliegt, ist es sinnvoll, in einem ersten Schritt die UMLsec-Analysebedingungen zu negieren, sodass sie nicht mehr die Absenz eines Sicherheitsproblems im Kontext der Secure-

Links-Analyse zeigen, sondern das Auftreten eines Sicherheitsproblems. Diese Umformung ergibt folgende Formeln, wobei c den Kommunikationspfad, d die Abhängigkeit und $stereotypes(d)$ die Menge von Stereotypen von d beschreibt.

1. $f_{high}(c, d) \coloneqq$ «high» $\in stereotypes(d) \wedge \text{threats}_A^{\mathcal{S}}(c) \neq \varnothing$
2. $f_{secrecy}(c, d) \coloneqq$ «secrecy» $\in stereotypes(d) \wedge$ `read` $\in \text{threats}_A^{\mathcal{S}}(c)$
3. $f_{integrity}(c, d) \coloneqq$ «integrity» $\in stereotypes(d) \wedge$ `insert` $\in \text{threats}_A^{\mathcal{S}}(c)$

Diese Bedingungen sind in der Prolog-Formel der DCP-Analyse umgesetzt, welche bereits erläutert wurde. Da viele der Parameter der Funktion nur dazu dienen, dem Nutzer eine sinnvolle Ausgabe zu präsentieren, ist nachfolgend eine Darstellung derselben Logik ohne die Ausgabeparameter gegeben. Bei s handelt es sich hierbei um den Wert des *Sensitivity*-Attributs eines Datums, das über die untersuchte Operation fließt. l entspricht der *Link*-Eigenschaft der untersuchten Operation.

1. $g_{high}(s, l) \coloneqq s = High \wedge attacker_A(l) \neq \varnothing$
2. $g_{secrecy}(s, l) \coloneqq s = Secrecy \wedge$ `read` $\in attacker_A(l)$
3. $g_{integrity}(s, l) \coloneqq s = Integrity \wedge$ `insert` $\in attacker_A(l)$

Es ist bereits zu erkennen, dass die Bedingungen, die überprüft werden, sich in beiden Ansätzen stark ähneln. Sie unterscheiden sich hauptsächlich in der Funktion, die verwendet wird, um die Angreiferaktionen zu bestimmen und in den Eingaben. In der DCP-Analyse werden direkt die entsprechenden *Link*- bzw. *Sensitivity*-Charakteristiken eingegeben. Wie bereits in der Beantwortung von Frage F1.1 erklärt (siehe Unterunterabschnitt 8.2.4.1), ist dies deshalb möglich, da die Operation, auf die sich diese Variablen beziehen, bereits in einer vorherigen Abfrage der Analyse gebunden wird. Eine explizite Referenz ist daher an dieser Stelle nicht notwendig. Durch einen Wechsel von dieser impliziten Bindung der Variable zu einem expliziten Eingabeparameter kann eine zu f ähnliche Struktur erreicht werden, in der die Eingaben die annotierten Modellelemente sind und nicht die Annotationen selbst. Im Folgenden wird eine Umformung an dem Prolog-Programm und der Formel g vorgenommen, um dies zu erreichen. Dadurch ist es möglich, eine direkte Aussage über die Gleichheit der Struktur der beiden Ansätze abzuleiten. Die Umformung ist eine rein syntaktische Änderung, die die Eingabeparameter ändert, und hat keinerlei Einfluss auf die Logik des Prolog-Programms.

Im Folgenden ist o eine Operation, a der betrachtete Eingabeparameter oder Rückgabewert und $sensitivityCharacteristics(a)$ eine Funktion, die die Attributswerte des Typs *Sensitivity* des Datums a zurückgibt. Weiter sei eine Umformung der Funktion $attacker_A(l)$ gegeben, sodass die neue Funktion $attacker_A(o)$ eine Operation als Eingabe bekommt, den *Link*-Eigenschaftswert l der Operation extrahiert und anschließend die alte Funktion $attacker_A(l)$ mit l aufruft und das Ergebnis zurückgibt.

1. $g_{high}(o, a) \coloneqq High \in sensitivityCharacteristics(a) \wedge attacker_A(o) \neq \varnothing$
2. $g_{secrecy}(o, a) \coloneqq Secrecy \in sensitivityCharacteristics(a) \wedge$ `read` $\in attacker_A(o)$
3. $g_{integrity}(o, a) \coloneqq Integrity \in sensitivityCharacteristics(a) \wedge$ `insert` $\in attacker_A(o)$

Dies ist in Algorithmus 8.1 als Auszug eines Prolog-Programms dargestellt, um zu zeigen, dass es sich hierbei nicht um eine rein theoretische Betrachtung handelt. Diese Umformung

geht zu Lasten der Performanz und Lesbarkeit, da nun an mehreren Stellen die Prolog-Schnittstelle abgefragt wird, um Informationen zu bekommen, die schon verfügbar sind. Abgesehen davon verhält sie sich äquivalent zu dem in dieser Arbeit definierten, schlankeren Prolog-Programm. Der Auszug zeigt die Prädikate für die Vertraulichkeitsbedingung für die Rückgaberichtung des Datenflusses und die Definition des Standardangreifers. In Zeile 15 bis 18 wird erneut der *Sensitivity*-Attributswert des betrachteten Datums ermittelt. Analog wird in Zeile 25 erneut die *Link*-Eigenschaft der betrachteten Operation abgefragt.

Nun müssen noch die Ausgaben von $\text{threats}_A^S(c)$ und $attacker_A(o)$ betrachtet werden. Die Funktion $\text{threats}_A^S(c)$ nimmt einen Kommunikationspfad c als Eingabe, bestimmt den dazugehörigen Stereotypen des Kommunikationspfads und gibt abhängig von diesem und dem Angreifer A eine Menge von Angreiferaktionen zurück. Die Funktion $attacker_A(o)$ dagegen nimmt eine Operation als Eingabe entgegen und gibt abhängig von dieser und dem Angreifer A eine Menge von Angreiferaktionen zurück. In beiden Fällen sind die Ausgaben der Funktionen bezüglich ihrer Semantik identisch und die Korrektheit ergibt sich aus den äquivalenten Definitionen der Angreifer in den beiden Analysen, welche für den Standardangreifer für UMLsec und DCP bereits in Kapitel 5 erläutert ist. Der Insiderangreifer ist ebenfalls äquivalent definiert, wie anhand des Datensatzes [44] nachvollzogen werden kann. Er ist z. B. in der Datei `Evaluation/models/secure-links-evaluation-dcp-1/analysis.pl` definiert.

Zusammenfassend wurde durch eine Reihe von Äquivalenzumformungen und logischen Schlüssen gezeigt, dass auf einer strukturellen Ebene die definierten logischen Analysebedingungen äquivalent sind. Weiter ist ersichtlich, dass die Definition des Angreifers und die Rückgabewerte der Funktionen, die für einen Angreifer die möglichen Angreiferaktionen bestimmen, semantisch identisch sind.

Semantik der Modellelemente

Um schließlich die Frage F1.2 zu beantworten und eine Aussage über die Gleichheit der Semantik der UMLsec-Bedingungen und des Prolog-Programms treffen zu können, müssen die unterschiedlichen Modellelemente, die als Eingabe an f bzw. g dienen, betrachtet werden. Im Folgenden wird zunächst gezeigt, dass die Operation, die von der Prolog-Analyse geliefert wird, bezüglich ihrer Semantik äquivalent zu dem Kommunikationspfad der UMLsec-Analyse ist. Anschließend wird gezeigt, dass im Kontext der Secure-Links-Analyse die Daten, an denen in DCP die Sicherheitsanforderungen annotiert sind, bezüglich ihrer Semantik äquivalent sind zu den Abhängigkeiten zwischen Artefakten, an denen die Sicherheitsanforderungen in UMLsec annotiert sind.

Es wurde bereits in Unterunterabschnitt 8.2.4.1 gezeigt, dass eine der in dieser Analyse betrachteten Operationen immer genau zu einer Netzwerkressource zuzuordnen ist. Eine solche Operation stellt daher eine Repräsentation einer physischen Verbindung zwischen Rechnern im Operationsmodell dar. Es ist daher unmittelbar ersichtlich, dass diese Operation im Kontext der Secure-Links-Analyse die gleiche Semantik aufweist, wie

```
1  secureLinks(ATT, OP, D, ACT, L, DS) :-
2      hasLinkWithDataSensitivity(OP, L, DS, D, DATA),
3      secureLinksViolation(ATT, ACT, OP, DATA).
4
5  hasLinkWithDataSensitivity(OP, L, DS, D, DATA) :-
6      isOperation(OP),
7      operationProperty(OP, 'Link', L),
8      S=[OP|_],
9      stackValid(S),
10     operationReturnValue(OP, DATA),
11     D = 'ReturnValue',
12     returnValue(S, DATA, 'Sensitivity', DS).
13
14 secureLinksViolation(ATT, ACT, OP, DATA) :-
15     S = [OP|_],
16     stackValid(S),
17     operationReturnValue(OP, DATA),
18     returnValue(S, DATA, 'Sensitivity', DS),
19     DS = 'Secrecy',
20     attacker(ATT, ACT, OP),
21     is_List(ACT),
22     member('read', ACT).
23
24 attacker(ATT, ACT, OP) :-
25     operationProperty(OP, 'Link', L),
26     attackerInternal(ATT, ACT, L).
27
28 attackerInternal(
29     'default',
30     ['read', 'insert', 'delete'],
31     'Internet'
32 ).
33 attackerInternal('default', ['delete'], 'Encrypted').
```

Algorithmus 8.1.: Ein Auszug aus der Modifikation des Prolog-Programms, um die Parallelen zu der UMLsec-Analyse aufzuzeigen.

die Kommunikationspfade, die als Eingabe an die Bedingungsüberprüfung der UMLsec-Analyse dienen.

Schließlich bleiben die Abhängigkeitsbeziehungen zu betrachten. Die Stereotypen aus UMLsec, welche an Abhängigkeitsbeziehungen angebracht sind, stellen die Sicherheitsanforderungen an die Daten dar, die zwischen den Artefakten kommuniziert werden. In DCP sind diese Sicherheitsanforderungen an den Daten selbst annotiert. Die Überprüfung findet allerdings nur an den Stellvertreteroperationen statt. Diese Operationen werden von der Transformation ins Operationsmodell nur genau dann erstellt, wenn ein Datenfluss eines externen Aufrufs einer Komponente Daten an eine andere Komponente überträgt. Genauer werden diese Operationen nur zwischen Datenübertragungsoperationen (PerformDataTransmissionOperation) und der zu der aufgerufenen RDSEFF gehörigen Operation (ResourceDemandingSEFFOperation) erstellt. Für die Secure-Links-Analyse werden diese Daten also nur überprüft, wenn sie von einer Komponenteninstanz zur anderen übertragen werden. Im Kontext der Secure-Links-Analyse sind also die *Sensitivity*-Charakteristiken nur an den Schnittstellen zwischen Komponenten relevant. Dies entspricht der Semantik der Stereotypen der Abhängigkeitsbeziehungen in UMLsec. Eingeschränkt auf die Analyse weisen also die um Sicherheitsanforderungen annotierten Daten die gleiche Semantik auf, die auch von den um Sicherheitsanforderungen annotierten Abhängigkeitsbeziehungen aufgewiesen wird.

Abschließend ist zu erwähnen, dass ein semantischer Unterschied vorliegt, der in der bisherige Argumentation nicht betrachtet wurde. In DCP können unterschiedliche Sicherheitsanforderungen für die zwei Datenflussrichtungen einer Abhängigkeit vorliegen. Es können also unterschiedliche Sicherheitsanforderungen für Parameter und Rückgabevariablen vorliegen. Dies ist für diese Argumentation allerdings nicht von Relevanz, da die definierte Transformation ein UMLsec-Modell immer so auf ein DCP-Modell abbildet, dass für Parameter und Rückgabevariable stets dieselben Sicherheitsanforderungen gelten.

8.2.4.3. Ergebnisse der Experimente (F2.1 bis F2.4)

Die Evaluationsergebnisse der Experimente zur Beantwortung der Fragen F2.1 bis F2.4 sind in diesem Unterunterabschnitt dargestellt. Alle referenzierten Modelle, auch die, die als Teil der Evaluation durch eine Transformation erzeugt werden, sind in einem Datensatz [44] verfügbar. Die Basisfälle sind dabei von 1 bis 80 bzw. 40 durchnummeriert und verfügen über keinen weiteren Namenszusatz. Die strukturellen Fälle S1 bis S5 und T1 bis T4 verfügen über den Zusatz *structural* und schließlich verfügen die Modelle, die als Teil einer Transformation eines Experiments der Evaluation erstellt sind über den Zusatz *transformed*.

Die Transformation von UMLsec nach DCP und die anschließende Überprüfung der DCP-Analyse-Ergebnisse mit den ursprünglichen UMLsec-Analyse-Ergebnissen erfüllt für 82 der 85 betrachteten Eingabemodelle die Erwartungen. In diesen Fällen entspricht die Anzahl an erwarteten Sicherheitsverletzungen jeweils den tatsächlich ermittelten Sicherheitsverletzungen und zu jeder Sicherheitsverletzung, die von der UMLsec-Analyse

gefunden wird, wird die dazu äquivalente Sicherheitsverletzung in der DCP-Analyse gefunden. Die Fehlerrate ist demnach $f_{UMlsec \rightarrow DCP} = 0,0353$.

Die vollständigen Ergebnisse sind in Tabelle A.1 im Anhang dargestellt. Dabei beschreibt die erste Spalte das verwendete UMLsec-Eingabemodell und die zweite Spalte das DCP-Modell, das aus dem UMLsec-Eingabemodell transformiert wird. Die erwartete Anzahl von Sicherheitsverletzungen $\mathcal{E}$ ist ermittelt durch die Analyse in UMLsec und anschließende Verdopplung der gefundenen Anzahl von Sicherheitsverletzungen, da die DCP-Analyse jeweils Eingabeparameter und Rückgabevariablen als separate Sicherheitsverletzungen erkennt (siehe Unterabschnitt 8.2.2). Die Anzahl der ermittelten Sicherheitsverletzungen $\mathcal{R}$ beschreibt die tatsächliche Anzahl, die durch die DCP-Analyse ermittelt wird. Es gibt nur drei Eingabemodelle, in denen die Erwartungen nicht erfüllt werden. Bei diesen handelt es sich um die strukturellen Fälle S2, S4 und S5. In diesen Fällen wird je eine Sicherheitsverletzung in Eingabe- und Rückgaberichtung mehr erkannt, als erwartet wird. Die Gründe für diese Diskrepanz sind in Unterunterabschnitt 8.2.4.4 diskutiert.

Für das Experiment, das ein Ausgangsmodell in DCP verwendet und anschließend nach UMLsec transformiert, um die Ergebnisse der beiden Analysen miteinander zu vergleichen, entsprechen die UMLsec-Analyseergebnisse für 42 der 44 betrachteten DCP-Eingabemodelle den aufgestellten Erwartungen.

