Attributbasierte Autorisierung in einer Branchenlösung für das Versicherungswesen

Analyse, Konzept und prototypische Umsetzung

Stefan Kapferer & Samuel Jost

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Betreuer: Prof. Dr. Olaf Zimmermann

Industriepartner: Adcubum AG

Experte: Dr. Gerald Reif

Gegenleser: Prof. Beat Stettler

Frühjahrssemester 2017

Inhaltsverzeichnis

Abstract

In adcubum SYRIUS®, einer geschichteten ERP-Lösung für Versicherungen, werden Berechtigungen direkt in der Datenbank verwaltet. Zugriffe auf die Daten aus SYRIUS werden in der Persistenzschicht autorisiert. Diese Lösung bringt Probleme mit sich, sobald Datenteilbestände in externen Komponenten, wie zum Beispiel einer Such- und Indexierungslösung, gehalten werden. Um diese Daten zu autorisieren, muss heute zusätzlich SYRIUS aufgerufen werden. Damit zukünftig Daten in einer externen Komponente autorisiert werden können, wurde in der Studienarbeit von Stefan Kapferer ein «Redesign» der Persistenzschicht vorgeschlagen und eine RESTful HTTP Schnittstelle für die neue Komponente entworfen. Die vorliegende Arbeit prüft, ob eine auf dem «Attribute-based Access Control» (ABAC)-Paradigma basierende Komponente die jetzige Berechtigungslösung ersetzen kann.

In dieser Bachelorarbeit wurden die fachlichen Schutzanforderungen an die Autorisierungskomponente in Zusammenarbeit mit Kunden von Adcubum aufgenommen und in Form von User Stories erfasst. Anhand der Anforderungen wurde analysiert, welche Informationen das System für die Autorisierungsentscheidungen benötigt und aus welchen Datenquellen diese bezogen werden. Weiterhin wurde ein Performancekostendach für die Autorisierungsanfragen festgelegt und untersucht, an welchen Stellen der vorgeschlagenen Architektur Performanceoptimierungen möglich sind. Policies zu den wichtigsten funktionalen Anforderungen zeigen auf, dass die technische Umsetzung mit dem ABAC-Paradigma möglich ist. Für die Erstellung der Policies wurden verschiedene Policy-Syntaxen evaluiert, wobei die Entscheidung auf die verbreitete XML-Sprache XACML fiel. Der entwickelte Prototyp verwendet die in der vorangegangenen Studienarbeit definierte Schnittstelle, um Autorisierungsanfragen auf Basis der verfassten XACML-Policies zu verarbeiten. Weiter wurden Vorschläge für das Migrationsvorgehen und dessen Herausforderungen erarbeitet.

Diese Arbeit liefert die konzeptionellen Grundlagen für die Entwicklung eines ABAC-basierten Autorisierungssystems. Die in dieser Arbeit erfassten funktionalen und nichtfunktionalen Anforderungen ermöglichen eine fundierte Evaluation eines Autorisierungsproduktes. Der Prototyp mit den verfassten XACML-Policies zeigt, dass sich die an SYRIUS gestellten Schutzanforderungen mit ABAC umsetzen lassen. Technische Risiken, welche der Architekturdesignwechsel hin zu ABAC mit sich bringt, konnten durch diese Arbeit minimiert werden.

1. Management Summary

1.1. Ausgangslage

In adcubum SYRIUS®, nachfolgend SYRIUS genannt, sind Zugriffsberechtigungen in einer relationalen Datenbank gespeichert. Die Persistenzschicht in SYRIUS autorisiert jeden Zugriff auf die Datenbank indem das SQL-Statement um eine WHERE-Klausel erweitert wird. Die Klausel stellt sicher, dass der Benutzer nur Daten laden kann, für welche er berechtigt ist. Dieser Ansatz wird kompliziert sobald eine Applikation Datenteilbestände ausserhalb der SYRIUS-Datenbank hält. Die Such- und Indexierungslösung Elasticsearch hält bereits heute solche Datenteilbestände. Die in Elasticsearch indexierten Daten können aktuell nur autorisiert werden, indem zusätzlich SYRIUS aufgerufen wird. Aus diesem Grund soll sich langfristig eine neue Autorisierungslösung etablieren, die als eigenständige Applikationskomponente («Microservice») agiert.

Die Studienarbeit von Stefan Kapferer stellt bereits eine auf RESTful HTTP basierende Autorisierungsschnittstelle zur Verfügung, die von SYRIUS und anderen Komponenten wie Elasticsearch verwendet werden soll. Ausserdem wird in der Studienarbeit aufgezeigt, an welchen Stellen ein «Redesign» der Persistenzschicht von SYRIUS durchzuführen ist, um diese Autorisierungsschnittstelle nutzen zu können.

Das Berechtigungskonzept der heutigen Lösung ist dem Role-Based Access Control (RBAC) Modell ähnlich. Die neue Autorisierungskomponente soll hingegen auf dem Attribute-Based Access Control (ABAC) Paradigma basieren, welches eine dynamischere Berechtigungsvergabe ermöglicht. Dieser Schritt ist gewünscht da der Parametrieraufwand in der heutigen Lösung erheblich ist. Das in der Studienarbeit entworfene «Architectural Refactoring» zeigt die Vor- und Nachteile eines solchen Wechsels von RBAC zu ABAC auf. Ausserdem beschreibt das «Architectural Refactoring» ein Vorgehen um diesen Wechsel zu vollziehen.

1.2. Vorgehen

Diese Bachelorarbeit hat die konkreten Anforderungen an die Autorisierungslösung zusammen mit vier zu dem Thema Berechtigungen repräsentativen Kunden der Adcubum erarbeitet. Die Anforderungen wurden dabei in Form von User Stories verfasst. Um die Berechtigungsanfragen zu beantworten, benötigt das Autorisierungssystem gewisse Infor-

mationen aus externen Systemen, wie zum Beispiel der Datenbank von SYRIUS. Diese Bachelorarbeit thematisiert, welche externen Systeme dabei als Informationsquellen agieren.

Auf der Basis der erarbeiteten Anforderungen wurde überprüft, ob sich diese technisch mit dem ABAC-Paradigma umsetzen lassen. Dafür wurde die Policy-Syntax XACML, welche als «de facto» Standard Implementation des ABAC-Paradigma bezeichnet werden kann, evaluiert und ausgewählt. Ob XACML den nichtfunktionalen Anforderungen der Adcubum bezüglich der Einfachheit und Wartbarkeit genügt, verifizierte der mit einem Consultant der Adcubum durchgeführte Benutzertest. Anschliessend wurden für die wichtigsten User Stories entsprechende XACML-Policies entwickelt. Die erarbeiteten Policies bildeten die Basis für einen in dieser Arbeit entwickelten Prototypen, der die Machbarkeit der angestrebten Autorisierungslösung beweist. Dieser in der Arbeit entwickelte Prototyp verwendet die Open Source Library «Balana», welche einen XACML-basierten Policy Decision Point (PDP) implementiert um Entscheidungen aufgrund der entwickelten Policies zu treffen. Der Prototyp ist dabei über die Autorisierungsschnittstelle aus der im letzten Abschnitt erwähnten Studienarbeit aufrufbar.

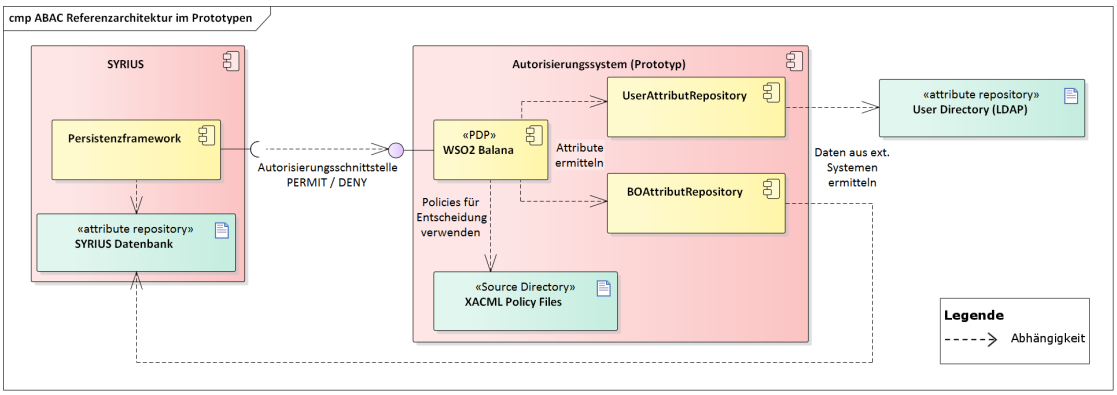

Abbildung 1.1.: Übersicht Prototyp (UML Komponentendiagramm)

Während der Bachelorarbeit wurde eine Performanceanalyse, die den Overhead der aktuellen Autorisierung misst, durchgeführt. Aus den Ergebnissen der Performanceanalyse wurde ein Performancekostendach für die neue Lösung abgeleitet. Damit diese Limite mit der vorgeschlagenen Architektur eingehalten werden kann, sind Performanceoptimierungen aufgeführt, die bei der Implementation der Autorisierungskomponente umgesetzt werden können.

