Refaktorisierung einer Architekturanalyse für Vertraulichkeit

Alina Valta

Institut für Informationssicherheit und Verlässlichkeit (KASTEL)
Betreuender Mitarbeiter: M.Sc. Frederik Reiche

1 Aufgabenstellung

Die Aufgabe dieses Praktikums ist eine Refaktorisierung der Projekte Confidentiality4CBSE[3] und PCM2Prolog[1]. Die Aufgabe besteht dabei aus zwei Teilen: Die Entfernung des Profil-Mechanismus aus dem Confidentiality4CBSE Projekt und dem Verfeinern des Modells, sodass der Informationsfluss genauer modelliert werden kann. Außerdem soll es sich bei dem Praktikum um eine Refaktorisierung handeln, weshalb der PCM2Prolog Generator so angepasst werden muss, dass trotz verändertem Modell die Ausgabe als Prolog Prädikate unverändert bleibt.

2 Setup

Um das Projekt Confidentiality4CBSE in Betrieb zu nehmen, müssen folgende Projekte und Submodule richtig in Ecplise eingebunden werden:

- Installation desEclipse Modeling Tools[1] (Version 2020-12)
- Installationen über den „Install new Software…“ Dialog von Eclipse:
  - Palladio Component Model[2] (Version 4.3)
  - SDQ Commons[3] (EMF Commons und MDSD Profiles Commons)
- Clonen folgender Projekte von GitHub (inklusive aller Submodule) und Import in Eclipse:

[1] https://www.eclipse.org/downloads/packages/release/2020-12/r/eclipse-modeling-tools
[2] https://updatesite.palladio-simulator.com/palladio-bench-product/releases/4.3.0/
[3] https://kit-sdq.github.io/updatesite/release/commons/

- Confidentiality4CBSE[4]
- PCM2Prolog[5]

Um die Vertraulichkeitsanalyse durchzuführen, wird das Modul `confidentialitymm` als Eclipse Application ausgeführt. In der inneren Eclipseinstanz können eigene Projekte erstellt werden oder eines Beispiel Projekte[6] importiert werden. Jetzt können die Dateien, die in in der Analyse berücksichtigt werden sollen markiert werden. Dann kann über Rechtsklick auf `Create Prolog XSB Code` die Generierung des Prolog Code gestartet werden. Das Ergebnis befindet sich dann im den `.P` Dateien des `src-gen` Ordner.

3 Vertraulichkeitsanalyse

Die Idee hinter der Vertraulichkeitsanalyse für die Komponenten basierte Softwareentwicklung ist in dem technischen Bericht[4] von Max Kramer beschrieben worden. Es geht dabei darum eine Analyse über die Vertraulichkeit des Systems auf Architekturebene durchzuführen.

Die Architektur des Systems wird im Palladio Komponenten Modell spezifiziert. Auf dieser Basis können dann, mit Hilfe des Confidentiality4CBSE Projekts, Vertraulichkeitsinformationen ergänzt werden. Durch den UML Profil-Mechanismus können bestehende Palladio Komponenten mit Stereotypen annotiert werden, die Informationen über die Vertraulichkeit von Informationsflüssen an Interfaces oder von Operationen angeben oder Eigenschaften von Ressourcen spezifizieren. Dabei kann es sich zum Beispiel um Maßnahmen zum Schutz von Server vor Angreifern handeln oder um Information welche Daten unverschlüsselt übertragen werden. Das Profil referenziert dabei Klassen aus dem Confidentiality-Modell. Dieses Modell definiert Schutzmechanismen, Angreifer und Datenset, die für die Architektur relevant sind.

Die Analyse erfolgt in zwei Schritten: Zuerst werden die modellierten Vertraulichkeitsinformationen in Prolog Prädikate übersetzt. Dies wird von dem Projekt PCM2Prolog umgesetzt. Danach werden die Prolog Prädikate analysiert und wenn es ein Problem gefunden wird, wird dieses textuell ausgeben. In diesem Praktikum sind jedoch nur die Schritte bis zu den Prolog Prädikaten relevant.