In diesen Fällen stimmen für beide erzeugte UMLsec-Modelle, die unter einem Angreifer A erkannten Sicherheitsverletzungen mit den Sicherheitsverletzungen überein, die von der DCP-Analyse für den Angreifer A ermittelt sind.

Dabei gilt weiter, dass zu jeder von der DCP-Analyse erkannten Sicherheitsverletzung eine äquivalente Sicherheitsverletzung durch die UMLsec-Analyse gefunden wird. Die beiden Eingabemodelle, für die das nicht der Fall ist, sind erneut die strukturellen Fälle T2 und T4. Die Fehlerrate ist damit $f_{DCP \rightarrow UMLsec} = 0,0455$.

Die vollständigen Ergebnisse sind in Tabelle A.2 dargestellt. Dabei ist in der ersten Spalte das DCP-Ausgangsmodell dargestellt und in der zweiten Spalte die beiden UMLsec-Modelle, die durch die Transformation erzeugt werden. Die erwartete Anzahl von Sicherheitsverletzungen $\mathcal{E}$ ist in dieser Tabelle durch ein 2-Tupel (a, b) gegeben, wobei a die erwartete Anzahl von Sicherheitsverletzungen unter dem Standardangreifer beschreibt und b unter dem Insiderangreifer. Dies gilt analog für die Anzahl von ermittelten Sicherheitsverletzungen $\mathcal{R}$. Die erwartete Anzahl von Sicherheitsverletzungen wird durch das Ausführen der DCP-Analyse und anschließendem Zusammenfassen von paarweisen Sicherheitsverletzungen, die lediglich die gleiche Verletzung einmal an Eingabeparameter und Rückgabevariablen darstellen, bestimmt.

Schließlich sind die Ergebnisse der Experimente zur Beantwortung von F2.3 und F2.4 dargestellt. Für beide Experimente gilt, dass die Analyseergebnisse in UMLsec (bzw. DCP) nicht beeinflusst werden durch eine Transformation nach DCP (bzw. UMLsec) und anschließender Rücktransformation nach UMLsec (bzw. DCP). Die Analyseergebnisse des Eingabemodells vor dieser Hin- und Rücktransformation und die Analyseergebnisse des Modells, das aus der Hin- und Rücktransformation entsteht, sind in allen betrachteten 85 (bzw. 44) Fällen identisch. In Tabelle A.3 und Tabelle A.4 sind die Ausgangsmodelle

in der ersten Spalte dargestellt, das bzw. die Modelle, die durch die erste Transformation entstehen, in der zweiten Spalte und das Modell das aus der Rücktransformation entsteht in der dritten Spalte. Für die Transformation von UMLsec nach DCP und zurück nach UMLsec ist relevant, dass sich hierbei der Angreifer gemerkt wird und dadurch die Transformation von DCP nach UMLsec eindeutig durchgeführt werden kann. Für die Transformation von DCP nach UMLsec und zurück nach DCP ist zu erwähnen, dass zwei UMLsec-Modelle erzeugt werden, für die als Teil der Rücktransformation jeweils ein DCP-Modell erzeugt wird. Diese beiden erzeugten DCP-Modelle sind in jedem Fall identisch, daher wird in Tabelle A.4 nur der Bezeichner eines DCP-Modells in der dritten Spalte dargestellt. Für die Fehlerraten ergibt sich also $f_{UMLsec \to DCP \to UMLsec} = f_{DCP \to UMLsec \to DCP} = 0$.

8.2.4.4. Zusammenfassung und Diskussion

In Unterunterabschnitt 8.2.4.1 und Unterunterabschnitt 8.2.4.2 wurde gezeigt, dass die in Prolog umgesetzte Analyse dieselbe Semantik aufweist wie die UMLsec-Analyse. Dies entspricht dem definierten Evaluationsziel Z1. Anschließend wurden in Unterunterabschnitt 8.2.4.3 die Ergebnisse präsentiert, die für die Beantwortung der zu dem Evaluationsziel Z2 gehörigen Fragen relevant sind. Im Folgenden sind die beiden strukturellen Fälle S2 und S4 bzw. T2 und T4 betrachtet, für die eine Diskrepanz zwischen den ermittelten und erwarteten Analyseergebnissen aufgetreten ist.

Für die einfachen Transformationen und Vergleiche der Analyse-Ergebnisse zwischen DCP-Analyse und UMLsec-Analyse wurden drei Eingabemodelle festgestellt, für die die Erwartung nicht mit dem Ergebnis übereinstimmt. Bei diesen Eingabemodellen handelt es sich um den strukturellen Fall S2 (bzw. T2), S4 (bzw. T4) und S5. Für die Fälle S2 und S4 hat die DCP-Analyse mehr Sicherheitsverletzungen erkannt als die UMLsec-Analyse. Bei der zusätzlich erkannten Sicherheitsverletzung handelt es sich um ein Duplikat einer bereits gefunden Sicherheitsverletzung. Für beide Fälle kommt dies vor, weil es ein Sicherheitsproblem ausgehend von einer Komponente gibt und die zur Verfügung gestellte Methode der Schnittstelle der Komponente von mehreren anderen Komponenten oder Nutzungsszenarien aufgerufen wird. In diesem Fall existieren im Operationsmodell mehrere zusammenhängende Datenflüsse ausgehend von verschiedenen Nutzungsszenarien, sodass mehrere Daten über die gleiche Datenübertragungsoperationen fließen. Somit existiert eine zusätzliche gültige Variablenbelegung für das Prolog-Prädikat, in der sich nur das Datum und die Aufrufhierarchie unterscheiden. Wenn das Datum über die gleichen Charakteristiken verfügt, wie andere Daten, die über die Datenübertragungsoperation fließen, erscheint das in der Prolog-Ausgabe als Duplikat der erkannten Sicherheitsverletzung. Dabei muss erwähnt werden, dass dieses Verhalten prinzipiell semantisch sinnvoll ist, da jedes Datum unterschiedliche Charakteristiken haben kann und daher nur manche übertragenen Daten eine Sicherheitsverletzung darstellen. Dies ist gerade einer der beschriebenen Vorteile von der Secure-Links-Analyse in DCP. Weiter ist zu erwähnen, dass diese duplizierte Sicherheitsverletzung problemlos herausgefiltert werden kann bevor die Ausgabe dem Nutzer präsentiert wird. Dabei müssen lediglich alle erkannten Parameterbelegungen, die sich nur in der Aufrufhierarchie und im Datum, nicht aber in den Charakteristiken

des Datums, unterscheiden, zu einer Sicherheitsverletzung zusammengefasst werden. Da sie sich nicht in der Operation unterscheiden, ist eindeutig, dass diese Sicherheitsverletzungen sich auf dieselbe Netzwerkressource beziehen und damit tatsächlich die gleiche Sicherheitsverletzung darstellen.

Im Folgenden sind die beiden DCP-Modelle, die zu den UMLsec-Modellen S2 bzw. S4 korrespondieren konkret betrachtet. In Fall T2 existieren zwei Datenflüsse über Komponente *C1*, daher wird der Aufruf von *C1* zu *C3*, der eine Sicherheitsverletzung darstellt, doppelt erkannt. Analog existieren in Fall T4 zwei Datenflüsse über Komponente *C3*, deren Aufruf an *C5* eine Sicherheitsverletzung darstellt und doppelt erkannt wird. Wenn diese dupliziert erkannten Sicherheitsverletzungen als jeweils eine Sicherheitsverletzung betrachtet werden, sinkt die Anzahl der tatsächlich ermittelten Sicherheitsverletzungen auf den erwarteten Wert.

Für den Fall S5 liegt eine andere Ursache für das zusätzliche Sicherheitsproblem vor. Dieser Fall tritt ein, wenn es ein Sicherheitsproblem ausgehend von einer Komponente gibt, von der mehrere Komponenteninstanzen im System existieren, aber auf denselben Rechnerknoten verteilt sind und über dieselbe Netzwerkressource mit anderen Komponenteninstanzen kommunizieren. Da das Operationsmodell und die Analyse auf Instanzebene arbeiten, wird für jede dieser Komponenteninstanzen eine eigene Datenübertragungsoperation erzeugt. In diesem Fall unterscheidet sich also die Prolog-Parameterbelegung um die Operation, das Datum und die Aufrufhierarchie. Da es sich semantisch trotzdem um die selbe Sicherheitslücke handelt, da dieselbe Komponente und dieselbe Netzwerkressource dafür verantwortlich sind, können auch solche duplizierten Sicherheitslücken zusammengefasst werden. Dazu muss für die potentiellen Duplikate die Netzwerkressource ermittelt werden, über die sie kommunizieren. Es ist bereits gezeigt, dass der Schluss von Operation auf dazugehörige Netzwerkressource möglich ist. Sind die bestimmten Netzwerkressourcen identisch, handelt es sich um den beschriebenen Fall und die Sicherheitsverletzung wurde mehrfach von der Analyse gefunden. Für beide Ursachen kann die duplizierte Sicherheitsverletzung also problemlos herausgefiltert werden, sodass für den Zweck dieser Evaluation diese duplizierten Sicherheitsverletzungen als ein und dieselbe Sicherheitsverletzung betrachtet werden können. Wenn auch in diesem Fall die dupliziert erkannte Sicherheitsverletzung als eine einzige Sicherheitsverletzung betrachtet wird, sinkt die Anzahl der tatsächlich ermittelten Sicherheitsverletzungen auf den erwarteten Wert. Damit ist die Fehlerrate $f_{UMLsec \to DCP} = f_{DCP \to UMLsec} = 0$.

8.3. Secure Dependencies

Für die Secure-Dependencies-Sicherheitsanalyse sind im Folgenden die Fragen und Metriken zu den Zielen Z1 und Z2 vorgestellt. Anschließend wird der Versuchsaufbau und die Erzeugung der Eingabemodelle erklärt. Abschließend werden die Ergebnisse präsentiert und diskutiert.

8.3.1. Fragen und Metriken

Im Folgenden sind zunächst die Fragen zu Evaluationsziel Z1 formuliert. Gegenstand der Untersuchung sind dabei die eingeführten Prolog-Prädikate, für die überprüft wird, ob sie der Semantik der UMLsec-Analysedefinition der Secure-Dependencies-Analyse entsprechen. Dabei ist zu erwähnen, dass die UMLsec-Analysedefinition zwei zu überprüfende Bedingungen spezifiziert. Wie bereits in Kapitel 6 erklärt, handelt es sich bei einer dieser Bedingungen jedoch um eine rein syntaktische Bedingung. Da diese Bedingung nicht in der DCP-Analyse umgesetzt wird, wird im Folgenden nur die zweite Bedingung der Secure-Dependencies-Analyse betrachtet. Diese Bedingung wird im Folgenden als semantische Bedingung der Secure-Dependencies-Analyse bezeichnet. Sie wird in eine logische Formel überführt, durch die eine Argumentation bezüglich der Äquivalenz zu den Prolog-Prädikaten möglich ist. Analog zu der Evaluation der Äquivalenz der Prolog-Prädikate von der Secure-Links-Analyse, geht durch diese Betrachtung die Information über die Modellelemente verloren, die für die Sicherheitsverletzung verantwortlich sind. Daher muss zusätzlich geprüft werden, ob die Prolog-Prädikate ebenfalls einen Rückschluss auf die für eine Sicherheitsverletzung verantwortlichen Modellelemente zulassen. Daher ergeben sich die folgenden Fragestellungen für die Evaluierung. Diese Fragen werden argumentativ beantwortet, daher sind für sie keine Metriken definiert.

F1.1 Erlaubt die Prolog-Formel den Rückschluss auf die Modellelemente, die im Fall einer Sicherheitsverletzung für diese Verletzung verantwortlich sind?

F1.2 Entspricht die Semantik der aufgestellten Prolog-Formel der Semantik der in UMLsec definierten semantischen Bedingung der Secure-Dependencies-Analyse?

Durch das Evaluationsziel Z2 wird überprüft, dass die Modelltransformation zu einem UMLsec-Eingabemodell ein äquivalentes DCP-Eingabemodell liefert. Wie auch schon in der Evaluation der Secure-Links-Analyse kommt dabei die in Z1 evaluierte Analyseformel zum Einsatz. Eine Schwierigkeit, die diese Evaluation beeinflusst, ist jedoch, dass die Secure-Dependencies-Analyse des CARiSMA-Werkzeugs fehlerhaft ist. Diese Analyse liefert als Ausgabe immer, dass das betrachtete Modell über keine Sicherheitsverletzungen verfügt, selbst wenn offensichtliche Verletzungen der definierten Bedingungen in das Modell eingefügt werden. Das CARiSMA-Werkzeug kann daher nicht als Referenz für die erwarteten Ergebnisse der Secure-Dependencies-Analyse verwendet werden. Das UML Analysis Tool implementiert ebenfalls die Secure-Dependencies-Analyse, allerdings in einer älteren Version, die nicht mit der vorgestellten Variante der Analyse übereinstimmt und insbesondere Unterschiede in der Syntax der Eingabemodelle aufweist. Da die Modelltransformation für die Syntax der aktuellen Definition der Secure-Dependencies-Analyse erstellt ist, kann die Modelltransformation nicht für Eingabemodelle verwendet werden, die die Syntax der älteren Version der Analyse verwenden. Aus diesem Grund wird für die Überprüfung des Evaluationsziels Z1 ein anderes Vorgehen gewählt, als in der Evaluation der Secure-Links-Analyse. Es werden mehrere Modelle erzeugt, für die aus der Definition der Secure-Dependencies-Analyse direkt ersichtlich ist, ob sie eine Sicherheitsverletzung enthalten. Diese Modelle werden unter Verwendung der definierten Modelltransformation nach DCP transformiert. Anschließend wird überprüft, ob die DCP-Analyse für jedes

Modell, das eine Sicherheitsverletzung enthält, eine Sicherheitsverletzung erkennt. Dies ist durch folgende Fragestellung beschrieben.

F2.1 Sei ein UMLsec-Eingabemodell gegeben, für das bekannt ist, ob es eine Sicherheitsverletzung enthält. Erkennt die Analyse in DCP mit dem nach DCP transformierten Eingabemodell genau dann eine Sicherheitsverletzung, wenn auch tatsächlich eine vorhanden ist?

Da ein direkter Vergleich mit den Ergebnissen einer Implementation der UMLsec-Analyse nicht möglich ist, kann im Falle einer Abweichung des Analyseergebnisses vom erwarteten Ergebnis nicht durch einen detaillierten Vergleich der UMLsec- und DCP-Analyseergebnisse die Ursache ermittelt werden, wie das in der Evaluation der Secure-Links-Analyse der Fall ist. Daher werden im Folgenden als Metriken die Sensitivität und Spezifität [45] verwendet. Die Sensitivität gibt den Anteil von Szenarien, in denen von der DCP-Analyse korrekt Sicherheitsverletzungen erkannt werden, an der Gesamtheit von Szenarien, in denen tatsächlich Sicherheitsverletzungen vorliegen, an. Analog gibt die Spezifität den Anteil von Szenarien, in denen von der DCP-Analyse korrekt keine Sicherheitsverletzungen erkannt werden, an der Gesamtheit von Szenarien, in denen tatsächlich keine Sicherheitsverletzungen vorliegen, an. Dies erlaubt es, die Genauigkeit der Analyse bezüglich Szenarien mit bzw. ohne Sicherheitsverletzungen getrennt zu bewerten.