Des Weiteren dokumentiert die Arbeit, wie bei der Migration der bestehenden Lösung hin zu dem ABAC-basierten Autorisierungssystem vorgegangen werden kann. Die XACML-Policies der neuen Lösung müssen dabei die Autorisierungskonfiguration aus der heutigen SYRIUS-Datenbank ablösen.

1.3. Ergebnisse

Die vorliegende Arbeit und insbesondere der im Rahmen der Arbeit entwickelte Prototyp zeigen auf, dass die funktionalen Anforderungen mittels eines ABAC-basierten Autorisierungssystems realisiert werden können. Die evaluierte Policy-Syntax erfüllt dabei die von Adcubum an die ABAC-Policies gestellten Anforderungen. Auf Basis der in dieser Arbeit dokumentierten User Stories ist Adcubum in der Lage, eine Produktevaluation des ABAC-basierten Produktes durchzuführen und später zu einem «Make or Buy» Entscheid zu gelangen. Das in dieser Arbeit definierte Performancekostendach legt die Obergrenze des erlaubten «Overheads» der neuen Lösung fest. Die vorgeschlagenen Performanceoptimierungen zeigen zudem Massnahmen auf, welche helfen die Performanceziele zu erreichen. Einige dieser Optimierungen identifizieren architektonische Herausforderungen, welche dabei auftreten können. Die Vorschläge für das Migrationsvorgehen weisen darauf hin, dass eine automatisierte Migration nicht zu empfehlen ist. Der Programmieraufwand für eine automiatisierte Erstellung der Zugriffsregeln und die Wartbarkeit der daraus generierten Policies sind nicht verhältnismässig. Es ist mit einer manuellen Erstellung der XACML-Policies zu rechnen.

1.4. Ausblick

Adcubum plant auf Basis der Anforderungen eine Evaluation von ABAC- bzw. XACML-basierten Produkten für die neue Autorisierungskomponente durchzuführen. Anschliessend kann die Implementationsphase der neuen Lösung geplant werden. Dabei müssen die entsprechenden Architekturumbauten innerhalb von SYRIUS, die notwendig sind um das Performancekostendach einzuhalten, berücksichtigt werden. Nachdem die Autorisierungslösung entwickelt und XACML-Policies für die Standardanwendungsfälle implementiert wurden, müssen gemeinsam mit den einzelnen Kunden Policies für die individuellen Spezialfälle ausgearbeitet werden. Dabei ist sicherzustellen, dass die Autorisierungsentscheide der neuen Lösung mit den Entscheidungen der heutigen Berechtigungslösung übereinstimmen.

2. Kontext

Die Adcubum AG ist der Industriepartner dieser Bachelorarbeit. Der Softwarehersteller bietet die Standardsoftware adcubum SYRIUS®, für Kranken-, Unfall- und Sachversicherungen an. SYRIUS ist eine Standardsoftware, welche spezifisch für Versicherungen entwickelt wurde und sämtliche Kernprozesse einer Versicherung abdeckt. Mit dem Ziel ein neues Autorisierungssystem einzuführen und den aktuellen Objektschutz im Persistenzframework abzulösen, wurde in der Studienarbeit [28] von Stefan Kapferer bereits eine auf RESTful HTTP basierende Autorisierungsschnittstelle entworfen und ein Refactoring in SYRIUS durchgeführt, welches die Benutzung der Autorisierungsschnittstelle ermöglicht.

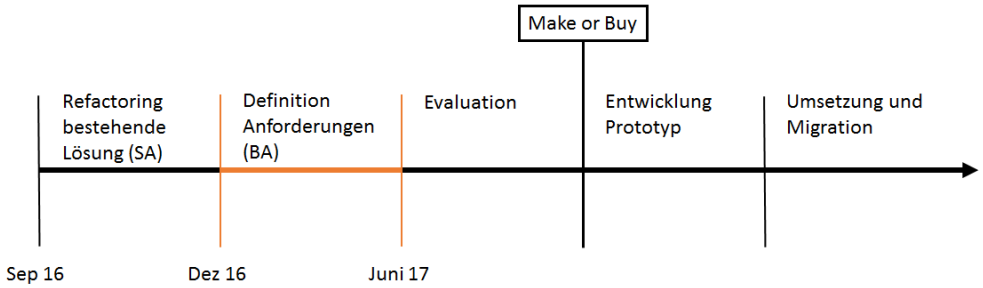

Abbildung 2.1.: Gesamtplanung von Adcubum aus Aufgabenstellung in Anhang C

Während dieser Bachelorarbeit wurde die «Mock»-Implementation hinter dieser Schnittstelle durch eine prototypische Implementation ersetzt und die Anforderungen an das neue Autorisierungssystem ermittelt. Ein Ziel dieser Arbeit war, für Adcubum die konzeptionellen Grundlagen zu erarbeiten, welche den späteren «Make or Buy» Entscheid unterstützen sollen. Dieses Kapitel gibt einen Überblick über SYRIUS, die aktuelle Autorisierungslösung, den gewünschten Zielzustand und den Scope dieser Arbeit.

2.1. SYRIUS

Das Backend von SYRIUS ist eine geschichtete Java EE Applikation, welche die Daten in einer Oracle [43] Datenbank persistiert. Die Benutzer greifen über den sogenannten Ultra-Thin Client (UTC), welcher keinerlei Businesslogik implementiert, auf die Applikation zu. Der Aufbau des User Interface (UI) wird auf dem *Presentation Server* gemacht, der über *Remote Method Invocation (RMI) over Internet Inter-Orb Protocol (IIOP)* mit dem Java EE *Application Server* kommuniziert. Die Kommunikation zwischen UTC und *Presentation Server* funktioniert über das Half Object plus Protocol (HOPP) [19]. Die Abbildung 2.2 gibt einen Überblick über diese Architektur von SYRIUS.

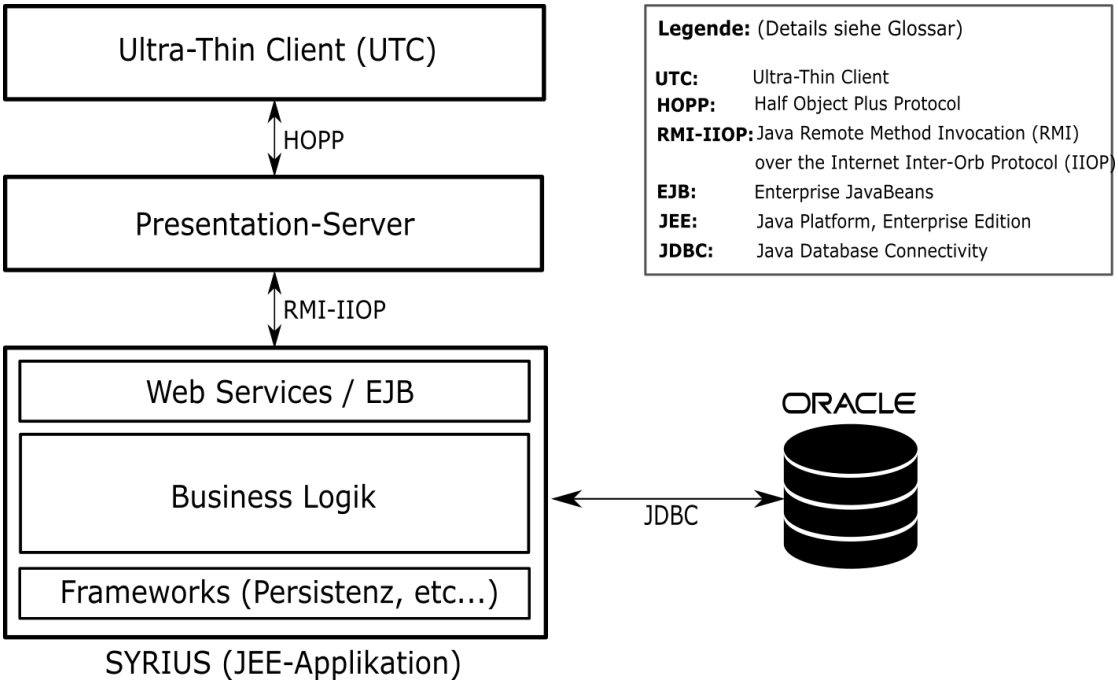

Abbildung 2.2.: SYRIUS Architektur [28]

Da die Bachelorarbeit das Backend behandelt, geht dieser Bericht nicht weiter auf die Komponenten *Presentation Server* und UTC ein.

2.2. Ausgangslage

Aktuell wird die Autorisierung der Business Objects (BOs) aus der relationalen SYRIUS-Datenbank über den sogenannten Objektschutz gelöst, welcher in der Persistenzschicht implementiert ist. Dabei wird jede Structured Query Language (SQL)-Abfrage um eine

entsprechende WHERE-Klausel erweitert, die sicherstellt, dass der Benutzer keine BOs aus der Datenbank laden kann, für welche er nicht berechtigt ist.