4 Modell

Das Modell besteht aus fünf Paketen: Data, Repository, System, Resources und Adversary. Diese Aufteilung wurde in der Refaktorisierung beibehalten. Die Funktionalität des Profils wird in dem neuen Modell den Paketen Repository und Resources hinzugefügt. Um den Profil-Mechanismus des Modells zu entfernen, mussten neue Klassen eingeführt werden, welche wiederum Referenzen auf Palladio Komponenten haben.

[4] https://github.com/KASTEL-SCBS/Confidentiality4CBSE.git
[5] https://github.com/KASTEL-SCBS/PCM2Prolog.git
[6] https://github.com/KASTEL-SCBS/Examples4SCBS

4.1 Data

Das Paket Data, zusehen in Abbildung 1, wird dazu verwendet Ein- und Ausgabedaten von Komponenten in einzelne Datensätze DataSets gruppieren. Die Trennung in einzelne DataSets könnte beispielsweise nach Benutzergruppen oder Zweck der Daten erfolgen. Als InformationFlow modellierte Datenflüsse ordnen die Ein- und Ausgabedaten der Komponenten dann diesen DataSets zu. Außerdem gib es die Möglichkeit eine Gruppe von Datensätze zu definieren: Eine DataSetMap. Diese kann zum einen dafür genutzt werden, um DataSets für verschiedene Zugriffsrechte zu definieren. Für jedes Zugriffsrecht wird ein SpecificationParameter definiert und für eine DataSetMap gibt es mehrere Datensätze (ParamiziedDataSetMapEntry) mit einem jeweils zugewiesenem Zugriffsrecht. Zum anderen kann ein Datensatz auch für verschiedene User definiert werden, wenn modelliert werden soll, dass mehrere User das System verwenden und diese nur teilweise die Daten des Anderen kennen dürfen. Dazu kann für ein Datensatz und einen User ein DataSetMapEnty erstellt werden.

Im ursprünglichen Modell hattet ein DataSetMapEntry kein eigenen Parameter, um den User zu identifizieren, sondern der Name des DataSetMapEntry wurde als Attributwert angegeben, wenn dieser User referenziert werden soll. Um diese Referenz explizit zu machen, wurde im neuen Modell die Klasse UserIdentifier hinzugefügt.

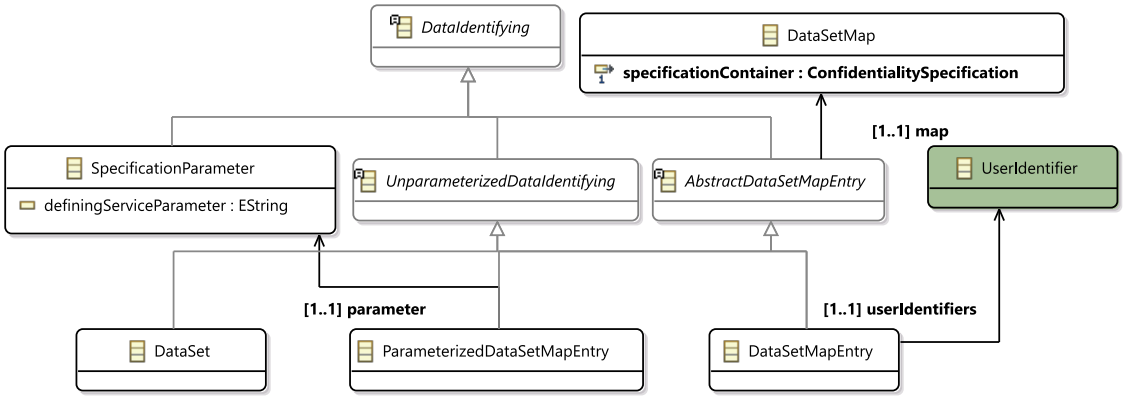

Abbildung 1: Klassendiagramm des Pakets Data mit neuer Klasse UserIdentifier, um Referenzen auf die User, die zu dem DataSetMapEntry gehören, zu ermöglichen.