8.3.2. Versuchsaufbau

Da die Fragestellungen F1.1 und F1.2 argumentativ beantwortet werden, wird für sie keine Erklärung eines Versuchsaufbaus benötigt. Daher wird im Folgenden der Versuchsaufbau zur Beantwortung der Frage F2.1 erklärt. In dieser Frage wird die Modelltransformation eines UMLsec-Eingabemodells nach DCP untersucht. Der nachfolgend beschriebene Versuch wird für mehrere UMLsec-Eingabemodelle durchgeführt. Dabei ist für jedes Eingabemodell bekannt, ob es Sicherheitsverletzungen enthält. Die systematische Erstellung dieser Modelle wird in Unterabschnitt 8.3.3 erklärt. Für ein gegebenes UMLsec-Eingabemodell wird die in Kapitel 6 definierte Modelltransformation durchgeführt, um aus dem UMLsec-Modell ein DCP-Eingabemodell zu erhalten. Für das entstehende DCP-Modell wird anschließend die Secure-Dependencies-Analyse in DCP ausgeführt. Dazu wird das DCP-Eingabemodell mittels einer Modell-zu-Modell-Transformation in das Operationsmodell überführt, aus dem mittels einer Modell-zu-Text-Transformation das Prolog-Programm erzeugt wird. Anschließend wird das Prolog-Prädikat *secureDependencies\4* des entstehenden Prolog-Programms abgefragt. Wenn keine Parameterbelegung für die beiden Parameter *OP* und *V* von *secureDependencies\4* gefunden werden kann, bedeutet dies, dass die DCP-Analyse keine Sicherheitsverletzung der Secure-Dependencies-Bedingungen ermitteln kann. Wenn jedoch eine Parameterbelegung zurückgegeben wird, liegt gemäß der Analyse eine Sicherheitsverletzung vor. Dabei ist *OP* die Operation an der die Sicherheitsverletzung festgestellt wird und *V* der Wert der Charakteristik *Critical*, der für die Verletzung verantwortlich ist. Handelt es sich bei dem UMLsec-Modell um ein Modell mit einer Sicherheitsverletzung und erkennt die DCP-Analyse eine Sicherheitsverletzung, wird dieses durch das UMLsec-Modell gegebene Szenario als richtig-positiv klassifiziert. Erkennt die

DCP-Analyse dagegen für ein solches UMLsec-Eingabemodell keine Sicherheitsverletzung, wird das Szenario als falsch-negativ klassifiziert. Analog werden die Ergebnisse für UMLsec-Modelle, die keine Sicherheitsverletzungen enthalten klassifiziert. Findet die DCP-Analyse für ein solches Modell eine Sicherheitsverletzung, wird dieses Szenario als falsch-positiv klassifiziert. Findet die DCP-Analyse keine Sicherheitsverletzung, wird es als richtig-negativ klassifiziert. Die Sensitivität $\frac{r_p}{P}$ wird anschließend über den Anteil der Menge der richtig-positiv klassifizierten Szenarien r_p an der Gesamtheit von Szenarien P, in denen Sicherheitsverletzungen vorliegen, bestimmt. Analog wird die Spezifität $\frac{r_n}{N}$ über den Anteil der Menge der richtig-negativ klassifizierten Szenarien r_n an der Gesamtheit von Szenarien N, in denen keine Sicherheitsverletzungen vorliegen, bestimmt.

8.3.3. Modellerstellung

Bei den Eingabemodellen für die Evaluation handelt es sich um UMLsec-Modelle. Dabei ist gefordert, dass für diese Modelle die Existenz einer Sicherheitsverletzung gemäß der Secure-Dependencies-Analyse anhand der Definition der Analyse ersichtlich ist. Es werden daher strukturell minimale Modelle erstellt, die sich voneinander nur durch die angewandten Stereotypen und Eigenschaftswerte der Eigenschaftsdefinitionen des Stereotyps «`Critical`» unterscheiden. Im Folgenden wird die Struktur der Modelle vorgestellt und aufgezeigt, welche Kombinationen von Stereotypen und Eigenschaftswerten möglich und relevant sind. Abschließend ist erklärt, wieso aus der Betrachtung dieser Modelle eine repräsentative Aussage gefolgert werden kann.

Das notwendige Minimum eines Modells für die Secure-Dependencies-Analyse benötigt eine Schnittstelle, die von einer Klasse realisiert wird und von einer anderen Klassen benötigt wird. Beide Klassen sind mit dem Stereotyp «`Critical`» annotiert und verfügen gegebenenfalls über Eigenschaftswerte für die Eigenschaftsdefinitionen `{secrecy}`, `{integrity}` oder `{high}`. Für die Analyse eines Modells dieser Struktur wird genau ein einziges Paar von Klassen auf die Secure-Dependencies-Bedingungen geprüft. Ein Modell dieser Art wird im Folgenden Basisfall genannt. Die Struktur dieser Modelle ist in Abbildung 8.3 dargestellt. In diesem Modell ist zu erkennen, dass die Klasse $K1$ eine Abhängigkeit auf die Schnittstelle I hat, welche von der Klasse $K2$ realisiert wird. Die Schnittstelle definiert eine Methode $m()$ ohne Parameter, aber mit einem Rückgabewert. Durch die Eigenschaftswerte des Stereotyps «`Critical`» der Klasse $K1$ können also die Sicherheitsanforderungen an die Rückgabevariable von der Methode $m()$ definiert werden. Die Eigenschaftswerte des Stereotyps «`Critical`» der Klasse $K2$ dagegen können die tatsächlichen Sicherheitseigenschaften der Rückgabevariable spezifizieren.

Für jede Kombination von gültigen Eigenschaftswerten für den Stereotyp «`Critical`» beider Klassen wird ein Modell erzeugt. Gültige Eigenschaftswerte sind solche, die der impliziten Syntax der Secure-Dependencies-Analyse folgen und nur entweder eine leere Menge darstellen, oder Namen von Methoden oder Parametern enthalten, die von einer Schnittstelle definiert sind, die von der Klasse, an der die Eigenschaftswerte annotiert sind, realisiert oder benötigt werden. Für die Modelle der in Abbildung 8.3 gegebenen Struktur sind die gültigen Eigenschaftswertbelegungen in Tabelle 8.1 dargestellt.

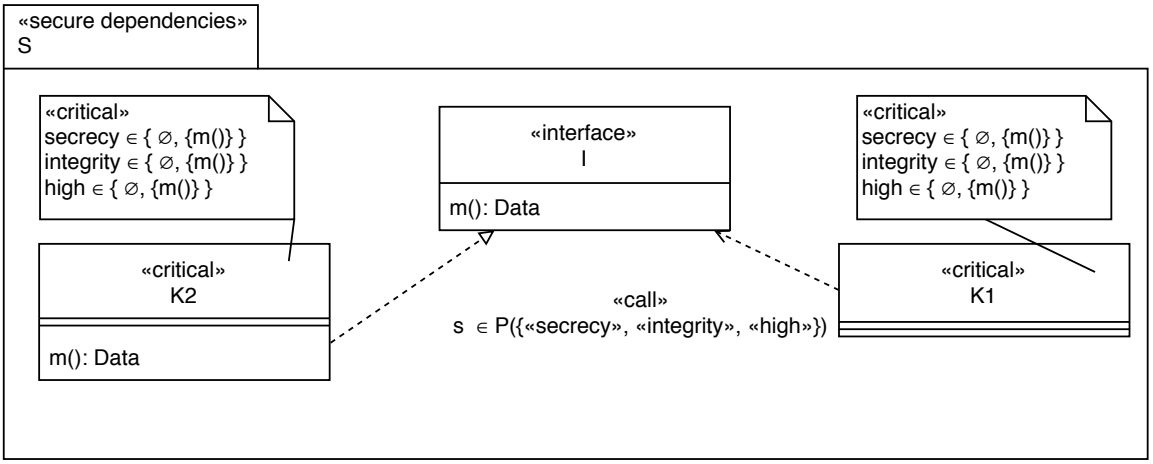

Abbildung 8.3.: Die Struktur der Basisfälle der Secure-Dependencies-Analyse für eine Schnittstelle mit einer Methode ohne Parameter.

Tabelle 8.1.: Tabellarische Darstellung der gültigen Eigenschaftswertbelegungen für den Stereotyp «`Critical`» in Modellen der in Abbildung 8.3 gegebenen Struktur.

`{secrecy}`	`{integrity}`	`{high}`
∅	∅	∅
$m()$	∅	∅
∅	$m()$	∅
∅	∅	$m()$
$m()$	$m()$	∅
$m()$	∅	$m()$
∅	$m()$	$m()$
$m()$	$m()$	$m()$

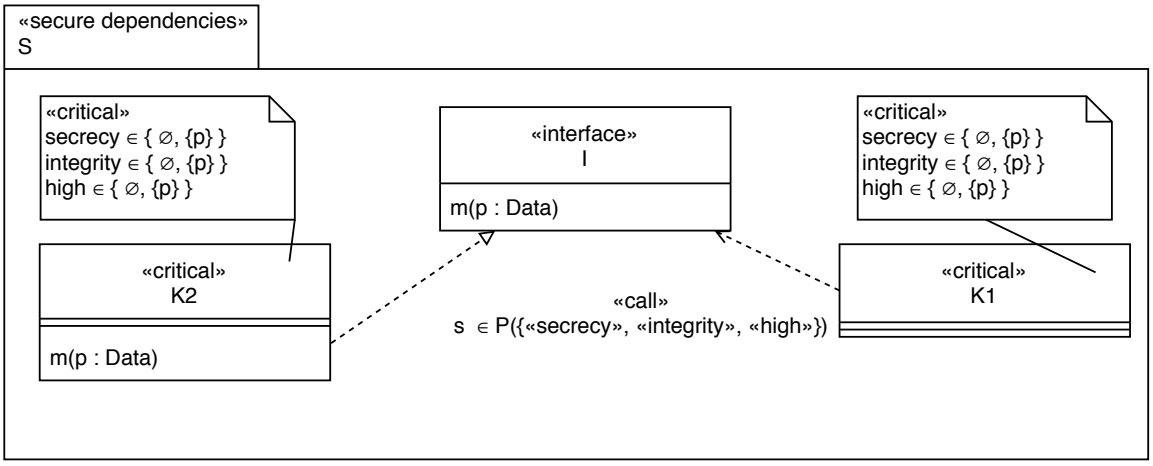

Abbildung 8.4.: Die Struktur der Basisfälle der Secure-Dependencies-Analyse für eine Schnittstelle mit einer Methode mit Parameter.

Da die Schnittstelle nur eine Methode ohne Parameter bereitstellt, existieren acht gültige Eigenschaftswertbelegungen für die Eigenschaftsdefinitionen des Stereotyps «`Critical`» einer Klasse dieser Modelle. In Kombination mit der zweiten Klasse sind somit $8 \cdot 8 = 64$ Modelle möglich, die sich durch die Belegung der Eigenschaftswerte unterscheiden. Dies sind die Modelle, die für die Evaluation erzeugt werden. Sie stellen alle möglichen Fälle dar, die von den Bedingungen der Secure-Dependencies-Analyse überprüft werden. Es ist ersichtlich, dass für eine feste Eigenschaftswertbelegung von der Klasse $K1$ nur eine Eigenschaftswertbelegung der Klasse $K2$ existiert, sodass die Eigenschaftswertbelegungen identisch sind und somit die Bedingung der Secure-Dependencies-Analyse erfüllen. Von den 64 Modellen enthalten also lediglich acht Modelle keine Sicherheitsverletzung. Dabei handelt es sich gerade um die, in denen die Eigenschaftswertbelegungen von den Klassen $K1$ und $K2$ identisch sind. Mit diesen Modellen werden jedoch nur Sicherheitsanforderungen an die Rückgabevariablen der Methode überprüft. Für die UMLsec-Analyse ist dies unerheblich, aber für die DCP-Analyse werden Sicherheitsanforderungen an Rückgabevariablen an anderen Datenverarbeitungsvorgängen annotiert, als Sicherheitsanforderungen an Parameter. Daher muss eine analoge Betrachtung für eine Menge von Modellen erfolgen, in denen die Schnittstelle über eine Methode verfügt, die einen Parameter erhält. Die Struktur dieser Modelle ist in Abbildung 8.4 dargestellt. Da die Methode $m(p : Data)$ über keine Rückgabevariable verfügt, existieren wieder acht mögliche Eigenschaftswertbelegungen für den Stereotyp «`Critical`». Diese sind in Tabelle 8.2 dargestellt. Es existieren also weitere $8 \cdot 8 = 64$ zu überprüfende Modelle. Insgesamt werden also 128 Modelle erstellt. Dabei ist zu erwähnen, dass der Stereotyp $s \in \mathcal{P}(\{\text{«}\texttt{secrecy}\text{»}, \text{«}\texttt{integrity}\text{»}, \text{«}\texttt{high}\text{»}\})$ der Abhängigkeit stets so gewählt ist, dass die syntaktische Bedingung der Secure-Dependencies-Analyse erfüllt wird. Nachfolgend wird kurz erklärt, wieso die Betrachtung dieser Modelle ausreichend ist, um eine repräsentative Aussage über die Genauigkeit der DCP-Analyse zu treffen.

Tabelle 8.2.: Tabellarische Darstellung der gültigen Eigenschaftswertbelegungen für den Stereotyp «`Critical`» in Modellen der in Abbildung 8.4 gegebenen Struktur.

`{secrecy}`	`{integrity}`	`{high}`
∅	∅	∅
p	∅	∅
∅	p	∅
∅	∅	p
p	p	∅
p	∅	p
∅	p	p
p	p	p

In Kapitel 6 wird bereits anhand eines Beispiels verdeutlicht, dass die Analyse eines strukturell komplexen Eingabemodells zurückgeführt werden kann auf die Komposition der Analyseergebnisse der minimalen Eingabemodelle, die aus dem komplexen Eingabemodell extrahiert werden können. Sei z. B. ein Modell gegeben, in dem zwei Klassen $K1$ und $K2$ eine Abhängigkeit auf eine Schnittstelle haben, die von einer dritten Klasse $K3$ realisiert wird. Dann überprüft die Analyse die Secure-Dependencies-Bedingungen jeweils für die Klassen $K1$ und $K3$ und für die Klassen $K2$ und $K3$. Die Analyse von zwei Modellen, wovon das eine nur $K1$, $K3$, die Schnittstelle und Relationen und das andere nur $K2$, $K3$, die Schnittstelle und Relationen enthält, liefert also zwei Analyseergebnisse, die vereinigt das Analyseergebnis des ursprünglichen Modells darstellen. Diese extrahierten zwei Modelle entsprechen der in Abbildung 8.3 und Abbildung 8.4 vorgestellten Struktur. Es ist also ersichtlich, dass wenn die DCP-Analyse für diese Basisfälle korrekte Ergebnisse liefert, sie auch in der Lage ist, für komplexere Modelle korrekte Ergebnisse zu liefern, da das komplexe Modell vor der Transformation in die einzelnen Basisfälle zerlegt werden kann.