Auflistung 2.1 zeigt ein Beispiel einer solchen WHERE-Klausel. In diesem Fall wird der Partner über ein sogenanntes Partner-Schutzobjekt geschützt. Über die WHERE-Klausel wird sichergestellt, dass der Benutzer nur Partner laden kann, welche mit Schutzobjekten verknüpft sind für die der Benutzer berechtigt ist. Die Partner-Schutzobjekte sind für die Berechtigungslösung erstellte Objekte, welche es erlauben die Partner zu kategorisieren und Benutzergruppen auf einzelne Partnerkategorien zu berechtigen.

```
1  SELECT vertragId, itsPartner, ... FROM Vertrag vertrag
2          WHERE vertrag.vertragId = ...
3          AND EXISTS (
4                  SELECT partner.itsPartnerSchutz FROM Partner partner
5                          WHERE partner.BOId = vertrag.itsPartner
6                          -- Erlaubte Partnerschutzobjekte für den aktuellen Benutzer
7                          AND partner.itsPartnerSchutz IN
8                                  ('4','3','2','1')
9          );
```

Auflistung 2.1: Objektschutz: Erweiterung der WHERE-Klausel [28]

Neben dem Partner müssen weitere BOs, welche den Partner referenzieren, wie zum Beispiel Verträge des Partners, ebenfalls aufgrund des Partners geschützt werden. Dies wird über den sogenannten Schutzpfad gelöst. Das Konzept des Schutzpfades ist in Kapitel 3 unter Abschnitt 3.2.3 erklärt.

Auch wenn die jetzige Lösung mit den WHERE-Klauseln die Performance der SQL-Abfragen verlangsamt hat, ist sie insgesamt dennoch effizient. Wie die Performanceanalyse in Anhang A zeigt, benötigt die Autorisierung für die getesteten Use Cases zwischen durchschnittlich 22ms und 50ms. Allerdings ist diese Lösung nicht mehr zweckmässig, sobald Datenteilbestände ausserhalb der SYRIUS-Datenbank gehalten werden und auch andere Systeme ausser SYRIUS Autorisierungen durchführen sollen. Bereits heute werden Datenteilbestände im Such- und Indexierungssystem Elasticsearch [11] gehalten. Diese Daten können nicht autorisiert werden, ohne sie erneut per «Service-Call» aus SYRIUS zu laden und zu prüfen ob der Benutzer innerhalb von SYRIUS Zugriff auf die Daten hat. Dadurch können die externen Daten nicht sofort verwendet werden und der zusätzliche «Service-Call» verschlechtert die Performance. Des Weiteren sind in aktuellen «Front-Office» Projekten bei Adcubum neue Applikationen in Entwicklung, bei welchen auch die Endkunden, konkret die Versicherten, auf Daten aus SYRIUS zugreifen können. Dadurch müssen künftig auch Benutzer autorisiert werden können, welche nicht direkt SYRIUS-Benutzer sind. Aufgrund dieser Ausgangslage soll der jetzige Objektschutz aus der SYRIUS-Persistenzschicht ausgebaut und in einer externen Komponente gelöst werden.

2.3. Zielarchitektur

Adcubum arbeitet im Rahmen der strategischen Wartung mit Domain-Driven Design (DDD), um die einzelnen Module und deren Grenzen festzulegen. Diese Bestrebungen, welche in der Studienarbeit [28] detailliert festgehalten wurden, führen dazu, dass Module in Zukunft eigene «Microservices» [47, 48] bilden werden. Neben Elasticsearch [11] werden daher längerfristig weitere Applikationen in der Systemlandschaft um SYRIUS lokale Datenteilbestände halten, welche autorisiert werden müssen. Aus diesem Grund soll auch ein Autorisierungssystem entwickelt werden, welches eine eigene von SYRIUS entkoppelte Komponente, einen «Microservice» [47, 48], bildet.

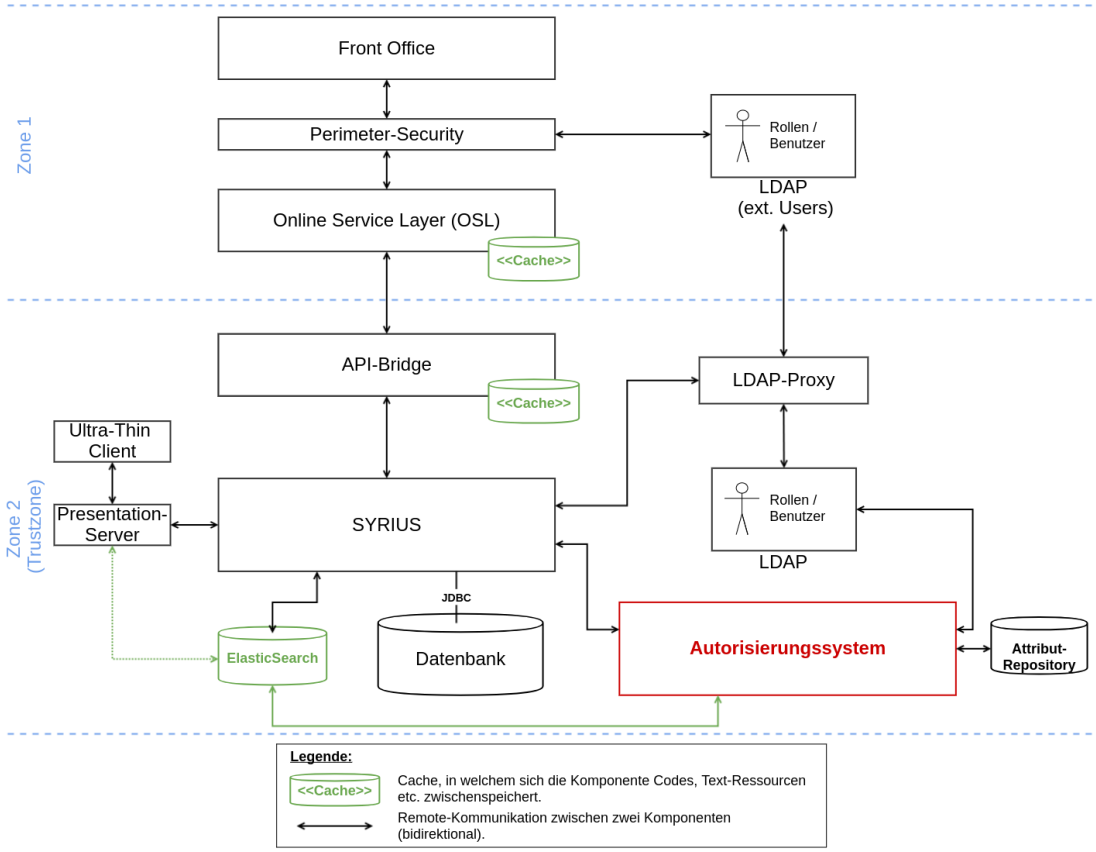

Abbildung 2.3.: SYRIUS Systemlandschaft mit zukünftigem Autorisierungssystem [28]

Diese Zielarchitektur, wie in Abbildung 2.3 dargestellt, wurde während der Studienarbeit [28] erarbeitet. An dieser Stelle dient die Referenz auf die Zielarchitektur, um aufzuzeigen in welchem Kontext sich diese Bachelorarbeit bewegt.

Somit ist ersichtlich in welchem Umfeld das Autorisierungssystem integriert werden soll

und wie die Systemlandschaft mittelfristig aussehen soll. In der Abbildung 2.3 sind es vorerst SYRIUS und Elasticsearch [11], welche das Autorisierungssystem aufrufen sollen. Neben den aufrufenden Systemen SYRIUS und Elasticsearch [11] benötigt das Autorisierungssystem eine Schnittstelle zu dem Lightweight Directory Access Protocol (LDAP)-Verzeichnis, da dort die Informationen der SYRIUS-Benutzer abgelegt sind. Ausserdem deutet die Abbildung 2.3 an, dass über den LDAP-Proxy auch Benutzerinformationen von Endkunden verwendet werden sollen. Somit können externe Benutzer, welche mit Endkunden-Applikationen wie zum Beispiel Webportalen oder Smartphone-Apps aus der in Abbildung 2.3 dargestellten «Zone 1» arbeiten, ebenfalls von der neuen Autorisierungsschnittstelle autorisiert werden.

2.3.1. Vor- und Nachteile der Zielarchitektur

Adcubum hat sich vor Beginn der Bachelorarbeit für diese Zielarchitektur ausgesprochen. Diese Entscheidung wird in der Bachelorarbeit nicht mehr evaluiert, sondern als Anforderung an das Projekt betrachtet. Vor- und Nachteile der angestrebten Architektur sind nicht detailliert in dieser Bachelorarbeit beschrieben. Zur Übersicht werden hier aber nochmals die Wichtigsten erläutert.

Vorteile

Ein gewichtiger Vorteil der neuen Lösung besteht darin, dass Datenzugriffe aus Drittanwendungen wie Elasticsearch ohne Aufruf von SYRIUS autorisiert werden können. Die neue externe Autorisierungskomponente sorgt dafür, dass die Autorisierung zentral an einem Ort gelöst ist. In Anbetracht der in Abbildung 2.3 dargestellten Zugriffe aus der «Zone 1» bietet die neue Lösung den Vorteil, dass zukünftig auch die Endkunden direkt autorisiert werden können.

Nachteile

Durch die getroffene Architekturentscheidung nimmt Adcubum den Nachteil in Kauf, dass durch die entkoppelte Komponente die Performance der Autorisierungsanfragen durch den zusätzlichen Aufruf des neuen Service beeinflusst werden kann. Die Autorisierungskomponente ist zudem nicht mehr Teil der Persistenzschicht, sondern muss als eigenständiger Service gewartet werden.