4.2 System

Im Paket System, zusehen in Abbildung 2, wurde in der Klasse DataSetMapParameter2-KeyAssignment das Attribute assigendKey von einem String Attribut zu einer Referenz zu UserIdentifier geändert 2. Im bisherigen Modell wurde hier der Name eines DataSetMapEntry eingetragen. Durch diese Änderung wird die Beziehung zwischen einem DataSetMapEntry und der Klasse DataSetMapParameter2KeyAssignment explizit modelliert. Diese Klasse modelliert, dass ein Connector ein User authentifiziert und ihm ein bestimmtes Zugriffsrecht (SpecificationParameter) zuweist. Die Stereotypen InformationFlowParameterEquation und InformationFlowParameterAssignment3 des Confidentiality-Profils, zusehen in Abbildung 3,

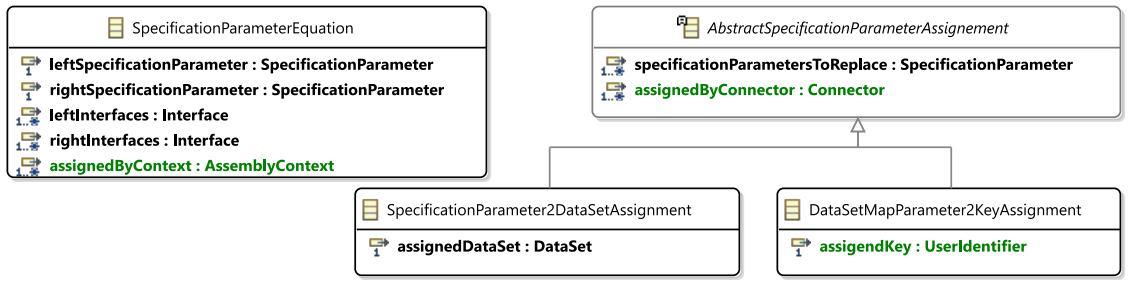

Abbildung 2: Klassendiagramm des Pakets System. Die grün markierten Referenzen wurden hinzugefügt, um die Stereotypen InformationFlowParameterEquation und InformationFlowParameterAssignment (siehe Abb. 3) zu ersetzen.

wurden entfernt, indem den Klassen SpecificationParameterEquation und AbstractSpecifiationParameterAssignment jeweils eine Referenz zu einem oder mehreren AssemblyContexten beziehungsweise Connectoren hinzugefügt wurde.

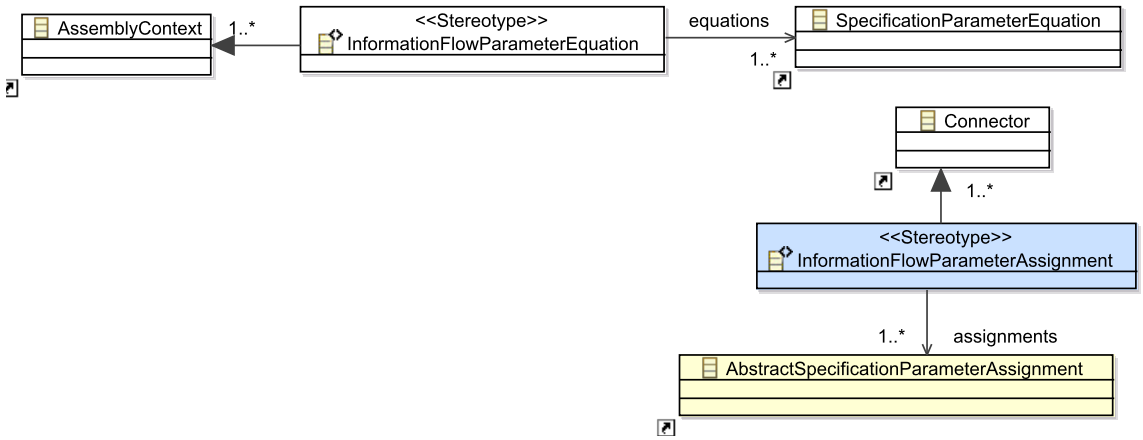

Abbildung 3: Teil des Confidentiality-Profils, welcher durch neue Referenzen im Paket System ersetzt wurde.