Die UMLsec-Modelle, die die definierten Basisfälle darstellen, und die DCP-Modelle, die aus der Transformation der Basisfälle entstehen, sind in einem Datensatz [44] verfügbar.

8.3.4. Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Evaluation präsentiert. In Unterunterabschnitt 8.3.4.1 und Unterunterabschnitt 8.3.4.2 werden jeweils die Fragestellungen F1.1 und F1.2 beantwortet, während in Unterunterabschnitt 8.3.4.3 die Ergebnisse der Experimente zur Beantwortung von Fragestellung F2.1 vorgestellt sind. Abschließend werden die Ergebnisse in Unterunterabschnitt 8.3.4.4 zusammengefasst und diskutiert.

8.3.4.1. Rückschluss auf verantwortliche Modellelemente (F1.1)

Zur Beantwortung der Frage F1.1 sei zunächst betrachtet, welche Modellelemente in UMLsec dafür verantwortlich sind, dass eine Verletzung der semantischen Bedingung

vorliegt. Die semantische Bedingung wird für eine Klasse C, die eine Abhängigkeitsbeziehung zu einer Schnittstelle hat, und eine Klasse D, die eine Realisierungsbeziehung zu der gleichen Schnittstelle besitzt, überprüft. Die Bedingung besagt, dass für jede Nachricht m in der Schnittstelle, m genau dann als Wert in einer der Eigenschaftsdefinitionen des Stereotyps «`Critical`» von C (bzw. D) auftaucht, wenn m auch Wert der gleichen Eigenschaftsdefinition von D (bzw. C) ist. Es ist somit ersichtlich, dass die für eine Sicherheitsverletzung verantwortlichen Elemente die Paare von Klassen sind, von denen eine Klasse eine Abhängigkeit zu einer Schnittstelle I hat und die andere Klasse die Schnittstelle I realisiert. Im Folgenden wird gezeigt, dass aus der Ausgabe der DCP-Analyse die beiden Komponenten ermittelt werden können, die zu den Klassen korrespondieren, die die verantwortlichen Elemente für die Sicherheitsverletzung im UMLsec-Modell sind.

Bei der Abfrage des Prädikats *secureDependencies\4* werden Parameterbelegungen für die Parameter OP, V, R und S genau dann gefunden, wenn von der Analyse eine Sicherheitsverletzung an der Operation OP unter der Aufrufhierarchie S festgestellt wird. Für die Ermittlung der dazugehörigen Komponenten sind die Parameter V und R nicht notwendig. Bei dieser Operation handelt es sich um eine Operation des Operationsmodells, die einen Datenverarbeitungsvorgang des DCP-Modells repräsentiert. Dies ist deshalb der Fall, da eine Sicherheitsverletzung nur an Operationen auftreten kann, wenn an dem zu der Operation gehörenden Datenverarbeitungsvorgang die erwarteten Charakteristiken der Rückgabevariablen annotiert werden, aber von den tatsächlichen zurückgegebenen Daten nicht erfüllt werden. Eine Operation, die Teil der Ausgabe der DCP-Analyse ist, kann also immer zu dem Datenverarbeitungsvorgang zurückverfolgt werden. Dieser Datenverarbeitungsvorgang ist Teil eines Datenverarbeitungscontainers, der an einer Aktion einer SEFF annotiert ist. Daher kann über diesen Weg die Komponente ermittelt werden, die über die SEFF verfügt, die eine Aktion enthält, die mit dem Datenverarbeitungscontainer annotiert ist, der den ermittelten Datenverarbeitungsvorgang enthält. Damit ist eine Komponente bekannt. Nun muss bestimmt werden, ob es sich bei dieser Komponente um die Komponente handelt, die zu der Klasse korrespondiert, welche die Schnittstelle konsumiert, oder zu der Klasse, welche die Schnittstelle bereitstellt.

Ist der Datenverarbeitungscontainer an einem externen Aufruf annotiert, handelt es sich bei der Komponente um eine Komponente, die zu der Klasse korrespondiert, die die Schnittstelle konsumiert. In diesem Fall wird zunächst die Komponenteninstanz bestimmt, zu der die Operation korrespondiert. Dies ist eindeutig möglich, da der eindeutige Bezeichner der Komponenteninstanz bei der Erzeugung des Operationsmodells in den Namen der Operation kodiert wird. Anschließend wird die benötigte Rolle des externen Aufrufs bestimmt, an dem der Datenverarbeitungscontainer annotiert ist, und es wird im Systemmodell überprüft, von welcher Komponenteninstanz diese benötigte Rolle bereitgestellt wird. Dadurch ist die Komponenteninstanz bekannt, welche eine Instanz der Komponente ist, die zu der Klasse korrespondiert, die die Schnittstelle bereitstellt. Damit ist ersichtlich, dass in diesem Fall beide Komponenten, die für die Sicherheitsverletzung verantwortlich sind, ermittelt werden können.

Ist dagegen der Datenverarbeitungscontainer an einer internen Aktion annotiert, handelt es sich bei der Komponente um eine Komponente, die zu der Klasse korrespondiert, die die

```
1 ?- operationProperty('Operation', 'Critical', V).
2 ?- returnValue('S', 'R', 'Critical', V).
```

Algorithmus 8.2.: Die Abfragen zur Ermittlung der Charakteristiken der Operation (Zeile 1) und der Rückgabevariable (Zeile 2) für eine Sicherheitsverletzung der Secure-Dependencies-Analyse.

Schnittstelle bereitstellt. Analog zum ersten Fall wird zunächst die zu der Operation gehörige Komponenteninstanz bestimmt. Da diese Komponenteninstanz die Schnittstelle für mehrere andere Komponenteninstanzen bereitstellen kann, muss hier die Aufrufhierarchie betrachtet werden. Die Aufrufhierarchie ist in dem Parameter *S* gegeben und ist eine Liste, die die aufgerufenen Operationen des Operationsmodells in umgekehrter Reihenfolge enthält. Das erste Element der Liste ist also *OP* und das letzte ist die Operation, die den Beginn der Aufrufhierarchie darstellt. Über diese Liste wird iteriert, bis eine Operation *PDT* mit dem Präfix *PerformDataTransmissionOperation* gefunden wird. Hierbei handelt es sich um eine Operation, die einen Datenverarbeitungsvorgang des Typs *PerformDataTransmission* repräsentiert. Dieser stellt den Aufruf der Methode dar, die von der aktuell betrachteten Komponente bereitgestellt wird. Mit dieser Information wird die Komponenteninstanz der Operation *PDT* ermittelt. Wie bereits beschrieben ist die Bestimmung der zu einer Operation gehörigen Komponenteninstanz eindeutig. Damit ist die Komponenteninstanz gefunden, die den Aufruf an die Komponenteninstanz, welche die Schnittstelle bereitstellt, getätigt hat. Somit ist ersichtlich, dass auch in diesem Fall sowohl die konsumierende Komponente der Schnittstelle, als auch die bereitstellende Komponente der Schnittstelle bestimmt werden können. Damit ist gezeigt, dass für jede entdeckte Sicherheitsverletzung der Secure-Dependencies-Analyse in DCP die verantwortlichen Modellelemente, gemäß der UMLsec-Analyse, ermittelt werden können.

Schließlich ist zu erwähnen, dass eine Ausgabe der Charakteristiken des Typs *Critical* nicht direkt aus den Komponenten ersichtlich ist, wie das in UMLsec für die Eigenschaftswerte des Stereotyps «`Critical`» der Fall ist. Eine Ausgabe der geforderten und tatsächlichen Charakteristiken ist jedoch trivial. Die Abfragen, um die geforderten Charakteristiken und die tatsächlichen Charakteristiken zu ermitteln, sind in Algorithmus 8.2 dargestellt. Die erste Abfrage findet eine Parameterbelegung des Parameters *V* für jeden Wert der Charakteristik *Critical*, der auf der betrachteten Operation gesetzt ist. Die zweite Abfrage findet ebenfalls eine Parameterbelegung des Parameters *V* für jeden Wert der Charakteristik *Critical*, der auf der betrachteten Rückgabevariable *R* unter der Aufrufhierarchie *S* gesetzt ist. Sowohl die Ermittlung der verantwortlichen Komponenten, als auch die Ermittlung der geforderten und tatsächlichen Charakteristiken kann automatisiert erfolgen.

8.3.4.2. Semantik der Analyseformel (F1.2)

Im Folgenden wird zunächst argumentiert, dass die in DCP überprüfte Bedingung äquivalent zu der überprüften Bedingung in UMLsec ist. Anschließend wird gezeigt, dass die

Modellelemente, auf denen die Bedingung überprüft wird, in beiden Analysen die gleiche Semantik aufweisen.

Äquivalenz der überprüften Bedingung

Für die Überprüfung der semantischen Bedingung werden zwei Klassen C und D benötigt, die mit dem Stereotyp «`Critical`» annotiert sind und wovon C eine Abhängigkeit zu einer Schnittstelle I hat, die von D realisiert wird. Für jede Methode und jeden Parameter der Schnittstelle I wird überprüft, dass diese Methode oder dieser Parameter genau dann Teil eines Eigenschaftswerts der Eigenschaftsdefinition `{secrecy}` (resp. `{integrity}`, resp. `{high}`) von C ist, wenn er auch Teil der Eigenschaftswerte der gleichen Eigenschaftsdefinitionen von D ist. Zum besseren Verständnis ist dies nachfolgend als logische Formel dargestellt. Dabei bezeichnet $n \in I$ den Namen einer Methode der Schnittstelle I, die über einen Rückgabewert verfügt, oder den Namen eines Parameters einer Methode der Schnittstelle I. Die Mengen $C_{secrecy}$, $C_{integrity}$ und C_{high} bezeichnen jeweils die Eigenschaftswerte der gleichnamigen Eigenschaftsdefinitionen der Klasse C.

$$\begin{aligned} \forall n \in I: \quad & n \in C_{secrecy} \Leftrightarrow n \in D_{secrecy} \\ \wedge \quad & n \in C_{integrity} \Leftrightarrow n \in D_{integrity} \\ \wedge \quad & n \in C_{high} \Leftrightarrow n \in D_{high} \end{aligned}$$

Da in UMLsec die Analyse als Bedingung definiert ist, die von einem Modell erfüllt werden muss und in DCP die Analyse als Prolog-Prädikat definiert ist, das Parameterbelegungen für Sicherheitsverletzungen findet, ist es sinnvoll, zunächst die Bedingung der UMLsec-Analyse zu negieren, sodass sie analog zur DCP-Analyse die Präsenz einer Sicherheitsverletzung zeigt. Die Negation ergibt folgende Formel.

$$\begin{aligned} \exists n \in I: \quad & n \in C_{secrecy} \nLeftrightarrow n \in D_{secrecy} \\ \vee \quad & n \in C_{integrity} \nLeftrightarrow n \in D_{integrity} \\ \vee \quad & n \in C_{high} \nLeftrightarrow n \in D_{high} \end{aligned}$$

Diese Bedingung ist in dem Prolog-Prädikat *secureDependenciesViolation\4* in der DCP-Analyse umgesetzt. Zur besseren Verständlichkeit ist dieses Prädikat, welches bereits in Kapitel 6 definiert ist, erneut in Algorithmus 8.3 dargestellt. Es ist zu erkennen, dass für jeden Wert V des Charakteristiktypen *CriticalLiteral* überprüft wird, ob die Charakteristik V in der Menge der erwarteten Charakteristiken enthalten ist (Zeile 3), aber nicht in der Menge der tatsächlichen Charakteristiken des Datums (Zeile 4). Außerdem wird überprüft, ob die Charakteristik V nicht in der Menge der erwarteten Charakteristiken enthalten ist (Zeile 8), aber dafür in der Menge der tatsächlichen Charakteristiken des Datums (Zeile 9). Als logische Formel kann dies folgendermaßen dargestellt werden. Dabei beschreibt *Expected* die Menge von erwarteten Charakteristiken und *Actual* die Menge

der tatsächlichen Charakteristiken eines Datums.

$$
\begin{aligned}
& (Secrecy \in Expected \land Secrecy \notin Actual) \\
\lor\ & (Secrecy \notin Expected \land Secrecy \in Actual) \\
\lor\ & (Integrity \in Expected \land Integrity \notin Actual) \\
\lor\ & (Integrity \notin Expected \land Integrity \in Actual) \\
\lor\ & (High \in Expected \land High \notin Actual) \\
\lor\ & (High \notin Expected \land High \in Actual)
\end{aligned}
$$

Dies kann offensichtlich zu folgender Formel umformuliert werden.

$$
\begin{aligned}
& Secrecy \in Expected \nleftrightarrow Secrecy \in Actual \\
\lor\ & Integrity \in Expected \nleftrightarrow Integrity \in Actual \\
\lor\ & High \in Expected \nleftrightarrow High \in Actual
\end{aligned}
$$

Es ist zu erkennen, dass die Bedingungen essentiell dieselben Eigenschaften prüfen. Ein Unterschied besteht jedoch darin, dass in UMLsec die Sicherheitseigenschaften als Eigenschaftsdefinition gegeben sind und die Methoden oder Parameter, für die diese Sicherheitseigenschaften gefordert oder garantiert sind, als Werte der entsprechenden Eigenschaftsdefinitionen modelliert sind. In DCP dagegen existiert für den Aufruf einer Methode einer Schnittstelle ein Datenverarbeitungscontainer. Über diesen Datenverarbeitungscontainer ist die aufgerufene Methode eindeutig definiert. Dadurch können in der DCP-Analyse die geforderten Sicherheitseigenschaften direkt an den entsprechenden Datenverarbeitungsvorgang des Datenverarbeitungscontainers annotiert werden. Die tatsächlichen Sicherheitseigenschaften werden ebenfalls direkt an den Daten annotiert. Hierbei handelt es sich jedoch lediglich um einen Unterschied in der Modellierung. Im Folgenden wird argumentiert, dass die Modellelemente, auf denen die Überprüfung durchgeführt wird, im Kontext der Secure-Dependencies-Analyse dieselbe Semantik aufweisen.