2.4. Attributbasierte Autorisierung ABAC

Das neue Autorisierungssystem soll auf dem Attribute-Based Access Control (ABAC)-Zugriffskontrollmodell basieren. Dies bedeutet neben dem Umbau der Architektur auch einen Wechsel des Zugriffskontrollparadigmas. Die jetzige Role-Based Access Control (RBAC)-ähnliche Lösung wird durch eine ABAC Komponente ersetzt. Welche Vorteile dies gegenüber der alten RBAC-basierten Lösung bringt und warum Adcubum ABAC verwenden möchte, wurde in der Studienarbeit [28] beleuchtet und festgehalten. Diese Entscheidung ist darum im Rahmen der Bachelorarbeit als gegeben zu betrachten und wird nicht mehr im Detail dokumentiert. Ausserdem schlägt die Studienarbeit [28] ein wiederverwendbares «Architectural Refactoring» [33, 65, 64] für den Wechsel von RBAC nach ABAC vor. Das angestrebte ABAC-Paradigma ist nachfolgend beschrieben.

2.4.1. Prinzip

Wie der Name «Attribute-based Access Control» (ABAC) bereits zum Ausdruck bringt, verwendet dieses Zugriffskontrollparadigma [41, 50, 3] Attribute um Autorisierungsentscheidungen zu treffen. Diese Attribute stammen typischerweise vom zugreifenden Benutzer, der zu schützenden Ressource, der zu autorisierenden Aktion, wie zum Beispiel «READ» oder «WRITE», oder dem System. Im Kontext von ABAC spricht man beim Benutzer vom *Subjekt* und bei der zu schützenden Ressource vom *Objekt*.

Die erwähnten vier Attributkategorien in ABAC lassen sich wiefolgt beschreiben:

- **Subjekt**: Der Benutzer, welcher auf ein bestimmtes Objekt zugreifen möchte.
- **Objekt**: Das Objekt oder die Ressource auf welche der Benutzer zugreifen will.
- **Aktion**: Die Aktion, welche der Benutzer auf dem Objekt oder der Ressource ausführen möchte. Typischerweise sind dies die Aktionen «READ» oder «WRITE» (Zugriffs- und Mutationsschutz).
- **Umgebung/System**: Attribute der Umgebung oder des Systems, wie zum Beispiel das aktuelle Datum oder der Clienttyp (Mobile App, Web, etc.).

Sogenannte Policies treffen beim ABAC-Paradigma [41, 50, 3] die Entscheidung, ob eine Anfrage «PERMIT» oder «DENY» zurückliefert. Dabei lassen sich aus den Attributen der Subjekte und Objekte Regeln bzw. Policies, wie zum Beispiel «Alle Benutzer der Abteilung Verkauf dürfen auf Dokumente des Typs Offerte zugreifen.», verfassen. Dieses Beispiel würde die beiden Attribute *benutzer.abteilung* des *Subjekts* und *dokument.typ* des *Objekts* benötigen.

2.4.2. Referenzarchitektur

Das ABAC-Paradigma kennt eine Referenzarchitektur, in der eine Reihe von Komponenten und deren Aufgaben vorgegeben sind. Die zentralste Komponente ist der Policy Decision Point (PDP). Er ist dafür zuständig die Entscheidungen zu treffen, ob ein *Subjekt* auf ein bestimmtes *Objekt* zugreifen darf. Der Policy Enforcement Point (PEP) ist die Komponente, welche den PDP aufruft und sicherstellen muss, dass dessen Entscheidung durchgesetzt wird.

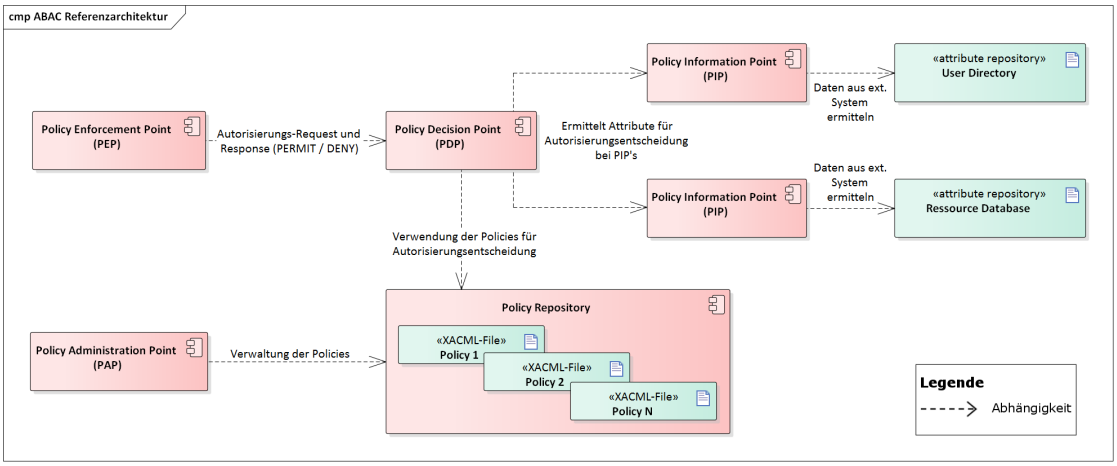

Abbildung 2.4.: ABAC Referenzarchitektur (UML Komponentendiagramm)

Wie aus Abbildung 2.4 hervorgeht, benötigt der PDP sogenannte Policy Information Points (PIPs) um die Attribute zu ermitteln, die er von externen Systemen bezieht um die Entscheidung zu treffen. Ausserdem verwendet der PDP Policies, welche die Regeln für den Zugriff festlegen. Die Verwaltung dieser Policies erfolgt über den Policy Administration Point (PAP).

Die einzelnen Komponenten der ABAC-Referenzarchitektur werden üblicherweise nicht alle als separate Applikationen bereitgestellt. Der PEP ist oft Teil der Applikation, welche Daten schützen und autorisieren möchte. Diese Applikation ruft dann ein ABAC-basiertes Autorisierungssystem auf, welches die Komponenten PDP, PIP und PAP vereint. Abbildung 2.5 zeigt diese typische Implementation der ABAC-Referenz-architektur in einem Komponentendiagramm.

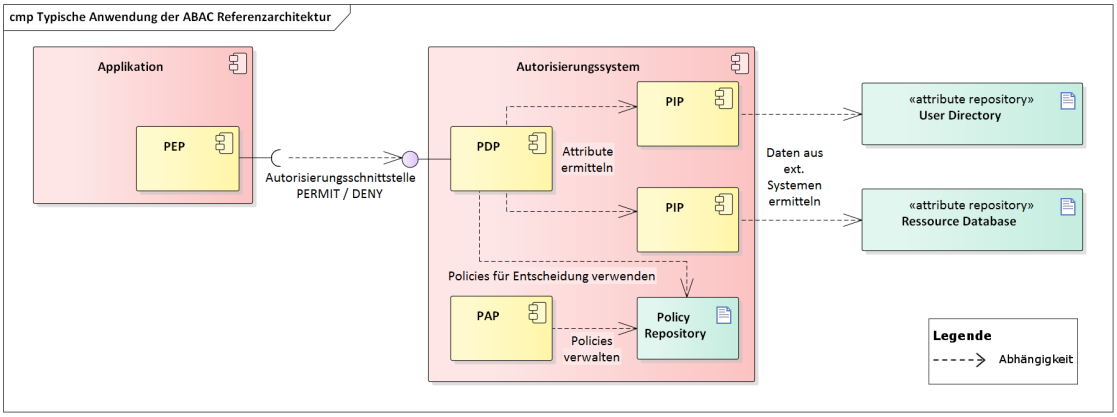

Abbildung 2.5.: Typische Anwendung der Referenzarchitektur (UML Komp.-Diagramm)

Die PIPs werden in diesem Fall innerhalb des Autorisierungssystems implementiert, beziehen ihre Daten aber von den benötigten externen Systemen. Adcubum strebt ein ähnlich aufgebautes Autorisierungssystem an.

Nachdem nun die von Adcubum mittelfristig angestrebte Zielarchitektur für SYRIUS und die Funktionsweise von ABAC aufgezeigt sind, geht der nächste Abschnitt auf den konkreten Umfang und die Abgrenzungen dieser Bachelorarbeit ein.

2.5. Umfang der Bachelorarbeit

In der Studienarbeit [28] wurde eine Autorisierungsschnittstelle entworfen, über welche die Systeme wie SYRIUS und Elasticsearch [11] zukünftig die Zugriffe auf Daten aus SYRIUS autorisieren sollen. Zusammen mit dem durchgeführten «Redesign» der Persistenzschicht von SYRIUS konnte die technische Machbarkeit, der neu angestrebten Architektur mit einem externen Autorisierungssystem, aufgezeigt werden.