4.3 Repository

Das Paket Repository ist in Abbildung 4 zu sehen. Der InformationFlow Stereotyp weist Interfaces oder Signaturen ParameterAndDataPairs zu. Diese wiederum weisen ein oder mehrere DataSets bestimmte Ein- oder Ausgabedaten dieser Schnittstellen oder Methoden zu, indem das String Attribut parameterSources entweder der Name eines Parameter oder ”\call”, ”\return”, ”*” (alle Information) oder ”sizeof(<parameterSourceString>)” ist. Diese Modellierung ist uneindeutig, wenn mehrere Parameter den selben Namen haben. Außerdem muss der Syntax bekannt sein. Deshalb wurde im neuen Modell ein weiteres Paket Information (siehe Abbildung 5) eingeführt, das die ParameterAndDataPair Klasse ersetzen soll. Die abstrakte Klasse Information repräsentiert hierbei ein ParameterAndDataPair

welches jedoch nur noch eine parameterSource haben kann. Diese Einschränkung wurde vorgenommen, da so die Unterklassen direkt den Wert des String Attributes ersetzen können. Zum Beispiel ersetzt die Klasse CallInfomation ein ParamterAndDataPair mit dem String Attribut "\call". Die Klasse PararameterInformation hat jetzt eine explizite Referenz auf einen Parameter. Diese Variante wurde gewählt, da in keinem der Beispiele mehr als eine Wert für die parameterSources eines ParameterAndDataPair definiert wurden, sondern immer mehrere ParameterAndDataPairs erstellt wurden. Diese können auch teilweise die gleichen DataSets haben und auf die gleiche Signaturen angewandt werden. Der gleiche Sachverhalt konnte so im ursprünglichen Modell auf mehrere Weisen modelliert werden. Jetzt ist nur noch eine Variante möglich.

Der InformationFlow Stereotyp wurde entfernt, indem eine abstrakte Klasse AbstractInformationFlow eingeführt wurde, die mehrere Information Instanzen referenzieren kann und je nach Unterklasse auf ein Interface oder eine Signatur verweist. Im ursprünglichen Modell konnte ein InformationFlow auch auf mehre Signaturen angewandt werden, dies ist im neuen Modell nicht vorgesehen, da die Informations wie zum Beispiel CallInformation für jede Signatur andere Daten repräsentieren und deshalb pro Signatur als eigene Instanz der Klasse modelliert werden sollten. Außerdem sind Referenzen zu Parameter nur sinnvoll, wenn der InformationFlow auch nur auf die Signatur oder Interface dieses Parameters angewendet wird.

Die Stereotypen ServiceParameterAddition und InformationFlowParameter (siehe Abbildung 6) werden entfernt, indem die neue Klasse InformationFlowParameter einem AbstractInformationFlow einen AddedServiceParameter und mehrere SpecifiactionParamter zuweisen kann.

4.4 Resources

Das Klassendiagramm des Pakets Resources ist zu sehen in Abbildung 7. Dieses Paket übernimmt im neuen Modell auch die Aufgaben der Stereotypen Sharing, FutherPhysicalConnections, Encryption und LocationAndTamperProtection (siehe Abbildung 8). Die Stereotypen Sharing und FutherPhysicalConnections wurden durch die Klasse ResourceContainerConfidentiality ersetzt. Der Stereotyp Encryption wird ersetzt durch die Klasse Encryption. Die Klasse LocationsAndTemperProtectionsPair wird zur abstrakten Klasse AbstraktResource-

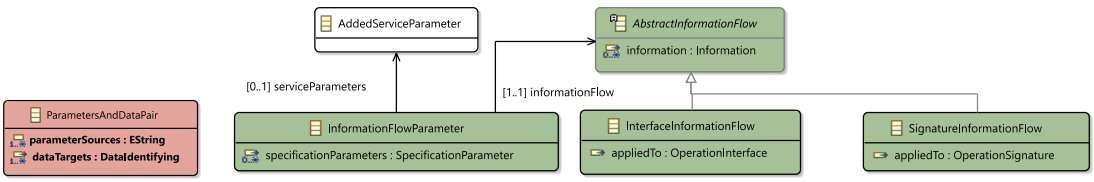

Abbildung 4: Klassendiagramm des Pakets Repository. Die grün markierten Klassen wurden hinzugefügt, um die Stereotypen ServiceParameterAddition und InformationFlowParameter zu ersetzen. Die rot markierte Klasse ParamterAndDataPair wurde aus dem ursprünglichen Modell entfernt. Ihre Funktionalität wird jetzt von dem Paket Information übernommen