Semantik der für die Analyse relevanten Modellelemente

Wie bereits erklärt operiert die DCP-Analyse auf Datenflüssen und vergleicht die Charakteristiken eines Datenverarbeitungsvorgangs mit den Charakteristiken eines Datums, das von diesem Datenverarbeitungsvorgang bearbeitet wird. Die UMLsec-Analyse definiert die Sicherheitseigenschaften anhand von Eigenschaftsdefinitionen, deren Werte die Namen der Methoden und Parameter enthalten, für die diese Sicherheitseigenschaften gefordert oder garantiert sind. Die Semantik der Eigenschaftsdefinitionen des Stereotyps «`Critical`» ist bereits in Kapitel 6 erklärt und wird im Folgenden nur kurz zusammengefasst. Für eine Klasse C, die eine Schnittstelle I konsumiert und eine Methode $m() : Data$ von I als Wert einer der Eigenschaftsdefinitionen `{secrecy}`, `{integrity}` oder `{high}` hat, ist die Semantik, dass diese Sicherheitseigenschaften für die Rückgabevariable der Methode

```
1 secureDependenciesViolation(OP, S, R, V) :-
2     valueSetMember('CriticalLiteral', V),
3     operationProperty(OP, 'Critical', V),
4     lnot(returnValue(S, R, 'Critical', V)).
5
6 secureDependenciesViolation(OP, S, R, V) :-
7     valueSetMember('CriticalLiteral', V),
8     lnot(operationProperty(OP, 'Critical', V)),
9     returnValue(S, R, 'Critical', V).
```

Algorithmus 8.3.: Das Prädikat *secureDependenciesViolation\4* der Secure-Dependencies-Analyse.

m() gefordert sind. Für eine Klasse *D*, die eine Schnittstelle *I* bereitstellt und dieselbe Methode *m*() : *Data* von *I* als Wert einer der Eigenschaftsdefinitionen hat, ist die Semantik, dass diese Sicherheitseigenschaft für die Rückgabevariable der Methode garantiert ist. Es gilt also, dass die Klasse *C* die Erwartungen bezüglich der Sicherheitseigenschaften des Datums spezifiziert und die Klasse *D* die tatsächlichen Sicherheitseigenschaften des Datums angibt. Analog gilt dies für Parameter, wobei hier die konsumierende Klasse die tatsächlichen Sicherheitseigenschaften des Parameters angibt und die bereitstellende Klasse die Sicherheitsanforderungen an den Parameter stellt.

In DCP werden die erwarteten Sicherheitseigenschaften immer an Datenverarbeitungsvorgängen definiert und die tatsächlichen Sicherheitseigenschaften an den Daten selbst. Wenn Sicherheitseigenschaften der Rückgabevariable einer Methode *m*() gefordert sind, wird gemäß der definierten Modelltransformation diese Sicherheitsanforderung an dem Datenverarbeitungsvorgang des Typs *PerformDataTransmission* des Datenverarbeitungscontainers des externen Aufrufs, der die Methode *m*() als Ziel hat, annotiert. Dieser Datenverarbeitungsvorgang gibt die Rückgabevariable der Methode *m*() zurück. Da in der DCP-Analyse die Secure-Dependencies-Bedingung an den Rückgabevariablen der Datenverarbeitungsvorgänge überprüft wird, entspricht dies der Semantik einer Methode *m*(), die Teil eines Eigenschaftswertes einer konsumierenden Klasse ist. Die tatsächlichen Sicherheitseigenschaften der Rückgabevariable werden in einem Datenverarbeitungsvorgang der aufgerufenen SEFF zugewiesen, indem ein Datenverarbeitungsvorgang des Typs *CreateData* ein neues Datum mit den entsprechenden Sicherheitseigenschaften erstellt. Diese Daten werden zurückgegeben an den aufrufenden Datenverarbeitungsvorgang des Typs *PerformDataTransmission*. Daher entsprechen die Charakteristiken, die der Datenverarbeitungsvorgang des Typs *CreateData* dem erstellten Datum zuweist, der Semantik einer Methode *m*(), die Teil eines Eigenschaftswertes einer bereitstellenden Klasse ist. Analog gilt dies für Parameter, wobei hier die aufrufende SEFF über einen Datenverarbeitungsvorgang des Typs *CreateData* verfügt, der die tatsächlichen Sicherheitseigenschaften des Parameters spezifiziert. Ein Datenverarbeitungsvorgang des Typs *SelectData* in dem Datenverarbeitungscontainer der aufgerufenen SEFF spezifiziert die Sicherheitsanforderungen.

Es ist zu sehen, dass die gewählte Modellierung in DCP bezüglich der Semantik der Modellierung der Sicherheitseigenschaften in UMLsec entspricht. Allgemein ist für die Secure-Dependencies-Analyse zu erkennen, dass die Abbildung der Semantik der Sicherheitseigenschaften wenigen Einschränkungen unterliegt. Dies ist deshalb der Fall, da die Sicherheitseigenschaften nicht direkt als Stereotypen an semantisch für die Analyse relevanten Modellelementen, wie z. B. Kommunikationspfaden in der Secure-Links-Analyse, angewendet werden, sondern als Eigenschaftsdefinitionen des Stereotyps «`Critical`», welcher an Klassen annotiert wird. Da Klassen im Kontext dieser Transformation auf Komponenten abgebildet werden, ist es also notwendig, dass die Charakteristiken, die zu den Sicherheitseigenschaften korrespondieren, an DCP-Modellelementen angebracht werden, die eindeutig einer Komponente zuzuordnen sind. Semantisch ist es hier sinnvoll, den Datenverarbeitungsvorgang zu wählen, der den Methodenaufruf repräsentiert und die Datenverarbeitungsvorgänge des Datenverarbeitungscontainers der aufgerufenen SEFF, die die Parameter verwenden. Es ist bereits in Unterunterabschnitt 8.3.4.1 gezeigt, dass ausgehend von den Operationen dieser Datenverarbeitungsvorgänge eindeutig die zugehörigen Komponenten ermittelt werden können. Weiter ist es notwendig, die aus der Analysedefinition extrahierte Semantik zu erhalten. Dass dies gegeben ist, ist in den vorherigen Paragraphen gezeigt. Insgesamt ist also zu sehen, dass die gewählte Art der Modellierung die Semantik der Sicherheitseigenschaften der Secure-Dependencies-Analyse erhält und dadurch auch die Bedingung, die von der DCP-Analyse überprüft wird, dieselbe Semantik aufweist, wie die semantische Bedingung der UMLsec-Analyse.

8.3.4.3. Ergebnisse der Experimente (F2.1)

Im Folgenden werden die Ergebnisse der durchgeführten Experimente zur Beantwortung von Fragestellung F2.1 vorgestellt. Alle DCP-Modelle, die aus der Transformation der UMLsec-Eingabemodelle während der Evaluation entstehen, sind Teil eines Datensatzes [44]. Die Abbildung von den UMLsec-Modellen des Datensatzes auf entsprechende DCP-Modelle ist in Tabelle A.5 dargestellt. Die Modelle, die von 1 bis 64 nummeriert sind, verfügen über eine Methode mit Rückgabevariable und ohne Parameter. Die Modelle mit der Nummerierung von 65 bis 128 verfügen über eine Methode ohne Rückgabevariable und mit Parameter. Es existieren insgesamt 16 Fälle, in denen keine Sicherheitsverletzung vorliegt. Es handelt sich dabei um die Fälle mit den Nummern 1, 10, 19, 28, 37, 46, 55, 64, 65, 74, 83, 92, 101, 110, 119 und 128. Alle anderen Fälle verfügen über eine Sicherheitsverletzung.

Für alle 112 Modelle mit einer Sicherheitsverletzung erkennt die DCP-Analyse ebenfalls eine Sicherheitsverletzung. Die Sensitivität beträgt also 100%. Ebenso ist die DCP-Analyse für alle 16 Modelle, die über keine Sicherheitsverletzung verfügen, korrekt. Die Analyse erkennt in diesen Fällen also ebenfalls keine Sicherheitsverletzung. Die Spezifität liegt daher ebenfalls bei 100%.

8.3.4.4. Zusammenfassung und Diskussion

In Unterunterabschnitt 8.3.4.1 und Unterunterabschnitt 8.3.4.2 wurde gezeigt, dass die in Prolog umgesetzte Analyse dieselbe Semantik aufweist wie die UMLsec-Analyse. Dies entspricht dem definierten Evaluationsziel Z1. Anschließend wurden in Unterunterabschnitt 8.3.4.3 die Ergebnisse der Experimente präsentiert, welche für die Beantwortung der Frage F2.1 verwendet werden. Für jedes Eingabemodell kann die DCP-Analyse korrekt feststellen, ob es über Sicherheitsverletzungen verfügt. Sensitivität und Spezifität betragen also jeweils 100%. Dabei ist zu erwähnen, dass nur auf die Präsenz von Sicherheitsverletzungen geprüft wurde. Dies ist aussagekräftig, da jedes Eingabemodell über entweder keine oder genau eine Sicherheitsverletzung verfügt, wie nachfolgend skizziert ist. Jedes Modell verfügt über genau eine Schnittstelle mit genau einer Methode. Es gilt, dass in den Modellen, in denen die Methode einen Rückgabewert besitzt, sie über keinen Parameter verfügt und in den Modellen, in denen die Methode einen Parameter besitzt, sie über keinen Rückgabewert verfügt. Daher kann in jedem Modell nur genau eine Sicherheitsverletzung auftreten. Je nach Modell kann also eine Sicherheitsverletzung nur an der Rückgabevariable der Methode oder dem Parameter der Methode vorliegen. Dabei ist zu erwähnen, dass die DCP-Analyse für manche Sicherheitsverletzungen mehrere Parameterbelegungen zurückgibt. Dies passiert deshalb, da jede Sicherheitseigenschaft $V \in \{Secrecy, Integrity, High\}$ getrennt geprüft wird und bei einer Sicherheitsverletzung eine Parameterbelegung mit dieser Sicherheitseigenschaft ausgegeben wird. Ist z. B. ein Datum mit der Sicherheitseigenschaft *Secrecy* erwartet, aber es wird ein Datum mit der Sicherheitseigenschaft *Integrity* empfangen, wird von der DCP-Analyse jeweils eine Parameterbelegung mit $V = Integrity$ und eine Parameterbelegung mit $V = Secrecy$ ausgegeben. Es ist jedoch trivial zu erkennen, dass es sich semantisch um dieselbe Sicherheitsverletzung handelt, da die Parameter OP, S und R identisch sind und sich die Prolog-Ausgabe nur in dem Parameter V unterscheidet.

8.4. Bewertung der Validität

Die Bewertung der Validität ist anhand der Richtlinien für Fallstudienforschung [46] strukturiert. Auch wenn es sich bei dieser Evaluation nicht um die Betrachtung von Fallstudien handelt, sind die Kategorien auch hier anwendbar und erleichtern die Zuordnung und das Verständnis.

Interne Validität Zunächst ist die interne Validität betrachtet. Diese stellt sicher, dass die erwarteten Einflussfaktoren auf die Evaluation die einzigen Einflussfaktoren sind. In dieser Evaluation wird die Genauigkeit der erstellten Sicherheitsanalysen geprüft. Es ist zu erwarten, dass die Spezifikation der Modelltransformationen und die Umsetzung der Analyse Einflussfaktoren sind. Ein weiterer möglicher Einflussfaktor kann die Auswahl der Eingabemodelle für die Evaluation sein. Um dies auszuschließen, wurden für die Secure-Links-Analyse alle möglichen Kombinationen von Verteilungen der Secure-Links-

Stereotypen in einem Modell fester Struktur erzeugt und als Eingabe für die Evaluation verwendet. Analog wurde für die Secure-Dependencies-Analyse vorgegangen. Daher ist zu erwarten, dass die Auswahl der Modelle keinen Einflussfaktor darstellt. Schließlich sind die Auswertung und insbesondere die Zusammenfassung der duplizierten DCP-Sicherheitsprobleme zu erwähnen. Das Vorgehen für die Auswertung und das Zusammenfassen der Sicherheitsprobleme ist vollständig beschrieben und logisch hergeleitet, sodass diese ebenfalls keine Einflussfaktoren darstellen.

Externe Validität Die externe Validität beschreibt, inwieweit die Erkenntnisse generalisiert werden können. Für Secure Links wurden in der Evaluation vier Strukturen identifiziert, aus deren Komposition Eingabemodelle beliebiger Struktur erzeugt werden können. Da die Analyse in diesen vier Fällen erfolgreich war, kann davon ausgegangen werden, dass sie auch in Kompositionen dieser Fälle erfolgreich ist. Eine Gefahr für die Validität der Evaluation ist, dass nicht beachtete strukturelle Elemente, wie z. B. Verteilungsbeziehungen, auch einen Einfluss auf die Analyseergebnisse haben können. Um dies auszuschließen wurde argumentativ gezeigt, dass ein Einfluss anderer Modellelementen nicht zu erwarten ist. Für die Secure-Dependencies-Analyse konnte aufgrund einer fehlenden, aktuellen Referenzimplementation keine Evaluation der strukturellen Eigenschaften der Modelltransformation durchgeführt werden. Im Gegensatz zur Secure-Links-Analyse ist die Secure-Dependencies-Analyse allerdings strukturell wesentlich weniger komplex, sodass nicht zu erwarten ist, dass strukturelle Fälle existieren, die Einfluss auf das Analyseergebnis haben.

Konstruktvalidität Bei der Konstruktvalidität wird sichergestellt, dass die verwendeten Metriken angemessen für die Beantwortung der Fragestellungen sind. Für die Fragestellungen zu dem Evaluationsziel Z1 wurde die Evaluation argumentativ durchgeführt. Auch wenn im Allgemeinen es nicht möglich ist die Äquivalenz von Programmen zu zeigen, kann in diesem Fall eine argumentative Aussage über die Äquivalenz der UMLsec- und DCP-Analyse getroffen werden, da nur ein geringer Unterschied in der Abstraktionsebene zwischen den definierten UMLsec-Bedingungen der Analyse und den erstellten DCP-Prolog-Regeln der Analyse vorliegen. Für die Fragestellungen zu dem Evaluationsziel Z2 sind Messungen verschiedener Metriken erfolgt. Es liegen für die Evaluation der Secure-Links-Analyse Referenzwerte in Form des CARiSMA-Werkzeugs vor, wodurch eine detaillierte Betrachtung der Analyseergebnisse möglich ist. Dadurch kann für jeden Fall, in dem die Ergebnisse von den Erwartungen abweichen, ermittelt werden, wodurch diese Abweichung aufgetreten ist und ob es sich dabei um ein generelles Problem der erstellten Analyse handelt. Dadurch ist eine Unterteilung in z. B. richtig-positiv-Rate und falsch-negativ-Rate nicht sinnvoll, da diese nicht mehr Informationen liefern, als die Betrachtung der einfachen Fehlerrate und der Fälle, in denen die Ergebnisse von den Erwartungen abweichen. Für die Secure-Dependencies-Analyse liegen keine Referenzwerte vor, daher wurden Szenarien erzeugt, die entweder über eine oder keine Sicherheitsverletzung verfügen. Dadurch kann das Ergebnis der Analyse auf eine binäre Entscheidung reduziert werden. Die Metriken Sensitivität und Spezifität erlauben eine getrennte Betrachtung der Szenarien, in denen

eine Sicherheitsverletzung vorliegt, und der Szenarien, in denen keine vorliegt, und sind daher für das Vorgehen der Secure-Dependencies-Evaluation geeignet.