Mit dieser Bachelorarbeit soll nun untersucht werden, inwiefern sich die Anforderungen an die Autorisierung in SYRIUS mittels einem ABAC-basierten System umsetzen lassen. Für einen späteren «Make or Buy» Entscheid und eine Produktevaluation eines ABAC-basierten Autorisierungssystems seitens der Adcubum, sollen in dieser Bachelorarbeit die Anforderungen an dieses Autorisierungssystem erarbeitet werden. Desweiteren soll die Umsetzbarkeit dieser Anforderungen mittels eines Prototypen gezeigt werden. Die exakte Aufgabenstellung befindet sich in Anhang C.

Um den Umfang der Bachelorarbeit festzulegen, wurden mit dem Industriepartner einige Abgrenzungsentscheidungen getroffen, welche nachfolgend erläutert werden.

2.5.1. Objektschutz vs. «Entitlement»

Um die Anforderungen an das neue System evaluieren zu können, muss festgelegt werden, welche Features der alten Berechtigungslösung abgelöst werden sollen. Diesbezüglich wurde in Absprache mit dem Industriepartner die Abgrenzung zwischen dem klassischen Objektschutz und «Entitlement» getroffen. Hinter dem Objektschutz befindet sich der Schutz der BOs aus SYRIUS, welcher die Daten vor dem Zugriff unberechtiger Benutzer schützen soll. Beim «Entitlement» hingegen geht es darum, dass gewisse Graphical User Interface (GUI)-Elemente aufgrund der Berechtigungen eines Benutzers ausgeblendet werden, weil er diese für seine Arbeit nicht benötigt. Das GUI wird dadurch übersichtlicher, was dem Benutzer das Arbeiten mit SYRIUS erleichtert. Das «Entitlement» deckt aber keine Schutzbegehren ab, da es aus Sicht des Datenschutzes kein Problem wäre wenn der Benutzer die ausgeblendeten Daten sehen würde.

Die Kategorie «Entitlement» umfasst Funktionen des aktuellen Berechtigungssystems wie den Registerkarten- (Tab-) Schutz oder den Taskschutz. Der Registerkartenschutz blendet bestimmte Registerkarten auf einzelnen GUIs für Benutzer aus, während der Taskschutz die Möglichkeit bietet, das Ausführen bestimmter Tasks durch Benutzer zu verhindern.

Zusammen mit dem Industriepartner wurde festgelegt, dass diese Bachelorarbeit die Funktionalität des Objektschutzes, nämlich die Autorisierung von BOs, betrachtet. Alle Funktionen im Zusammenhang mit «Entitlement» gehören damit nicht zum Umfang dieser Arbeit.

2.5.2. Anbindung von externen Systemen

Das ABAC-Konzept setzt voraus, dass das Autorisierungssystem Attribute der *Objekte*, in unserem Fall SYRIUS-BOs, und der *Subjekte* (Benutzer) zur Verfügung hat, um Autorisierungsanfragen aufgrund der entsprechenden Policies beantworten zu können. Dafür müssen PIPs entwickelt werden, welche die benötigten Attribute aus externen Systemen beziehen.

Die Anbindung externer Systeme gehört gemäss Absprache mit dem Industriepartner nicht zum Umfang dieser Arbeit, da die Machbarkeit innerhalb des Prototypen auch mittels «Mock»-Implementationen dieser PIPs gezeigt werden kann.

2.5.3. Aussagen zu Performance der neuen Lösung

Die Aufgabenstellung aus Anhang C definiert, dass die Performanceeigenschaften der neuen Lösung analysiert werden sollen. Konkret wurde in der Aufgabenstellung folgender Erfolgsfaktor festgelegt:

> «Es soll abgeschätzt werden, wie die Laufzeiteigenschaften der vorgeschlagenen Informationsbeschaffungslösung aussehen werden. Ausserdem soll ersichtlich sein, an welchen Stellen die Performance, z.B. durch Caching-Mechanismen, verbessert werden kann.»

Unter «Informationsbeschaffungslösung» wird der Mechanismus verstanden, welcher die benötigten Attribute für Autorisierungsentscheidungen aus den externen Systemen, den PIPs, bereitstellt.

Aufgrund der Entscheidung aus Abschnitt 2.5.2 externe Systeme nicht anzubinden, lassen sich die Laufzeiteigenschaften der Informationsbeschaffungslösung nicht aussagekräftig messen. Aus diesem Grund wurde definiert, dass für die neue Lösung ein Performancekostendach auf Basis der Performance der aktuellen Lösung festgelegt werden soll. Des Weiteren sollen, wie in der Aufgabenstellung festgelegt, Vorschläge für performanceverbessernde Massnahmen innerhalb der vorgeschlagenen ABAC-Architektur gemacht werden.

Die Performancemessung der neuen Lösung ist folglich nicht im Umfang der Bachelorarbeit enthalten, da eine solche Messung aufgrund der «Mock»-Implementationen der PIPs nicht aussagekräftig wäre. Auch diese Entscheidung wurde in Absprache mit dem Industriepartner getroffen. Durch das in Kapitel 3 Abschnitt 3.3 festgelegte Performancekostendach konnte die Anforderung des Industriepartners, ein messbares Kriterium in Bezug auf die Performance des neuen Systems festzulegen, trotzdem erfüllt werden.

Nachdem dieses Kapitel den Kontext und den Umfang der Bachelorarbeit beschrieben hat, geht das nächste Kapitel auf die Anforderungen an das neue Autorisierungssystem ein. Diese wurden durch die Analyse der aktuellen Objektschutz-Lösung und durch Kundeninterviews mit diversen Kunden der Adcubum erarbeitet.

3. Anforderungen

Ein wichtiges Ergebnis dieser Bachelorarbeit sind die detaillierten Anforderungen an das neue Autorisierungssystem. Für den Industriepartner Adcubum ist es wichtig, die fachlichen Hintergründe der in der jetzigen Lösung konfigurierten Berechtigungsdefinitionen zu kennen. Da die Kunden der Adcubum diesen Hintergrund am Besten kennen, sind viele der erfassten Anforderungen in Zusammenarbeit mit diesen Kunden entstanden. Neben den fachlichen Anforderungen, den Use Cases und User Stories, sind die nichtfunktionalen Anforderungen an das künftige Autorisierungssystem entscheidend. Die nichtfunktionalen Anforderungen (NFAs) sind gegen Ende dieses Kapitels dokumentiert.

3.1. Use Cases

3.1.1. Einleitung

Aus Sicht des Autorisierungssystems haben sich die in Abbildung 3.1 dargestellten Use Cases herauskristallisiert. Die Notation des Unified Modelling Language (UML) Use Case Diagramm basiert auf der Vorlage von Craig Larman [30].

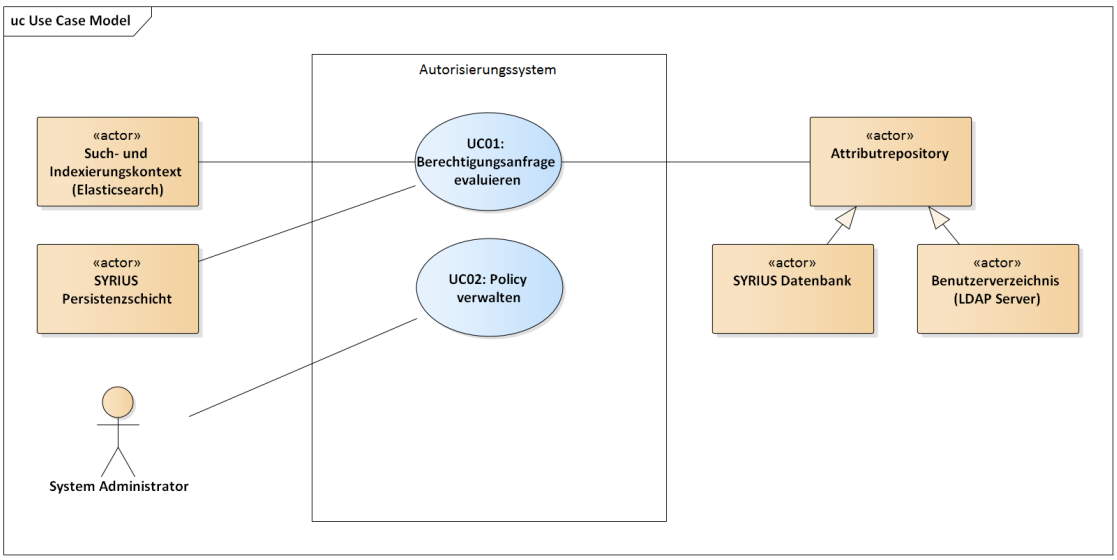

Abbildung 3.1.: Use Cases des Autorisierungssystems (UML Use Case Diagramm)

3.1.2. Aktoren

Folgende Aktoren interagieren mit den Use Cases des zu entwickelnden Autorisierungssystems:

Tabelle 3.1.: Aktoren

Name	Beschreibung
SYRIUS Persistenzschicht	Die Persistenzschicht von SYRIUS ist zuständig für das Laden und Speichern sämtlicher Objekte in der SYRIUS Datenbank. Bis jetzt wurden alle Berechtigungsanfragen direkt von der Persistenzschicht bearbeitet. Diese Funktionalität wird nun in das Autorisierungssystem ausgelagert. Die SYRIUS Applikation leitet die Anfragen an das Autorisierungssystem weiter. Aufgrund der Antwort des Autorisierungssystems muss SYRIUS die berechtigten Daten aus der Datenbank laden und dem Benutzer anzeigen.
Such- und Indexierungssystem (Elasticsearch)	*Elasticsearch* ist eine verteilte Such- und Analytikengine und wird im Umfeld von SYRIUS verwendet um Daten aus der Datenbank zu indexieren und somit schneller zu finden. Einige Suchanfragen welche im SYRIUS abgesetzt werden, verwenden die *Elasticsearch*-Instanz. Sobald die gesuchten Objekte gefunden wurden, werden die Daten über die Persistenzschicht in SYRIUS geladen. Alle Suchanfragen aus der *Elasticsearch*-Instanz werden von dem Autorisierungssystem überprüft.
System Administrator	Ein Mitarbeiter des Versicherers oder von Adcubum. Der *System Administrator* möchte das Autorisierungssystem auf seine fachlichen Anforderungen konfigurieren können. Früher wurde dies in Paramentrier-Objekten direkt in der Datenbank von SYRIUS gemacht. Im neuen Autorisierungssystem soll die Arbeit für den *System Administrator* einfacher werden, da die Syntax leichter zu lesen sein soll und das Erstellen von Policies einfacher ist als über die Parametrierungs-Graphical User Interfaces (GUIs) in SYRIUS.
Attributrepository	Da das Berechtigungssystem auf dem Attribute-Based Access Control (ABAC) Paradigma basieren soll, werden die Entscheidungen aufgrund von Attributen getroffen. Diese Attribute können entweder im Autorisierungsrequest enthalten sein, oder aus externen Systemen bezogen werden. Im Kontext von ABAC werden diese externen Systeme, welche die Attribute zur Verfügung stellen, Policy Information Points (PIPs) oder *Attributrepositories* genannt.

SYRIUS Datenbank	Die SYRIUS Datenbank persistiert die Businessobjekte welche letztendlich autorisiert werden. Für die Berechtigungsentscheidungen werden somit Daten aus SYRIUS benötigt. Die Datenbank dient als *Attributrepository*.
Benutzerverzeichnis (LDAP Server)	Die Benutzerdaten der Versicherer können aus einem *Benutzerverzeichnis* stammen, zum Beispiel einem Lightweight Directory Access Protocol (LDAP) Verzeichnis. SYRIUS erlaubt dem Versicherer entweder ein eigenes *Benutzerverzeichnis* zu verwenden, oder die Benutzerdaten direkt in SYRIUS zu speichern. Um Berechtigungsentscheidungen treffen zu können braucht das Autorisierungssystem bestimmte Daten aus dem *Benutzerverzeichnis*, welches damit ebenfalls als *Attributrepository* fungiert. Das können neben Benutzernamen auch Rollen oder Informationen zu den Mitarbeitern wie Geburtsdatum oder Funktion sein.

Die Kunden haben diverse konkrete Schutzanforderungen, für welche der Ablauf einer Autorisierung aus Sicht des Autorsierungssystems aber immer der Gleiche ist. Deshalb sind die folgenden Use Cases abstrakt verfasst und gelten für alle konkreten Schutzanforderungen. In Abschnitt 3.2 sind die detaillierten Schutzanforderungen in Form von User Stories definiert. Der Ablauf einer konkreten Autorisierung ist somit in «UC1: Berechtigungsanfrage evaluieren» beschrieben. Jede User Story in Abschnitt 3.2 verwendet den beschriebenen Ablauf aus UC1.

3.1.3. UC1: Berechtigungsanfrage evaluieren

Dieser Use Case deckt die bereits in der Studienarbeit [28] erarbeiteten Use Cases ab. Während der Studienarbeit [28] wurde ein konkreter Fall (allgemeiner Partnerschutz) betrachtet. Nun sollen die fachlichen Anforderungen an den Schutz verschiedenster Objekte in SYRIUS aufgezeigt werden. Die Form der Use Cases basiert auf der Vorlage von Craig Larman [30].

Use Case Brief

Ein Objekt in SYRIUS (Partner, Leistung, Organisationseinheit (OE)) soll geschützt werden, damit unberechtigte Mitarbeiter des Versicherers keine schützenswerten Informationen einsehen können. Manche Objekte, oder Attribute auf den Objekten, sind nur für bestimmte Mitarbeiter einsehbar.
Sobald ein Mitarbeiter über SYRIUS auf ein Objekt zugreifen möchte, wird aus der SYRIUS Persistenzschicht oder der Elasticsearch-Komponente eine Berechtigungsanfrage an das Autorisierungssystem geschickt. Diese Anfrage wird aufgrund der Daten aus

einem *Attributrepository* evaluiert. Je nach Antwort wird das angeforderte Objekt dem Mitarbeiter angezeigt oder nicht.

Use Case Fully Dressed

Tabelle 3.2.: UC1: Berechtigungsanfrage evaluieren - Fully Dressed

Name	Berechtigungsanfrage evaluieren
Primäraktor	SYRIUS Persistenzschicht, Elasticsearch
Stakeholder und Interessen	Für Versicherer und die versicherten Personen ist es überaus wichtig dass gewisse Objekte oder Attribute in SYRIUS nicht von jedem Mitarbeiter eingesehen oder bearbeitet werden. Häufig wird dies vom Datenschutzgesetz (DSG) vorgegeben und kann zu Schadensersatzforderungen und Klagen führen, wenn die Daten der Kunden nicht genügend geschützt wurden.
Preconditions	Das zu schützende Objekt und der Mitarbeiter sind in der SYRIUS Datenbank oder einem Benutzerverzeichnis (LDAP Server) vorhanden. Die benötigten Daten für eine Evaluation sind in den *Attributrepositories* vorhanden. Es existiert eine Policy, welche das Objekt vor unberechtigten Zugriffen schützt.
Postconditions	Das angeforderte Objekt wird dem Mitarbeiter entsprechend den Berechtigungen entweder ganz, nur teilweise, oder gar nicht angezeigt.
Haupt-Szenario: Benutzer darf Objekt nicht sehen	1. Der Sachbearbeiter greift in SYRIUS auf ein Business Object (BO) (Partner, Leistung, etc.) zu. 2. Die SYRIUS-Persistenzschicht sendet einen Autorisierungsrequest mit der ID des Benutzers, der ID des angeforderten Objektes, und der Art des Zugriffes (READ, WRITE) an das Autorisierungssystem. 3. Das Autorisierungssystem lädt aus den *Attributrepositories* die Informationen um eine Entscheidung zu treffen. 4. Die eingetragenen Policies werden aufgrund der geladenen Daten vom Autorisierungssystem ausgewertet. 5. Der Benutzer hat keine Berechtigung auf das Objekt und das Autorisierungssystem liefert daher «DENY» zurück. 6. Die SYRIUS Persistenzschicht stellt sicher dass das Objekt nicht geladen werden kann. 7. Der Benutzer kann nicht auf die Daten des Objektes zugreifen.

Alternative Szenarien	**A: Benutzer hat Berechtigung das Objekt zu sehen (READ):** A5: Der Benutzer hat die Berechtigung um das Objekt zu sehen. Das Autorisierungssystem liefert daher «PERMIT» zurück. A6: Die SYRIUS Persistenzschicht erlaubt, das Objekt zu laden. Dabei wird sichergestellt dass keine Mutationen erlaubt sind. A7: Der Benutzer kann das Objekt anzeigen aber nicht bearbeiten. **B: Benutzer mutiert die Daten anstatt sie nur anzuzeigen:** B2: SYRIUS teilt dem Autorisierungssystem im Autorisierungsrequest mit, dass der User die Operation «WRITE» auf dem BO ausführen möchte. **C: Benutzer darf nicht alle Attribute des Objektes sehen:** C5: Das Autorisierungssystem liefert ein «PERMIT» inkl. einer Liste von Attributen auf welche der Benutzer nicht zugreifen darf, entsprechend der Schnittstellendefinition aus der Studienarbeit [28]. C6: Die SYRIUS Persistenzschicht stellt sicher dass nur die erlaubten Attribute geladen werden können.

Use Case Scenario - VIP Kunden schützen

Der Use Case *UC1: Berechtigungsanfragen evaluieren* ist bewusst sehr abstrakt gehalten, da es viele verschiedene Schutzbegehren seitens der Kunden von Adcubum gibt. Der Ablauf wie oben im Abschnitt 3.1.3 beschrieben, gilt somit für alle Anwendungsfälle in welchen Daten in SYRIUS geschützt werden. Folgendes konkretes Szenario dient als Beispiel, um das Verständnis des Ablaufs zu stärken. Es verdeutlicht wie dieser Use Case effektiv aussehen kann. Für die übrigen Use Cases wird kein User Scenario dokumentiert.