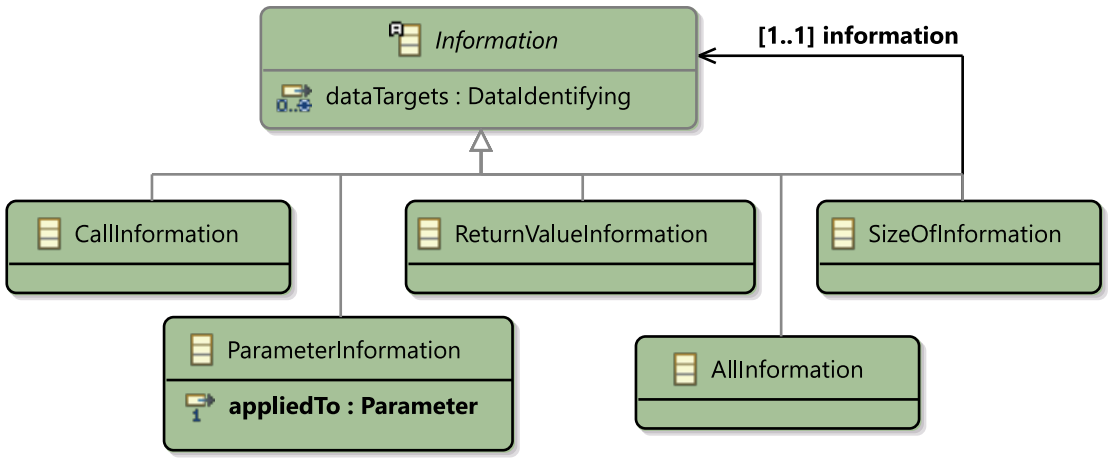

Abbildung 5: Klassendiagramm des Pakets Information. Diese neu hinzugefügten Klassen übernehmen die Aufgabe der ParamterAndDataPair Klasse

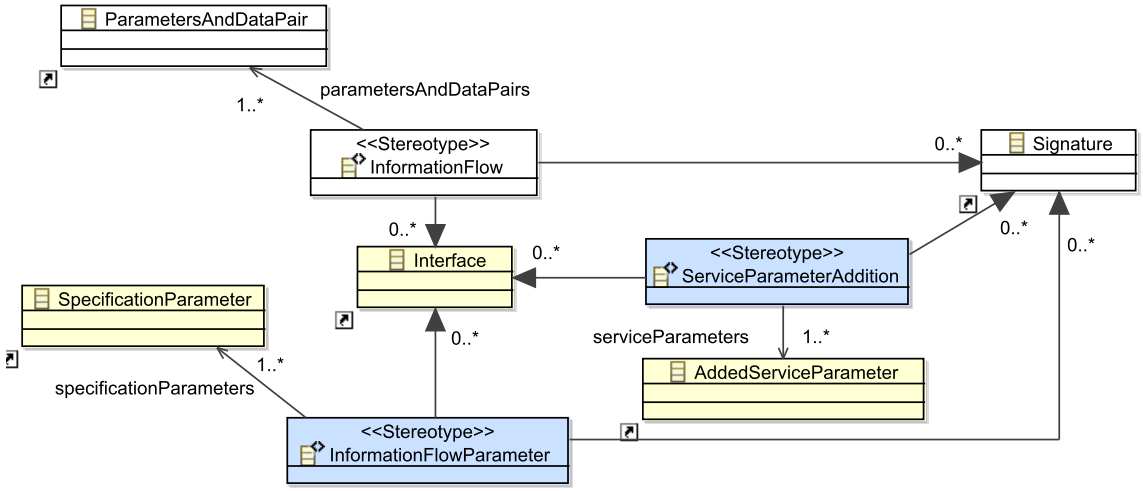

Abbildung 6: Teil des Confidentiality-Profils, welcher durch die neuen Klassen im Paket Repository ersetzt wird.