Zuverlässigkeit Schließlich ist die Zuverlässigkeit der Evaluation zu betrachten. Dies beschreibt, dass die Ergebnisse nicht abhängig von dem Forscher sind, von dem sie erzeugt wurden. In diesem Fall ist die Modellerstellung abhängig von dem erstellenden Forscher, da hier Wissen bezüglich des Umgangs mit der Modellierungssprache und über den Ablauf der betrachteten Sicherheitsanalyse notwendig ist. Um dies zu mitigieren, sind alle erstellten Modelle in einem Datensatz [44] verfügbar. Ferner sind als Teil dieses Datensatz der benötigte Code zur Erzeugung des Operationsmodells und des Prolog-Programms gegeben. Zusätzlich ist eine Anleitung enthalten, die beschreibt, wie die Ergebnisse erzeugt werden. Zusammen mit der vollständigen Beschreibung des systematischen Vorgehens zur Erzeugung der Evaluationsergebnisse dieser Arbeit, ist zu erwarten, dass damit die Evaluationsergebnisse nachvollzogen werden können.

8.5. Einschränkungen und Annahmen

In diesem Kapitel werden die Einschränkungen und Annahmen des in dieser Arbeit vorgenommenen Vergleichs der Ausdrucksmächtigkeit von DCP und UMLsec vorgestellt. Eine zentrale Einschränkung ist, dass im Kontext dieser Arbeit die betrachteten Sicherheitsanalysen nur separat analysiert und auf ihre Ausdrucksmächtigkeit untersucht sind. Es gibt jedoch Szenarien, in denen Aspekte mehrere UMLsec-Analysen gemeinsam verwendet werden. Ein im UMLsec-Buch [1, Seiten 88–118] vorgestelltes Szenario ist das Aufladen einer Geldkarte der Common Electronic Purse Specification (CEPS). In dieser werden sowohl die Konzepte der Secure-Dependency-Analyse, als auch der Secure-Links-Analyse verwendet. Die Untersuchung der Komposition der definierten Analysen ist nicht Teil dieser Arbeit, daher kann keine Aussage darüber getroffen werden, ob die definierten Analysen auch in Komposition funktionieren und über dieselbe Ausdrucksmächtigkeit verfügen.

Für die Secure-Links-Analyse sind zwei Annahmen getroffen worden. Zunächst existiert in UMLsec eine Angreiferaktion `access`, die beschreibt, ob ein Angreifer Zugang zu Rechnerknoten eines bestimmten Typs hat. Laut der Definition der UML-Maschine, wird diese Angreiferaktion als einer der ersten Schritte der allgemeinen Analyse umgewandelt. Dabei erhält der Angreifer für jeden Kommunikationspfad, der an einem Rechnerknoten anliegt, für den der Angreifer die `access`-Aktion unternehmen darf, die erlaubten Aktionen `read`, `insert`, `delete`. Hat ein Angreifer also Zugang zu einem Rechnerknoten, hat er dadurch auch auf allen physischen Verbindungen des Rechnerknotens alle verfügbaren Rechte. Das CARiSMA-Werkzeug berücksichtigt bei der Secure-Links-Analyse jedoch nicht die `access`-Aktionen, die der Angreifer unternehmen darf. Es existiert keine Kombinationen von Rechnerknotentypen und Angreifer, sodass das Ergebnis der Analyse durch das `access`-Recht beeinflusst ist. Damit entspricht das Verhalten des CARiSMA-Werkzeugs nicht dem definierten Verhalten [1, Seite 39 und 59] der Analyse. Es ist unklar, ob es sich dabei

um eine bewusste Entscheidung oder um eine Einschränkung der Implementation des CARiSMA-Werkzeugs handelt. Für diese Arbeit ist die Annahme getroffen worden, dass die `access`-Rechte eines Angreifers keinen Einfluss auf die Secure-Links-Analyse haben. Diese Annahme ist insbesondere im Kontext der Evaluation sinnvoll, da der Vergleich der Ergebnisse ansonsten erheblich erschwert wird, wenn dieses beschriebene Verhalten in der DCP-Analyse umgesetzt ist, aber nicht im CARiSMA-Werkzeug.

Die zweite Annahme bezieht sich auf die Struktur der für die Secure-Links-Analyse gültigen DCP-Eingabemodelle. Die Modelltransformation und die Umsetzung der Analyse nehmen an, dass zwischen zwei Rechnerknoten immer nur genau eine Netzwerkressource existiert. Dabei handelt es sich auch um ein gültiges PCM-Modell, wenn mehrere Netzwerkressourcen zwischen zwei Rechnerknoten existieren. Dieser Fall ist deshalb problematisch, weil die Semantik für den Datenfluss unklar ist. Es ist nicht eindeutig, ob Daten die zwischen solchen Rechnerknoten transferiert werden, immer über beide Netzwerkressourcen fließen, nur eine verwenden oder vielleicht abhängig von anderen Faktoren entscheiden, welche Netzwerkressource verwendet wird. Da dieser Fall auch von der UMLsec-Analyse nicht dargestellt werden kann, leidet die relative Ausdrucksmächtigkeit von DCP in Bezug zu UMLsec nicht unter dieser Annahme.

Für die Secure-Dependencies-Analyse sind ebenfalls zwei Annahmen getroffen worden. Die erste ist bereits in Kapitel 6 beschrieben. Dabei geht es um die unklare Semantik der annotierten Sicherheitseigenschaften einer Klasse, wenn eine Klasse sowohl eine Abhängigkeitsbeziehung, als auch eine Realisierungsbeziehung zu derselben Schnittstelle hat. Die zweite Annahme ist, dass in den UMLsec-Modellen die Abhängigkeiten stets über Schnittstellen realisiert sind. In der Definition der Analyse [1, Seite 59f] existiert eine kurz beschriebene Erweiterung, durch die auch direkte Abhängigkeiten zwischen Klassen gültige Strukturen für die Eingabemodelle an die Secure-Dependencies-Analyse sind. Da in DCP jegliche Kommunikation zwischen Komponenten über Schnittstellen definiert ist, ist dies so nicht in DCP darstellbar. Es ist grundsätzlich möglich, solche Modelle trotzdem nach DCP abzubilden, indem eine neue Schnittstelle in DCP erzeugt wird, die die Abhängigkeit zwischen diesen Klassen repräsentiert. Da dies aber die Semantik des Eingabemodells erheblich verändert, wurde für diese Arbeit die Entscheidung getroffen, auf diese Transformation zu verzichten.

9. Zusammenfassung und Ausblick

In dieser Arbeit wurden die Ansätze UMLsec und Data-Centric Palladio (DCP) zur Sicherheitsanalyse von Softwarearchitekturen bezüglich ihrer Ausdrucksmächtigkeit miteinander verglichen. Die grundlegende Idee dieses Vergleichs ist es, verschiedene bereits in UMLsec definierte Analysen darauf zu untersuchen, ob eine äquivalente Analyse in DCP definiert werden kann und, falls dies der Fall ist, die Analyse in DCP umzusetzen. Aus den Erkenntnissen dieser Untersuchungen und den zwei umgesetzten strukturellen Analysen wurde anschließend eine Aussage über die Ausdrucksmächtigkeit der beiden Ansätze getroffen. Bei den betrachteten UMLsec-Analysen handelt es sich um Secure Links, Secure Dependencies, Fair Exchange und sicherer Informationsfluss (no up-flow / no down-flow). Für Secure Links und Secure Dependencies, die sich für eine Umsetzung eignen, wurde die Semantik der UMLsec-Analyse aus der Definition [1, Kapitel3 und 4] und der Referenzimplementation [41] extrahiert. Auf dieser Basis wurde eine äquivalente Analyse in DCP spezifiziert und Modelltransformationen zwischen den Eingabemodellen der UMLsec-Analyse und den Eingabemodellen der DCP-Analyse definiert, sodass eine Evaluation der Genauigkeit der erstellten Analyse durchgeführt werden konnte.

Für die Secure-Links-Analyse und die Secure-Dependencies-Analyse konnte eine äquivalente Analyse in DCP definiert werden. Dabei wurde gezeigt, dass UMLsec-Analysen, die eher strukturelle Eigenschaften von Modellen überprüfen, sich gut für eine Umsetzung in DCP eignen, da der Abstraktionsunterschied zwischen den strukturbeschreibenden Modellelementen aus UML und dem Palladio-Komponentenmodell (PCM) gering ist. Daher ist eine Abbildung zwischen den Eingabemodellen von UMLsec und DCP, die die Semantik des ursprünglichen Modells erhält, in diesen Fällen möglich. Die Semantik der in UMLsec annotierten Sicherheitseigenschaften eignet sich bei dieser Art von Modellen auch gut für eine Umsetzung in DCP, da in diesen Fällen Vertraulichkeit und Integrität von Datenklassen beschrieben werden. In dem datenflussbasierten Ansatz von DCP ist die Annotation der Vertraulichkeits- bzw. Integritätseigenschaften direkt an den Daten möglich. Eine Analyse erfolgt in DCP dann über die Analyse des Datenflusses und der um Sicherheitseigenschaften annotierten Daten. Dies hat zu dem Vorteil geführt, dass es in DCP möglich ist, wesentlich detaillierte Analyseergebnisse zu erhalten, während in der Secure-Links- und Secure-Dependencies-Analyse jeweils Überabschätzungen bezüglich der Sicherheitseigenschaften getroffen werden müssen. In diesen konkreten Szenarien weist DCP im Vergleich zu UMLsec eine höhere Ausdrucksmächtigkeit auf.

Für die Fair-Exchange-Analyse, welche eine Analyse des Verhaltens eines Systems anhand eines UML-Aktivitätsdiagramms darstellt, wurde gezeigt, dass eine Umsetzung in DCP so nicht möglich ist. Der Kern der DCP-Analyse beschäftigt sich mit der Analyse von Daten

und mit der Frage, über welche sicherheitsrelevanten Eigenschaften diese an verschiedenen Stellen des Datenflusses verfügen. Anhand des Beispiels der Fair-Exchange-Analyse wurde gezeigt, dass der Analyseansatz von DCP nicht in der Lage ist, kontrollflussbasierte Analysen, die über einen interaktiven Angreifer verfügen, abzubilden. Ferner existiert mit der UMLsec-Analyse des sicheren Informationsflusses eine Analyse, die bezüglich der Semantik in DCP umsetzbar ist, aber für die aufgrund des hohen Abstraktionsunterschieds der UMLsec-Modelle dieser Analyse zu den DCP-Modellen keine allgemeingültige Abbildung möglich ist. An diesen Szenarien ist zu erkennen, dass Analysen in UMLsec existieren, die in DCP nicht oder nur eingeschränkt darstellbar sind.

Für den Vergleich der Ausdrucksmächtigkeit der beiden Ansätze konnte also gezeigt werden, dass weder DCP, noch UMLsec strikt mächtiger ist, als der jeweils andere Ansatz. Stattdessen ist der erwartete Anwendungsfall von erheblicher Bedeutung für die Wahl eines geeigneten Analyseansatzes.

Während die hier betrachteten Analysen bereits Erkenntnisse bezüglich des Vergleichs der Ausdrucksmächtigkeit dieser beiden Ansätze liefern, konnten im Laufe der Arbeit verschiedene Aspekte identifiziert werden, die sich für eine weiterführende Arbeit eignen. Zunächst ist die Betrachtung von weiteren in dieser Arbeit nicht untersuchten UMLsec-Analysen ein sinnvoller Einstieg. Ähnlich wie bei der Analyse des sicheren Informationsflusses ist zu erwarten, dass ein Großteil der UMLsec-Analysen bezüglich ihrer Semantik in DCP umgesetzt werden können, solange die Semantik der Analyse auf der höheren Abstraktionsebene von DCP darstellbar ist.

Zusätzlich ist die Komposition von UMLsec-Sicherheitsanalysen ein Thema, das nicht im Rahmen dieser Arbeit betrachtet wurde. Es ist möglich, auf ein UML-Subsystem eine Menge von UMLsec-Analysen anzuwenden. Dies ist für die äquivalenten DCP-Analysen in der aktuellen Form nicht möglich, da die Transformation der UMLsec-Eingabemodelle zu DCP-Modellen analyseabhängig ist. Wenn eine analyseunabhängige Transformation definiert werden kann, die auch für Subsysteme, auf die mehrere UMLsec-Analysen angewandt werden, ein gültiges und semantisch äquivalentes DCP-Eingabemodell generiert, könnten auch wesentlich komplexere Szenarien in DCP abgebildet werden. Ferner ist hierfür sicherzustellen, dass die definierten DCP-Analysen sich nicht gegenseitig beeinflussen, wenn sie gemeinsam angewandt werden.

Abschließend ist der Vergleich von DCP mit anderen Sicherheitsanalyseansätzen als weiteres Forschungsthema zu erwähnen. Da DCP nicht nur eine Analyse definiert, sondern auch einen Entwicklungsprozess, wäre hier insbesondere interessant, ebenfalls solche Ansätze zum Vergleich heranzuziehen. Ein möglicher Ansatz, für dessen Vergleich direkt an den Ergebnissen dieser Arbeit angesetzt werden kann, vereint Secure Tropos und UMLsec [32]. Dieser Ansatz hat das Ziel, einen Entwicklungsprozess zu definieren, der sowohl die Modellierung der sozialen, als auch technologischen Dimension von Sicherheit erlaubt.

Literatur

[1] Jan Jürjens. *Secure Systems Development with UML*. Berlin: Springer, 2005.

[2] Anna-Magdalena Seufert und Nico Vitt. „Medien zur DSGVO: Die Berichterstattung vor und seit dem Stichtag im Vergleich“. In: *Wirtschaftsinformatik & Management* 11.1 (2019), S. 22–31.

[3] John Fontana. *LinkedIn will pay $1.25 million to settle suit over password breach*. 2015. URL: `https://zd.net/2rnvggE` (besucht am 16. 03. 2019).

[4] J. Isaak und M. J. Hanna. „User Data Privacy: Facebook, Cambridge Analytica, and Privacy Protection“. In: *Computer* 51.08 (2018), S. 56–59.

[5] Herb Weisbaum. *Trust in Facebook has dropped by 66 percent since the Cambridge Analytica scandal*. 2018. URL: `https://www.nbcnews.com/business/consumer/trust-facebook-has-dropped-51-percent-cambridge-analytica-scandal-n867011` (besucht am 28. 03. 2019).

[6] Stephan Seifermann. „Architectural Data Flow Analysis“. In: *13th Working IEEE/IFIP Conference on Software Architecture, WICSA 2016, Venice, Italy, April 5-8, 2016*. IEEE Computer Society, 2016, S. 270–271.