Tabelle 3.3.: Use Case Scenario - VIP Kunden schützen

Name	Berechtigungsanfrage evaluieren - Very important Person (VIP) Kunden schützen
Primary Actor	SYRIUS Persistenzschicht
Stakeholder und Interessen	M arbeitet bei einer Versicherung A, welche auch Personen von öffentlichem Interesse (VIP) zu ihren Kunden zählt. Unter diesen VIPs ist auch der erfolgreiche Sportler «Patrick Superstar». Die Versicherung achtet darauf, dass die Daten der VIPs nicht jedem Angestellten einsehbar sind. Dafür wurden in der Software gewisse Sachbearbeiter als VIP-Betreuer definiert. Nur diese Mitarbeiter dürfen das vollständige Dossier der VIPs sehen. Anderen Angestellten wird in der Suche nur Name und Vorname angezeigt. Sollten die persönlichen Daten der VIPs an die Öffentlichkeit gelangen oder von allen Mitarbeitern eingesehen werden, könnte für Versicherung A ein grosser Imageverlust entstehen. Des Weiteren könnten auch Regressforderungen an die Versicherung A gestellt werden.
Preconditions	«Patrick Superstar» wurde in SYRIUS als VIP markiert. M ist nicht als VIP-Betreuer erfasst. Im Autorisierungssystem ist eine Regel definiert, die nur VIP-Betreuern erlaubt alle Informationen der VIPs auszulesen.
Postconditions	M sieht Name und Vorname von «Patrick Superstar» bei seiner Suchanfrage, allerdings keine Details.
Haupt-Szenario: M ist kein VIP-Betreuer	1. M sucht in SYRIUS Informationen über den VIP «Patrick Superstar». 3. SYRIUS sendet einen Autorisierungsrequest mit der ID von M, der ID von «Patrick Superstar», und der Zugriffsart READ an das Autorisierungssystem. 4. Das Autorisierungssystem lädt aus den *Attributrepositories* die Informationen von M und «Patrick Superstar». 5. Aufgrund der Policy für den VIP-Schutz wird überprüft ob M als VIP-Betreuer berechtigt ist. 6. M ist kein VIP-Betreuer. Das Autorisierungssystem schickt als Antwort «PERMIT» zurück. Zusätzlich wird eine Liste mit allen nicht autorisierten Attributen von «Patrick Superstar» mitgeschickt. 7. Die SYRIUS Persistenzschicht stellt sicher dass vom Partnerobjekt von «Patrick Superstar» nur Vor- und Nachname geladen wird. 8. M bekommt Namen und Vornamen von «Patrick Superstar» angezeigt, der Rest seiner Felder ist mit ## gefüllt.

3.1.4. UC2: Policy verwalten

Use Case Brief

Um die fachlichen Anforderungen in dem Autorisierungssystem abzubilden, müssen sogenannte Policies definiert werden. Aufgrund dieser Policies werden dann Berechtigungsanfragen ausgewertet. Der Versicherer beschäftigt einen System Administrator, der die fachlichen Anforderungen im System erfasst. Die Policies können durch den System Administrator auch editiert oder entfernt werden.

Use Case Fully Dressed

Tabelle 3.4.: UC2: Policy verwalten - Fully Dressed

Name	Policy definieren
Primary Actor	System Administrator
Stakeholder und Interessen	Der Versicherer hat ein neues Schutzbegehren und möchte dieses in dem Autorisierungssystem implementiert haben. Für den System Administrator muss das Einpflegen der neuen Policy auch ohne Programmierkenntnisse innerhalb von 1 bis 2 Stunden durchführbar sein. Dabei darf der Betrieb des Systems nicht beeinflusst werden. Ausserdem wird bei dieser Zeitangabe davon ausgegangen, dass alle Attribute welche für die neue Policy benötigt werden in den Attributrepositories bereits zur Verfügung stehen.
Preconditions	SYRIUS und das Autorisierungssystem sind bei dem Versicherer vorhanden. Der System Administrator kennt das Grundkonzept von ABAC und ist in der Lage eine Policy zu verfassen.
Postconditions	Das Autorisierungssystem berücksichtigt die neu eingefügte Regel ab sofort bei entsprechenden Berechtigungsanfragen.
Haupt-Szenario	1. Der System Administrator definiert die Policy in der vom Autorisierungssystem vorgegebenen Syntax. 2. Der System Administrator pflegt die neue Policy über eine Wartungsschnittstelle ein. 3. Das Autorisierungssystem lädt die Änderung. 4. Die neue Policy wird von dem Autorisierungssystem ab sofort bei der Evaluation der Berechtigungsanfragen verwendet.

Alternatives Szenario	**A: Autorisierungssystem ist nicht erreichbar:** A3: Die Änderung des System Administrators ist nicht kompatibel und wird ignoriert. A4: Der System Administrator wird über den Fehler informiert.

3.2. User Stories

3.2.1. Einleitung

Nachdem im letzten Abschnitt der Ablauf einer Autorisierung im Use Case (UC)1 beschrieben wurde, geht dieser Abschnitt nun auf die konkreten Anforderungen der Kunden von Adcubum an den Datenschutz ein. Diese Anforderungen werden in Form von User Stories (USs) dokumentiert. Sie definieren welche Daten die Versicherer, aufgrund welcher Objekte und Attribute, schützen wollen. Die User Stories repräsentieren die fachlichen Anforderungen an den Objektschutz in SYRIUS.

Alle User Stories wurden gemeinsam mit Kunden von Adcubum im Rahmen von Interviews oder Anforderungsanalyse per Mail erarbeitet, und stellen dadurch effektive Schutzbegehren dar, welche im Betrieb von SYRIUS auftreten. Verwendete Fachbegriffe aus dem Versicherungswesen oder aus SYRIUS werden im Glossar erläutert.

Die nachfolgenden User Stories basieren auf der Vorlage von Mike Cohn [9]. Ausserdem sind sie mit Akzeptanzkriterien [51] angereichert, um festzulegen wann die User Stories erfüllt sind und wie sie getestet werden können.

3.2.2. Rollen

Nachfolgend werden die Rollen, welche anschliessend in den User Stories verwendet werden, erklärt:

Tabelle 3.5.: User Stories - Rollen

Rolle	**Beschreibung**
Versicherer	Unter dem Begriff *Versicherer* wird die Versicherungsunternehmung verstanden, welche die Endkunden versichert und die dazugehörigen Daten in SYRIUS verwaltet. Ein *Versicherer* ist daran interessiert die Daten seiner Kunden vor unbefugten Zugriffen zu schützen. Unbefugte Datenzugriffe können für den *Versicherer* zu unzufriedenen Kunden und Imageverlust führen. Dies kann weitreichende Einflüsse auf die Businessziele der Versicherungsunternehmung haben.

Operating-Verantwortlicher	Der *Operating-Verantwortliche* ist ein Angestellter des *Versicherers* und kümmert sich um den Betrieb von SYRIUS. Er benötigt technische Benutzer in SYRIUS, mit welchen er Massenverarbeitungen (Batch-Läufe) durchführen kann. Diese werden benötigt um bestimmte Mutationen effizient auf grossen Datenmengen auszuführen, damit dies nicht von Hand durchgeführt werden muss.
Leistungsbearbeiter	Ein *Leistungsbearbeiter* ist bei dem Versicherer angestellt um in einem spezifischen Geschäftsbereich wie Krankenversicherung, Unfallversicherung, Taggelder, etc. Leistungen zu bearbeiten. Da die verschiedenen Bereiche sehr unterschiedlich sein können, kann jeder *Leistungsbearbeiter* nur für einen Geschäftsbereich zuständig sein. Gegenüber einem normalen Sachbearbeiter hat der *Leistungsbearbeiter* mehr Rechte und darf Leistungen auf Partnern einsehen.
Konzernspezialist	Ein *Konzernspezialist* ist ein Mitarbeiter des Versicherers, welcher sich im Detail mit den speziellen Gegebenheiten bei Konzernen auskennt, in welchen verschiedene Firmen zusammengefasst werden. Dieser Fall tritt im Versicherungswesen dann auf, wenn die Versicherten juristische Personen sind, wie zum Beispiel bei Berufsunfallversicherungen. Da die einzelnen Firmen des Konzerns in verschiedenen OEs angelegt sind, muss der *Konzernspezialist* nicht aufgrund seiner OE berechtigt werden, sondern auf die Firmen des Konzerns.

3.2.3. Abhängige Objekte (Schutzpfade)

In den nachfolgenden User Stories wird jeweils von den Hauptobjekten, welche geschützt werden sollen, zum Beispiel dem Partner, gesprochen. In der jetzigen Lösung in SYRIUS werden diese *Schutzobjekte* genannt. Allerdings gibt es in der Regel diverse, vom Schutzobjekt abhängige Objekte, welche ebenfalls geschützt werden müssen. Zum Beispiel soll ein Benutzer, der einen bestimmten Partner nicht sehen darf, auch nicht auf dessen Verträge zugreifen.

In der heutigen Lösung ist dies über sogenannte Schutzpfade gelöst. In der Berechtigungsparametrierung von SYRIUS können damit alle Objekte, welche neben dem Schutzobjekt ebenfalls geschützt werden müssen, angegeben werden.

Abbildung 3.2 zeigt ein Beispiel eines solchen Schutzpfad. Darin wird der Partner als Schutzobjekt, inklusive der anderen BOs im Schutzpfad, geschützt.