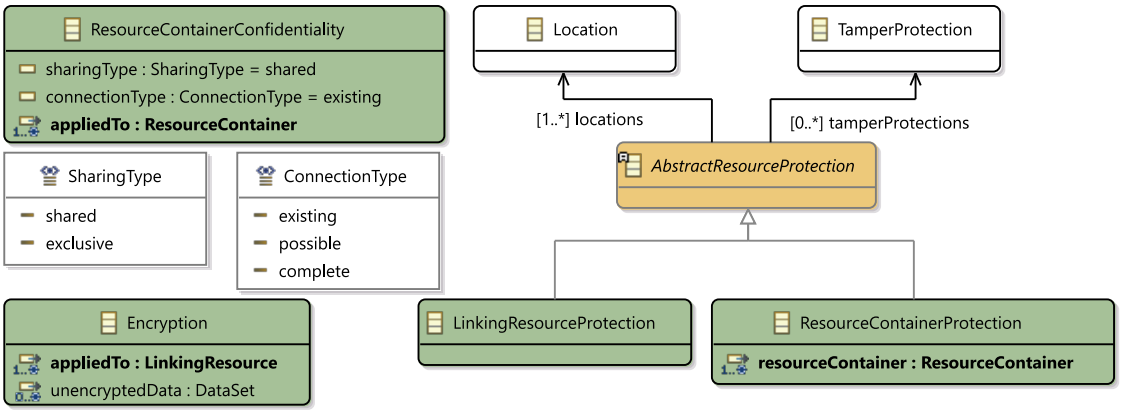

Abbildung 7: Klassendiagramm des Pakets Resources. Die grün markierten Klassen sind neu erstelle Klassen, um die Funktionalität der Stereotypen in Abbildung 8 zu übernehmen. Die gelb markierte Klasse AbstractResourceProtection ist die frühere LocationAndTamperProtectionsPair Klasse. Diese wurde lediglich umbenannt und zu einer abstrakten Klasse umgewandelt.

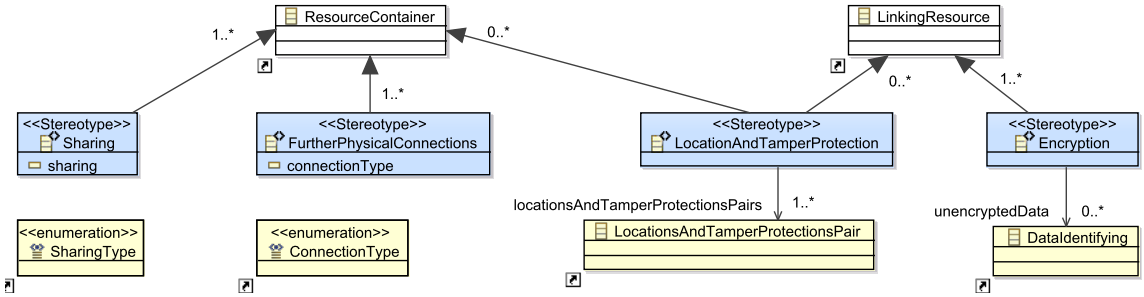

Abbildung 8: Teil des Confidentiality-Profils, welcher durch die neuen Klassen im Paket Resources ersetzt wird.

Protection, deren Unterklassen LinkingResourceProtection und ResourceContainerProtection referenzieren auf LinkingResources bzw. ResourceContainer, sodass der Stereotyp LocationAndTamperProtection nicht mehr benötigt wird. Allerdings kann eine AbstraktResourceProtection Instanz jetzt nur noch entweder auf LinkingResoucres oder auf ResourceContainer verweisen nicht mehr auf beides gleichzeitig. Wird dies benötigt, müssen dann jeweils eine LinkingResourceProtection und eine ResourceContainerProtection Instanz erstellt werden, die auf die gleichen Locations und TamperProtections zeigen.

4.5 Adversary

Das Paket Adversary (siehe Abbildung 9) hat sich nur dadurch verändert, dass nur die Referenz mayTamperWith nicht mehr auf die Klasse LocationAndTamperProtectionsPair zeigt, sondern auf die Klasse AbstractResourceProtection.