[7] Gary McGraw. *Software Security: Building Security In*. Upper Saddle River, NJ [u.a.]: Addison-Wesley, 2006.

[8] Premkumar T. Devanbu und Stuart G. Stubblebine. „Software Engineering for Security: A Roadmap“. In: *22nd International Conference on on Software Engineering, Future of Software Engineering Track, ICSE 2000, Limerick Ireland, June 4-11, 2000*. Hrsg. von Anthony Finkelstein. ACM, 2000, S. 227–239.

[9] Charles A. R. Hoare. „How Did Software Get So Reliable Without Proof?“ In: *FME '96: Industrial Benefit and Advances in Formal Methods, Third International Symposium of Formal Methods Europe, Co-Sponsored by IFIP WG 14.3, Oxford, UK, March 18-22, 1996, Proceedings*. Hrsg. von Marie-Claude Gaudel und Jim Woodcock. Bd. 1051. Lecture Notes in Computer Science. Springer, 1996, S. 1–17.

[10] Constance L. Heitmeyer. „Formal Methods for Developing Software Specifications: Paths to Wider Usage“. In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 1999, June 28 - Junlly 1, 1999, Las Vegas, Nevada, USA*. Hrsg. von Hamid R. Arabnia. CSREA Press, 1999, S. 1047–1053.

[11] Stephan Seifermann, Kateryna Yurchenko und Max E. Kramer. „Challenges to Trading-Off Performance and Privacy of Component-Based Systems“. In: *Softwaretechnik-Trends* 36.4 (2016).

[12] Stephan Seifermann, Robert Heinrich und Ralf Reussner. „Data-Driven Software Architecture for Analyzing Confidentiality“. In: *2019 IEEE International Conference on Software Architecture (ICSA2019)*. Accepted, to appear. IEEE, 2019.

[13] Ralf Reussner u. a., Hrsg. *Modeling and Simulating Software Architectures. The Palladio Approach*. Cambridge, Massachusetts: MIT Press, 2016.

[14] Bastian Best, Jan Jurjens und Bashar Nuseibeh. „Model-Based Security Engineering of Distributed Information Systems Using UMLsec“. In: *29th International Conference on Software Engineering (ICSE'07)*. IEEE. 2007, S. 581–590.

[15] Axelle Apvrille und Makan Pourzandi. „Secure Software Development by Example“. In: *IEEE Security & Privacy* 3.4 (2005), S. 10–17.

[16] Jan Jürjens. „Model-Based Security Testing Using UMLsec: A Case Study“. In: *Electronic Notes in Theoretical Computer Science* 220.1 (2008), S. 93–104.

[17] Holger Schmidt und Jan Jürjens. „Connecting Security Requirements Analysis and Secure Design Using Patterns and UMLsec“. In: *Advanced Information Systems Engineering*. Hrsg. von Haralambos Mouratidis und Colette Rolland. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 367–382.

[18] Ian Sommerville. *Software Engineering*. 9., aktualisierte Auflage. München: Pearson, 2012.

[19] Object Management Group (OMG). *Unified Modeling Language, Version 2.5.1*. OMG Document Number formal/17-12-05 (`https://www.omg.org/spec/UML/2.5.1`). 2017.

[20] Jack B. Dennis. „First Version of a Data Flow Procedure Language“. In: *Programming Symposium, Proceedings Colloque Sur La Programmation*. Berlin, Heidelberg: Springer-Verlag, 1974, S. 362–376.

[21] Krishna M. Kavi, Bill P. Buckles und U. Narayan Bhat. „A Formal Definition of Data Flow Graph Models“. In: *IEEE Transactions on Compututers* 35.11 (1986), S. 940–948.

[22] Paul T. Ward. „The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing“. In: *IEEE Transactions on Software Engineering* 12.2 (1986), S. 198–210.

[23] Edward Yourdon und Larry L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1979.

[24] Jonas Kunz. *Efficient Data Flow Constraint Analysis*. Masterarbeit. Karlsruher Institut für Technologie, 2018.

[25] SWI Prolog. *Reference Manual*. URL: `http://www.swi-prolog.org/pldoc/doc_for?object=manual` (besucht am 01. 03. 2019).

[26] Wolfgang Ahrendt u. a. *Deductive Software Verification – The KeY Book*. Cham: Springer, 2016.

[27] Hamid Bagheri u. a. „Practical, Formal Synthesis and Automatic Enforcement of Security Policies for Android“. In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2016, S. 514–525.

[28] Vibhu S. Sharma und Kishor S. Trivedi. „Architecture Based Analysis of Performance, Reliability and Security of Software Systems“. In: *Proceedings of the 5th International Workshop on Software and Performance*. New York, NY, USA: ACM, 2005, S. 217–227.

[29] Mohamed Almorsy, John Grundy und Amani S. Ibrahim. „Automated Software Architecture Security Risk Analysis Using Formalized Signatures“. In: *Proceedings of the 2013 International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2013, S. 662–671.

[30] Bernhard Hoisl, Stefan Sobernig und Mark Strembeck. „Modeling and Enforcing Secure Object Flows in Process-Driven SOAs: An Integrated Model-Driven Approach“. In: *Software & Systems Modeling* 13.2 (2014), S. 513–548.

[31] Kuzman Katkalov u. a. „Model-Driven Development of Information Flow-Secure Systems with IFlow“. In: *2013 International Conference on Social Computing*. IEEE, 2013, S. 51–56.

[32] Haralambos Mouratidis, Jan Jürjens und Jorge Fox. „Towards a Comprehensive Framework for Secure Systems Development“. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2006, S. 48–62.

[33] Rodolfo Villarroel, Eduardo Fernández-Medina und Mario Piattini. „Secure Information Systems Development – A Survey and Comparison“. In: *Computers & Security* 24.4 (2005), S. 308–321.

[34] Amir A. Khwaja und Joseph E. Urban. „A Synthesis of Evaluation Criteria for Software Specifications and Specification Techniques“. In: *International Journal of Software Engineering and Knowledge Engineering* 12.5 (2002), S. 581–599.

[35] Raimundas Matulevičius und Marlon Dumas. „A Comparison of SecureUML and UMLsec for Rolebased Access Control“. In: *Proceedings of the 9th Conference on Databases and Information Systems*. 2010, S. 171–185.

[36] Katarzyna Mazur und Bogdan Ksiezopolski. „Comparison and Assessment of Security Modeling Approaches in Terms of the QoP-ML“. In: *Cryptography and Security Systems*. Hrsg. von Zbigniew Kotulski, Bogdan Księżopolski und Katarzyna Mazur. Berlin, Heidelberg: Springer, 2014, S. 178–192.

[37] Raimundas Matulevičius, Henri Lakk und Marion Lepmets. „An Approach to Assess and Compare Quality of Security Models“. In: *Computer Science and Information Systems* 8.2 (2011), S. 447–476.

[38] Muhammad U. A. Khan und Mohammed Zulkernine. „On Selecting Appropriate Development Processes and Requirements Engineering Methods for Secure Software“. In: *2009 33rd Annual IEEE International Computer Software and Applications Conference*. Bd. 2. 2009, S. 353–358.

[39] Matthias Klatte. *Konsistenzhaltung zwischen UML- und PCM-Komponentenmodellen*. Bachelorarbeit. Karlsruher Institut für Technologie, 2017.

[40] Jan Jürjens. *UML Analysis Tool*. URL: https://rgse.uni-koblenz.de/jj/umlsectool/index.html (besucht am 19. 02. 2019).

[41] Jens Bürger und Sven Peldszus. *CARiSMA*. URL: https://rgse.uni-koblenz.de/carisma/index.shtml (besucht am 19. 02. 2019).

[42] Philip Müller. *Masterarbeit - Abbildung von UMLsec-Vertraulichkeitsanalysen auf Data-Centric Palladio - Beispiele*. Apr. 2019. DOI: 10.5281/zenodo.2623271. URL: https://doi.org/10.5281/zenodo.2623271.

[43] Victor R. Basili und David M. Weiss. „A Methodology for Collecting Valid Software Engineering Data". In: *IEEE Transactions on Software Engineering* 10.6 (1984), S. 728–738.

[44] Philip Müller. *Masterarbeit - Abbildung von UMLsec-Vertraulichkeitsanalysen auf Data-Centric Palladio - Evaluation*. Apr. 2019. DOI: 10.5281/zenodo.2629939. URL: https://doi.org/10.5281/zenodo.2629939.

[45] Charles E. Metz. „Basic Principles of ROC Analysis." In: *Seminars in Nuclear Medicine* 8.4 (1978), S. 283–298.

[46] Per Runeson und Martin Höst. „Guidelines for Conducting and Reporting Case Study Research in Software Engineering". In: *Empirical Software Engineering* 14.2 (2008), S. 131–164.

A. Anhang

A.1. Evaluationsergebnisse von Secure Links

Tabelle A.1.: Die Ergebnisse der Experimente, die zur Beantwortung von Forschungsfrage F2.1 durchgeführt wurden. Dabei beschreibt $\mathcal{E}$ die erwartete Anzahl von Sicherheitsproblemen und $\mathcal{R}$ die tatsächliche Anzahl von Sicherheitsproblemen, die von der DCP-Analyse entdeckt werden.

UMLsec-Modell	DCP-Modell	$\mathcal{E}$	$\mathcal{R}$
umlsec-1	dcp-1	0	0
umlsec-2	dcp-2	0	0
umlsec-3	dcp-3	0	0
umlsec-4	dcp-4	0	0
umlsec-5	dcp-5	0	0
umlsec-6	dcp-6	0	0
umlsec-7	dcp-7	0	0
umlsec-8	dcp-8	0	0
umlsec-9	dcp-9	0	0
umlsec-10	dcp-10	2	2
umlsec-11	dcp-11	2	2
umlsec-12	dcp-12	2	2
umlsec-13	dcp-13	4	4
umlsec-14	dcp-14	4	4
umlsec-15	dcp-15	4	4
umlsec-16	dcp-16	6	6
umlsec-17	dcp-17	0	0
umlsec-18	dcp-18	0	0
umlsec-19	dcp-19	0	0
umlsec-20	dcp-20	2	2
umlsec-21	dcp-21	0	0
umlsec-22	dcp-22	2	2
umlsec-23	dcp-23	2	2
umlsec-24	dcp-24	2	2
umlsec-25	dcp-25	0	0
umlsec-26	dcp-26	0	0
umlsec-27	dcp-27	0	0

umlsec-28	dcp-28	0	0
umlsec-29	dcp-29	0	0
umlsec-30	dcp-30	0	0
umlsec-31	dcp-31	0	0
umlsec-32	dcp-32	0	0
umlsec-33	dcp-33	0	0
umlsec-34	dcp-34	0	0
umlsec-35	dcp-35	0	0
umlsec-36	dcp-36	0	0
umlsec-37	dcp-37	0	0
umlsec-38	dcp-38	0	0
umlsec-39	dcp-39	0	0
umlsec-40	dcp-40	0	0
umlsec-41	dcp-1	0	0
umlsec-42	dcp-10	2	2
umlsec-43	dcp-11	2	2
umlsec-44	dcp-12	2	2
umlsec-45	dcp-13	4	4
umlsec-46	dcp-14	4	4
umlsec-47	dcp-15	4	4
umlsec-48	dcp-16	6	6
umlsec-49	dcp-17	0	0
umlsec-50	dcp-18	2	2
umlsec-51	dcp-19	2	2
umlsec-52	dcp-2	0	0
umlsec-53	dcp-20	2	2
umlsec-54	dcp-21	4	4
umlsec-55	dcp-22	4	4
umlsec-56	dcp-23	4	4
umlsec-57	dcp-24	6	6
umlsec-58	dcp-25	0	0
umlsec-59	dcp-26	2	2
umlsec-60	dcp-27	2	2
umlsec-61	dcp-28	2	2
umlsec-62	dcp-29	4	4
umlsec-63	dcp-3	0	0
umlsec-64	dcp-30	4	4
umlsec-65	dcp-31	4	4
umlsec-66	dcp-32	6	6
umlsec-67	dcp-33	0	0
umlsec-68	dcp-34	0	0
umlsec-69	dcp-35	0	0
umlsec-70	dcp-36	0	0
umlsec-71	dcp-37	0	0
umlsec-72	dcp-38	0	0

umlsec-73	dcp-39	0	0
umlsec-74	dcp-4	0	0
umlsec-75	dcp-40	0	0
umlsec-76	dcp-5	0	0
umlsec-77	dcp-6	0	0
umlsec-78	dcp-7	0	0
umlsec-79	dcp-8	0	0
umlsec-80	dcp-9	0	0
umlsec-structural-1	dcp-structural-1	2	2
umlsec-structural-2	dcp-structural-2	**4**	**6**
umlsec-structural-3	dcp-structural-3	4	4
umlsec-structural-4	dcp-structural-4	**2**	**4**
umlsec-structural-5	dcp-transformed-1	**2**	**4**

Tabelle A.2.: Die Ergebnisse der Experimente, die zur Beantwortung von Forschungsfrage F2.2 durchgeführt wurden. Dabei beschreibt $\mathcal{E}$ die erwartete Anzahl von Sicherheitsproblemen und $\mathcal{R}$ die tatsächliche Anzahl von Sicherheitsproblemen, die von der DCP-Analyse entdeckt werden. Der erste Wert des Tupels beschreibt die Sicherheitsprobleme unter dem Standardangreifer und der zweite Wert des Tupels beschreibt die Sicherheitsprobleme unter dem Insiderangreifer.