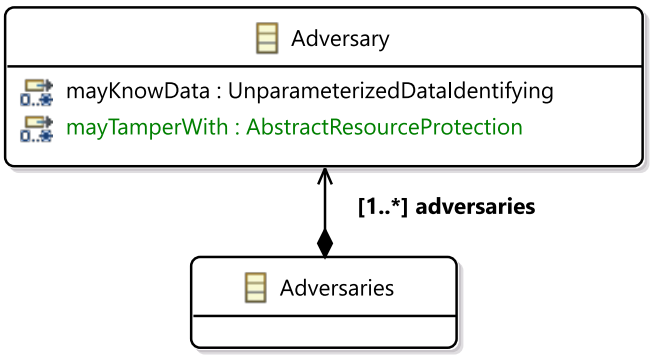

Abbildung 9: Klassendiagramm des Pakets Adversary. Die Referenz zu der Klasse AbstractResourceProtection wurde geändert, da die Klasse umbenannt wurde.

5 PCM2Prolog Generator

Der PCM2Prolog Generator wandelt Instanzen des Confidentiality-Modells in Prolog Prädikate um. Dieser musste im Rahmen dieses Praktikums so angepasst werden, dass trotz des veränderten Modells der Prolog Code identisch ist. Der Generator verwendet die Reflective-API von ECore und ist in Xtend geschrieben. Für jedes Entität wird die Methode `String generateDeeply(EObject e)` aufgerufen. Diese generiert für jedes Objekt und dessen Kinder Prolog Prädikate der Form: `referenceName("objId", ["refObjId1","refObjId2"])` für alle Referenzen und Attribute. In der Klasse `PCM2PrologXSBFilter` kann angegeben werden welche Klassen und Referenzen berücksichtigt werden sollen. Für Objekte, die nicht automatisch generiert werden können, können Dispatch-Methoden für die `String generateDeeply(EObject e)` Methode definiert werden. Dies wurde für viele der neu hinzugefügten Klassen gemacht, um sicherzustellen, dass die Prolog Prädikate identisch zum ursprünglichen Modell sind.

Durch das Entfernen des Profil-Mechanismus hat sich die Richtung der Referenzen der PCM Komponenten zu den Modell Klassen umgedreht. Im ursprünglichen Modell hat ein Stereotyp PCM Klassen erweitert und hatte Referenzen auf Modell Klassen. Ein solcher Stereotyp wurde in ein einziges Prädikat pro PCM Objekt umgewandelt. Im neuen Modell gibt es allerdings für jeden früheren Stereotyp eine Modell Klasse die Referenzen auf ein oder mehrere PCM Objekte hat. Damit diese Referenzen trotzdem in ein Prädikat pro ResourceContainer umgesetzt werden können, werden die Referenzen in einer Map gespeichert wobei das PCM Objekt als Key dient. So kann nach dem alle Objekte verarbeitet wurden, für jedes PCM Objekt der Map ein Prädikat generiert werden mit den entsprechenden Referenzen zu den zugehörigen Modell Klassen.

6 Evaluierung

In dem Repository `Examples4SCBS` [2] werden eine Reihe an Beispiel Projekten für die Vertraulichkeitsanalyse zur Verfügung gestellt.Um zu überprüfen, ob der generierte Prolog Code identisch zu dem Prolog Code vor der Refaktorisierung ist, wurden die Beispiel Projekte

cloudscenario-minimized und iflowexample im neuen Modell modelliert und das Ergebnis mit dem Ergebnis des ursprünglichen Modells verglichen. Diese zwei Beispiel Projekte wurden gewählt, da sie gemeinsam nahe zu alle Elemente des Confidentiality-Modells abdecken. Die einzige nicht verwendet Klasse ist die SpecificationParameter2DataSetAssignment Klasse. Diese wurde gesondert manuell getestet.

Um den Prolog Code automatisch vergleichbar zu machen, müssen beim Modellieren dieselben Identifier verwendet werden wie im original Projekt. Die Identifier der LocationAndTemperProtectionsPair Elemente müssen identisch zu den Identifierer der AbstractResourceProtection Elemente sein. Die Identifier der ParameterAndDataPair Elemente müssen identisch zu den Identifiern der Information Elemente sein. Die Identifier der anderen neuen Klassen können frei gewählt werden.

Beim Generieren des Prolog Codes müssen die Optionen fullIDs, singleFile und noComment ausgewählt werden, da bewusst die Kommentare nicht identisch sind, sondern so gewählt, dass auf die Entitäten, aus denen die Prolog Prädikate generiert wurden, zurück geschlossen werden kann. Im letzten Schritt müssen die Ausgabe Dateien noch vereinheitlicht werden. Dies ist lediglich notwendig um die die Dateien automatisch vergleichen zu können. Es gibt viele Prädikate der Form: `prädikatName("entity",["child1","child1","child3"]).` Die Reihenfolge der Elemente `child1` bis `child3` in der Liste hat keine Auswirkungen auf die Berechnung. Deshalb werden die Listenelemente solcher Prädikate alphabetisch sortiert. Außerdem kann sich die Reihenfolge der Zeilen unterscheiden, je nach dem in welcher Reihenfolge die Objekte verarbeitet wurden. Deshalb wird anschließend die Datei zeilenweise sortiert und leere Zeilen entfernt. Diese beiden Schritte können mit einem einfachen Shell Skript durchgeführt werden. Auf den vereinheitlichen Dateien kann dann der `diff` Befehl der GNU Diffutils ausgeführt werden.

Beide Beispiel Projekte `CloudScenario-minimized` und `IflowExample` konnten erfolgreich so im neuen Modell modelliert werden, dass der Prolog Code identisch ist. Allerdings hatten beide Projekte undefinierte Referenzen die zu zufälligen Identifiern in Prädikaten führen. Diese musste erst entfernt werden bevor der Vergleich durchgeführt werden konnte. Im `cloudscenario-minimized` Beispiel gab es eine Referenz in der `cloud.adverary` Datei auf ein LocationAndTemperProtectionsPair, welches nicht in der `cloud.confidentielity` Datei definiert war. Im `iflowexample` gab es undefinierte Referenzen innerhalb der Elementen `Allocation Energy Visualization`, `Allocation Database DBMS` und `Allocation Energy Meter` im File `default.allocation` und innerhalb der Elemente `ProvDelegation Provided AnInterface -> Provided AnInterface AComponent`, `Connector Energy Visualization -> Database` und `Connector Energy Visualization -> Energy Meter Assembly Context` und in der `default.system` Datei. Da diese Elemente nicht in der Confidentiality-Instanz referenziert werden, konnten diese gelöscht werden.

7 Fazit

Die Refaktorisierung der Vertraulichkeitsanalyse wurde vorgenommen, da es im ursprüngliche Modell implizite Referenzen im Hilfe von String Attributen gab und der Profil-Mechanismus in Kombination mit Eclipse in der Vergangenheit Probleme verursacht hat. Ersteres wurde durch die Modellierung als explizite Referenzen und eigene Klassen gelöst,

letzteres durch Einführen von neuen Klassen oder Erweitern von besehenden Klassen die Referenzen auf PCM Klassen haben. Außerdem musste der `PCM2Prolog` Generator angepasst werden, damit trotz der Änderungen am Modell der generiere Prolog Code gleich bleibt. Das konnte erreicht werden mit der Ausnahme von Wiederverwendung von Hilfsklassen wie zum Beispiel der Klasse ParameterAndDataPair oder LocationAndTemperProtectionsPair. In diesen Fällen kann der gleiche Sachverhalt aber auch durch mehrere dieser Klassen mit gleichen Parametern modelliert werden, deshalb ist es in diesen Fällen vertretbar, dass für diese Instanzen der Prolog Code nicht identisch ist.

Literatur

[1] Max E. Kramer, Martin Hecker und Frederik Reiche. *PCM2Prolog*. `https://github.com/KASTEL-SCBS/PCM2Prolog.git`.

[2] Max E. Kramer, Martin Hecker und Kateryna Yurchenko. *Examples4SCBS*. `https://github.com/KASTEL-SCBS/Examples4SCBS.git`.

[3] Max E. Kramer u. a. *Confidentiality4CBSE*. `https://github.com/KASTEL-SCBS/Confidentiality4CBSE.git`.

[4] Max E. Kramer u. a. *Model-Driven Specification and Analysis of Confidentiality in Component-Based Systems*. Techn. Ber. Karlsruhe: Karlsruhe Institute of Technology, Department of Informatics, 2017. DOI: `10.5445/IR/1000076957`. URL: `http://dx.doi.org/10.5445/IR/1000076957%20[Titel%20anhand%20dieser%20DOI%20in%20Citavi-Projekt%20%C3%BCbernehmen]`.