DCP-Modell	UMLsec-Modelle	$\mathcal{E}$	$\mathcal{R}$
dcp-1	umlsec-1 & umlsec-41	(0, 0)	(0, 0)
dcp-2	umlsec-2 & umlsec-52	(0, 0)	(0, 0)
dcp-3	umlsec-3 & umlsec-63	(0, 0)	(0, 0)
dcp-4	umlsec-4 & umlsec-74	(0, 0)	(0, 0)
dcp-5	umlsec-5 & umlsec-76	(0, 0)	(0, 0)
dcp-6	umlsec-6 & umlsec-77	(0, 0)	(0, 0)
dcp-7	umlsec-7 & umlsec-78	(0, 0)	(0, 0)
dcp-8	umlsec-8 & umlsec-79	(0, 0)	(0, 0)
dcp-9	umlsec-9 & umlsec-80	(0, 0)	(0, 0)
dcp-10	umlsec-10 & umlsec-42	(1, 1)	(1, 1)
dcp-11	umlsec-11 & umlsec-43	(1, 1)	(1, 1)
dcp-12	umlsec-12 & umlsec-44	(1, 1)	(1, 1)
dcp-13	umlsec-13 & umlsec-45	(2, 2)	(2, 2)
dcp-14	umlsec-14 & umlsec-46	(2, 2)	(2, 2)
dcp-15	umlsec-15 & umlsec-47	(2, 2)	(2, 2)
dcp-16	umlsec-16 & umlsec-48	(3, 3)	(3, 3)
dcp-17	umlsec-17 & umlsec-49	(0, 0)	(0, 0)
dcp-18	umlsec-18 & umlsec-50	(0, 1)	(0, 1)
dcp-19	umlsec-19 & umlsec-51	(0, 1)	(0, 1)
dcp-20	umlsec-20 & umlsec-53	(1, 1)	(1, 1)
dcp-21	umlsec-21 & umlsec-54	(0, 2)	(0, 2)

dcp-22	umlsec-22 & umlsec-55	(1, 2)	(1, 2)
dcp-23	umlsec-23 & umlsec-56	(1, 2)	(1, 2)
dcp-24	umlsec-24 & umlsec-57	(1, 3)	(1, 3)
dcp-25	umlsec-25 & umlsec-58	(0, 0)	(0, 0)
dcp-26	umlsec-26 & umlsec-59	(0, 1)	(0, 1)
dcp-27	umlsec-27 & umlsec-60	(0, 1)	(0, 1)
dcp-28	umlsec-28 & umlsec-61	(0, 1)	(0, 1)
dcp-29	umlsec-29 & umlsec-62	(0, 2)	(0, 2)
dcp-30	umlsec-30 & umlsec-64	(0, 2)	(0, 2)
dcp-31	umlsec-31 & umlsec-65	(0, 2)	(0, 2)
dcp-32	umlsec-32 & umlsec-66	(0, 3)	(0, 3)
dcp-33	umlsec-33 & umlsec-67	(0, 0)	(0, 0)
dcp-34	umlsec-34 & umlsec-68	(0, 0)	(0, 0)
dcp-35	umlsec-35 & umlsec-69	(0, 0)	(0, 0)
dcp-36	umlsec-36 & umlsec-70	(0, 0)	(0, 0)
dcp-37	umlsec-37 & umlsec-71	(0, 0)	(0, 0)
dcp-38	umlsec-38 & umlsec-72	(0, 0)	(0, 0)
dcp-39	umlsec-39 & umlsec-73	(0, 0)	(0, 0)
dcp-40	umlsec-40 & umlsec-75	(0, 0)	(0, 0)
dcp-structural-1	umlsec-structural-1 & umlsec-transformed-1	(1, 1)	(1, 1)
dcp-structural-2	umlsec-structural-2 & umlsec-transformed-2	**(3, 4)**	**(2, 3)**
dcp-structural-3	umlsec-structural-3 & umlsec-transformed-3	(2, 3)	(2, 3)
dcp-structural-4	umlsec-structural-4 & umlsec-transformed-4	**(2, 8)**	**(1, 6)**

Tabelle A.3.: Die Abbildung zwischen DCP- und UMLsec-Modellen für die Experimente, die zur Beantwortung von Forschungsfrage F2.3 der Secure-Links-Evaluation durchgeführt wurden.

UMLsec-Modell vorher	DCP-Modell	UMLsec-Modell nachher
umlsec-1	dcp-1	umlsec-1
umlsec-2	dcp-2	umlsec-2
umlsec-3	dcp-3	umlsec-3
umlsec-4	dcp-4	umlsec-4
umlsec-5	dcp-5	umlsec-5
umlsec-6	dcp-6	umlsec-6
umlsec-7	dcp-7	umlsec-7
umlsec-8	dcp-8	umlsec-8
umlsec-9	dcp-9	umlsec-9
umlsec-10	dcp-10	umlsec-10
umlsec-11	dcp-11	umlsec-11
umlsec-12	dcp-12	umlsec-12
umlsec-13	dcp-13	umlsec-13
umlsec-14	dcp-14	umlsec-14
umlsec-15	dcp-15	umlsec-15

umlsec-16	dcp-16	umlsec-16
umlsec-17	dcp-17	umlsec-17
umlsec-18	dcp-18	umlsec-18
umlsec-19	dcp-19	umlsec-19
umlsec-20	dcp-20	umlsec-20
umlsec-21	dcp-21	umlsec-21
umlsec-22	dcp-22	umlsec-22
umlsec-23	dcp-23	umlsec-23
umlsec-24	dcp-24	umlsec-24
umlsec-25	dcp-25	umlsec-25
umlsec-26	dcp-26	umlsec-26
umlsec-27	dcp-27	umlsec-27
umlsec-28	dcp-28	umlsec-28
umlsec-29	dcp-29	umlsec-29
umlsec-30	dcp-30	umlsec-30
umlsec-31	dcp-31	umlsec-31
umlsec-32	dcp-32	umlsec-32
umlsec-33	dcp-33	umlsec-33
umlsec-34	dcp-34	umlsec-34
umlsec-35	dcp-35	umlsec-35
umlsec-36	dcp-36	umlsec-36
umlsec-37	dcp-37	umlsec-37
umlsec-38	dcp-38	umlsec-38
umlsec-39	dcp-39	umlsec-39
umlsec-40	dcp-40	umlsec-40
umlsec-41	dcp-1	umlsec-41
umlsec-42	dcp-10	umlsec-42
umlsec-43	dcp-11	umlsec-43
umlsec-44	dcp-12	umlsec-44
umlsec-45	dcp-13	umlsec-45
umlsec-46	dcp-14	umlsec-46
umlsec-47	dcp-15	umlsec-47
umlsec-48	dcp-16	umlsec-48
umlsec-49	dcp-17	umlsec-49
umlsec-50	dcp-18	umlsec-50
umlsec-51	dcp-19	umlsec-51
umlsec-52	dcp-2	umlsec-52
umlsec-53	dcp-20	umlsec-53
umlsec-54	dcp-21	umlsec-54
umlsec-55	dcp-22	umlsec-55
umlsec-56	dcp-23	umlsec-56
umlsec-57	dcp-24	umlsec-57
umlsec-58	dcp-25	umlsec-58
umlsec-59	dcp-26	umlsec-59
umlsec-60	dcp-27	umlsec-60

umlsec-61	dcp-28	umlsec-61
umlsec-62	dcp-29	umlsec-62
umlsec-63	dcp-3	umlsec-63
umlsec-64	dcp-30	umlsec-64
umlsec-65	dcp-31	umlsec-65
umlsec-66	dcp-32	umlsec-66
umlsec-67	dcp-33	umlsec-67
umlsec-68	dcp-34	umlsec-68
umlsec-69	dcp-35	umlsec-69
umlsec-70	dcp-36	umlsec-70
umlsec-71	dcp-37	umlsec-71
umlsec-72	dcp-38	umlsec-72
umlsec-73	dcp-39	umlsec-73
umlsec-74	dcp-4	umlsec-74
umlsec-75	dcp-40	umlsec-75
umlsec-76	dcp-5	umlsec-76
umlsec-77	dcp-6	umlsec-77
umlsec-78	dcp-7	umlsec-78
umlsec-79	dcp-8	umlsec-79
umlsec-80	dcp-9	umlsec-80
umlsec-structural-1	dcp-structural-1	umlsec-structural-1
umlsec-structural-2	dcp-structural-2	umlsec-structural-2
umlsec-structural-3	dcp-structural-3	umlsec-structural-3
umlsec-structural-4	dcp-structural-4	umlsec-structural-4
umlsec-structural-5	dcp-transformed-1	umlsec-structural-5

Tabelle A.4.: Die Abbildung zwischen DCP- und UMLsec-Modellen für die Experimente, die zur Beantwortung von Forschungsfrage F2.4 der Secure-Links-Evaluation durchgeführt wurden.

DCP-Modell vorher	UMLsec-Modelle	DCP-Modell nachher
dcp-1	umlsec-1 & umlsec-41	dcp-1
dcp-2	umlsec-2 & umlsec-52	dcp-2
dcp-3	umlsec-3 & umlsec-63	dcp-3
dcp-4	umlsec-4 & umlsec-74	dcp-4
dcp-5	umlsec-5 & umlsec-76	dcp-5
dcp-6	umlsec-6 & umlsec-77	dcp-6
dcp-7	umlsec-7 & umlsec-78	dcp-7
dcp-8	umlsec-8 & umlsec-79	dcp-8
dcp-9	umlsec-9 & umlsec-80	dcp-9
dcp-10	umlsec-10 & umlsec-42	dcp-10
dcp-11	umlsec-11 & umlsec-43	dcp-11
dcp-12	umlsec-12 & umlsec-44	dcp-12

dcp-13	umlsec-13 & umlsec-45	dcp-13
dcp-14	umlsec-14 & umlsec-46	dcp-14
dcp-15	umlsec-15 & umlsec-47	dcp-15
dcp-16	umlsec-16 & umlsec-48	dcp-16
dcp-17	umlsec-17 & umlsec-49	dcp-17
dcp-18	umlsec-18 & umlsec-50	dcp-18
dcp-19	umlsec-19 & umlsec-51	dcp-19
dcp-20	umlsec-20 & umlsec-53	dcp-20
dcp-21	umlsec-21 & umlsec-54	dcp-21
dcp-22	umlsec-22 & umlsec-55	dcp-22
dcp-23	umlsec-23 & umlsec-56	dcp-23
dcp-24	umlsec-24 & umlsec-57	dcp-24
dcp-25	umlsec-25 & umlsec-58	dcp-25
dcp-26	umlsec-26 & umlsec-59	dcp-26
dcp-27	umlsec-27 & umlsec-60	dcp-27
dcp-28	umlsec-28 & umlsec-61	dcp-28
dcp-29	umlsec-29 & umlsec-62	dcp-29
dcp-30	umlsec-30 & umlsec-64	dcp-30
dcp-31	umlsec-31 & umlsec-65	dcp-31
dcp-32	umlsec-32 & umlsec-66	dcp-32
dcp-33	umlsec-33 & umlsec-67	dcp-33
dcp-34	umlsec-34 & umlsec-68	dcp-34
dcp-35	umlsec-35 & umlsec-69	dcp-35
dcp-36	umlsec-36 & umlsec-70	dcp-36
dcp-37	umlsec-37 & umlsec-71	dcp-37
dcp-38	umlsec-38 & umlsec-72	dcp-38
dcp-39	umlsec-39 & umlsec-73	dcp-39
dcp-40	umlsec-40 & umlsec-75	dcp-40
dcp-structural-1	umlsec-structural-1 & umlsec-transformed-1	dcp-structural-1
dcp-structural-2	umlsec-structural-2 & umlsec-transformed-2	dcp-structural-2
dcp-structural-3	umlsec-structural-3 & umlsec-transformed-3	dcp-structural-3
dcp-structural-4	umlsec-structural-4 & umlsec-transformed-4	dcp-structural-4

A.2. Evaluationsergebnisse von Secure Dependencies

Tabelle A.5.: Die Abbildung der UMLsec-Modelle auf die DCP-Modelle für die Experimente, die zur Beantwortung von Forschungsfrage F2.1 der Secure-Dependencies-Evaluation durchgeführt wurden.

UMLsec-Modell	DCP-Modell
umlsec-1	dcp-1
umlsec-2	dcp-2
umlsec-3	dcp-3

umlsec-4	dcp-4
umlsec-5	dcp-5
umlsec-6	dcp-6
umlsec-7	dcp-7
umlsec-8	dcp-8
umlsec-9	dcp-9
umlsec-10	dcp-10
umlsec-11	dcp-11
umlsec-12	dcp-12
umlsec-13	dcp-13
umlsec-14	dcp-14
umlsec-15	dcp-15
umlsec-16	dcp-16
umlsec-17	dcp-17
umlsec-18	dcp-18
umlsec-19	dcp-19
umlsec-20	dcp-20
umlsec-21	dcp-21
umlsec-22	dcp-22
umlsec-23	dcp-23
umlsec-24	dcp-24
umlsec-25	dcp-25
umlsec-26	dcp-26
umlsec-27	dcp-27
umlsec-28	dcp-28
umlsec-29	dcp-29
umlsec-30	dcp-30
umlsec-31	dcp-31
umlsec-32	dcp-32
umlsec-33	dcp-33
umlsec-34	dcp-34
umlsec-35	dcp-35
umlsec-36	dcp-36
umlsec-37	dcp-37
umlsec-38	dcp-38
umlsec-39	dcp-39
umlsec-40	dcp-40
umlsec-41	dcp-41
umlsec-42	dcp-42
umlsec-43	dcp-43
umlsec-44	dcp-44
umlsec-45	dcp-45
umlsec-46	dcp-46
umlsec-47	dcp-47
umlsec-48	dcp-48

umlsec-49	dcp-49
umlsec-50	dcp-50
umlsec-51	dcp-51
umlsec-52	dcp-52
umlsec-53	dcp-53
umlsec-54	dcp-54
umlsec-55	dcp-55
umlsec-56	dcp-56
umlsec-57	dcp-57
umlsec-58	dcp-58
umlsec-59	dcp-59
umlsec-60	dcp-60
umlsec-61	dcp-61
umlsec-62	dcp-62
umlsec-63	dcp-63
umlsec-64	dcp-64
umlsec-65	dcp-65
umlsec-66	dcp-66
umlsec-67	dcp-67
umlsec-68	dcp-68
umlsec-69	dcp-69
umlsec-70	dcp-70
umlsec-71	dcp-71
umlsec-72	dcp-72
umlsec-73	dcp-73
umlsec-74	dcp-74
umlsec-75	dcp-75
umlsec-76	dcp-76
umlsec-77	dcp-77
umlsec-78	dcp-78
umlsec-79	dcp-79
umlsec-80	dcp-80
umlsec-81	dcp-81
umlsec-82	dcp-82
umlsec-83	dcp-83
umlsec-84	dcp-84
umlsec-85	dcp-85
umlsec-86	dcp-86
umlsec-87	dcp-87
umlsec-88	dcp-88
umlsec-89	dcp-89
umlsec-90	dcp-90
umlsec-91	dcp-91
umlsec-92	dcp-92
umlsec-93	dcp-93

umlsec-94	dcp-94
umlsec-95	dcp-95
umlsec-96	dcp-96
umlsec-97	dcp-97
umlsec-98	dcp-98
umlsec-99	dcp-99
umlsec-100	dcp-100
umlsec-101	dcp-101
umlsec-102	dcp-102
umlsec-103	dcp-103
umlsec-104	dcp-104
umlsec-105	dcp-105
umlsec-106	dcp-106
umlsec-107	dcp-107
umlsec-108	dcp-108
umlsec-109	dcp-109
umlsec-110	dcp-110
umlsec-111	dcp-111
umlsec-112	dcp-112
umlsec-113	dcp-113
umlsec-114	dcp-114
umlsec-115	dcp-115
umlsec-116	dcp-116
umlsec-117	dcp-117
umlsec-118	dcp-118
umlsec-119	dcp-119
umlsec-120	dcp-120
umlsec-121	dcp-121
umlsec-122	dcp-122
umlsec-123	dcp-123
umlsec-124	dcp-124
umlsec-125	dcp-125
umlsec-126	dcp-126
umlsec-127	dcp-127
umlsec-128	dcp-128