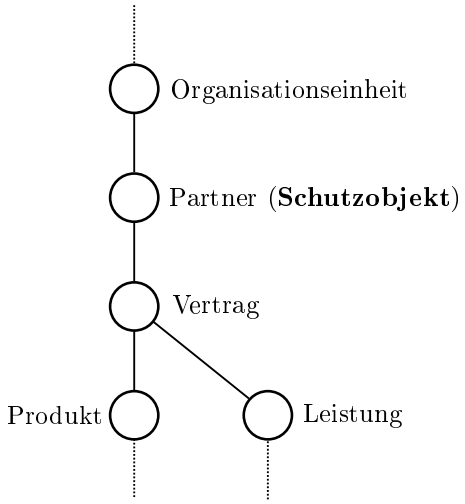

Abbildung 3.2.: Beispiel Schutzpfad [28]

Da diese Anforderung für alle nachfolgenden User Stories gilt, wird sie nur an dieser Stelle erläutert und nicht bei jeder Story. Bei jeder User Story kann es zusätzliche Objekte geben, welche aufgrund des selben Schutzbegehrens geschützt werden müssen und in SYRIUS eine Referenz auf das Schutzobjekt haben.

Innerhalb des Prototypen dieser Bachelorarbeit ist diese Funktionalität für mindestens eine User Story implementiert werden. Die Schutzpfade werden für die Standard User Stories, wie zum Beispiel den Mitarbeiterschutz, bereits in der heutigen Lösung mit der Basiskonfiguration von SYRIUS ausgeliefert. Die untenstehende Tabelle 3.6 zeigt einige vom Partner abhängige Objekte, welche ebenfalls geschützt werden müssen, wenn ein Sachbearbeiter auf den entsprechenden Partner keinen Zugriff hat.

Tabelle 3.6.: Beispiel-Schutzpfade

Zu schützendes BO	**Referenz zum Schutzobjekt «Partner»**
Adresse	itsPartner
Behandlung	itsPartner
Leistungsfall	itsLstFallPartner
Vertrag	itsPartnerVertrag
Vorbehalt	itsVorbehPartner
...	...

Die Liste dieser Objekte ist in der Praxis viel länger. Im Prototypen dieser Arbeit lässt sich die Machbarkeit des Konzepts an den obigen Beispielobjekten aufzeigen. Allerdings kann die Anzahl der zu schützenden Objekte die Performance der zukünftigen ABAC-basierten Lösung beeinflussen, da die Evaluation der Policies für den Policy Decision Point (PDP) aufwendiger wird.

Nachfolgend sind die konkreten Schutzbegehren der Kunden als User Stories dokumentiert.

3.2.4. US1: VIP Kunden schützen

Beschreibung

Als Versicherer möchte ich Partner von öffentlichem Interesse (VIP) nur durch bestimmte Sachbearbeiter einsehen und bearbeiten lassen, damit die privaten Daten der VIPs nicht in die Öffentlichkeit gelangen können. Da die persönlichen Daten eines VIP von grösserem Interesse sind als diejenigen eines normalen Kunden, müssen diese Daten noch besser geschützt werden.

Akzeptanzkriterien

- Den Sachbearbeitern kann das Recht auf die VIPs zuzugreifen erteilt oder entzogen werden.
- Ein unberechtigter Sachbearbeiter kann die VIPs nicht, oder nur einen Teil der Daten, sehen.

3.2.5. US2: Mitarbeiter schützen

Beschreibung

Als Versicherer möchte ich Partner, welche auch Mitarbeiter oder direkte Familienangehörige eines Mitarbeiters der Unternehmung sind, nur durch Angestellte der Abteilung Human Resources (HR) oder einer speziell geschaffenen OE einsehen und bearbeiten lassen, damit die privaten Daten des Mitarbeiters und seiner Familie nicht durch seine Arbeitskollegen eingesehen werden können und dessen Privatsphäre sichergestellt ist.

Neben dem Mitarbeiter werden auch weitere Familienmitglieder gleich behandelt, da diese im Datenmodell von SYRIUS über das Familienoberhaupt mit dem Mitarbeiter verknüpft sind. Dadurch können die Familien auch von vergünstigten Leistungen profitieren.

Akzeptanzkriterien

- Partner können in SYRIUS als Mitarbeiter deklariert werden.
- Der Zugriff auf private Daten eines Partners welcher Mitarbeiter ist, wird nur für Angestellte aus der Abteilung HR genehmigt.
- Ein unberechtigter Sachbearbeiter kann den Mitarbeiter nicht, oder nur einen Teil der Daten, sehen.

3.2.6. US3: Organisationseinheit schützen

Beschreibung

Als Versicherer möchte ich Partner, welche eine OE repräsentieren, zwar von allen Mitarbeitern suchen und anschauen lassen, aber nur durch bestimmte Sachbearbeiter, welche über die nötigen Ausbildungen und das benötigte Wissen verfügen, bearbeiten lassen, damit ich die Organisationsstruktur konsistent behalten kann und die Struktur nicht durch beliebige Mitarbeiter verändert wird.

OEs dienen als Stammdaten für diverse Prozesse in SYRIUS. Ungewollte oder fehlerhafte Mutationen dieser OE-Partner können zu fehlerhaften Resultaten dieser Prozesse führen. Weiter schützt sich der Versicherer mit diesem Mutationsschutz vor Betrugsfällen (z.B. durch Mutationen von Bankverbindungen).

Akzeptanzkriterien

- Partner welche eine OE repräsentieren, können als solche identifiziert werden.
- Eine Mutation auf einer OE kann nur durch den Leiter der OE oder durch das Management vorgenommen werden.
- Ein unberechtigter Sachbearbeiter kann die OE einsehen, jedoch nicht mutieren.

3.2.7. US4: Konzernspezialist berechtigen

Als Versicherer möchte ich einem Konzernspezialisten, einem Mitarbeiter der für einen Konzern zuständig ist, neben seiner zugeteilten OE, Zugriff auf sämtliche Partner innerhalb des von ihm betreuten Konzerns gewähren, sodass der Konzernspezialist seine Kunden wie gewünscht betreuen kann, ohne Einblick in zu viele Daten zu erhalten.

Firmen (Partner vom Typ juristische Person) können zu einem Konzern zusammengeschlossen werden. Dabei können Firmen aus verschiedenen OEs im gleichen Konzern

vereint werden. Als Konzernspezialist benötigt ein Mitarbeiter nun Rechte auf alle Partner, welche zum Konzern gehören, ohne auf alle betroffenen OEs den vollständigen Zugriff zu erhalten und somit auch unbeteiligte Partner einzusehen.

Akzeptanzkriterien

- Ein Konzernspezialist sieht alle Firmen welche dem Konzern zugeordnet sind, sogenannte Konzernmitglieder.
- Ein Konzernspezialist kann alle Firmen der OE, in welcher er arbeitet, bearbeiten.
- Der Konzernspezialist darf bei fremden OEs, nur die Firmen welche zum Konzern gehören, bearbeiten.

3.2.8. US5: Konzernspezialist auf zukünftiges Konzernmitglied berechtigen

Als Konzernspezialist möchte ich Zugriff auf den Konzern und seine Mitglieder auch wenn die Konzernzuordnung erst in der Zukunft gültig ist, damit ich bereits ab dem Zeitpunkt der Erfassung auf das zukünftige Konzernmitglied zugreifen kann, um mir bereits im Vorraus einen Überblick über die Daten und die Verträge des Konzernmitglieds verschaffen zu können.

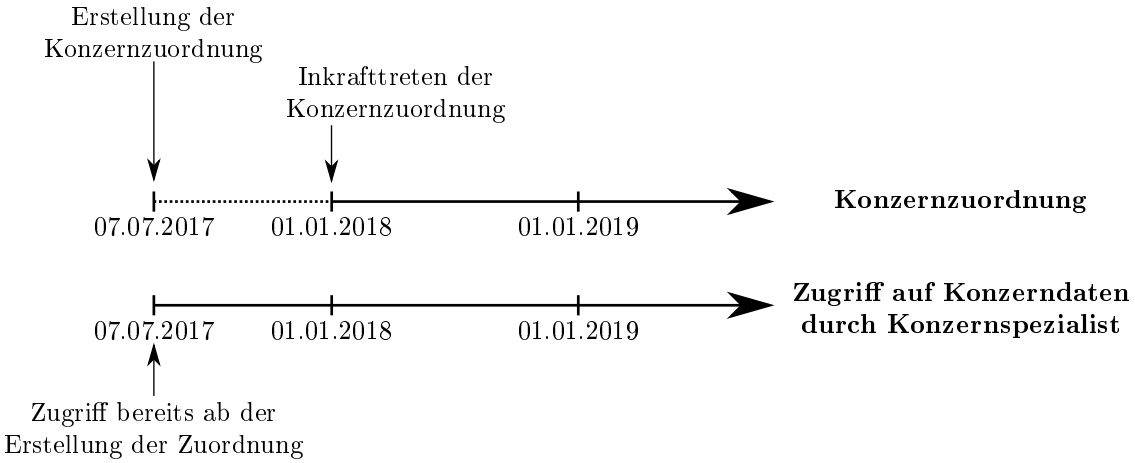

Abbildung 3.3.: Berechtigung aufgrund zukünftiger Konzernzuordnung

Durch eine Konzernzuordnung erhält ein Konzernspezialist, welcher Betreuer eines Konzernmitgliedes ist, Zugriff auf die gesamte Struktur und die Daten des Konzerns. Diese Zugriffsberechtigung muss der Konzernspezialist bereits mit der Erfassung erhalten, auch wenn die Zuordnung fachlich erst in der Zukunft gültig wird.

Akzeptanzkriterien

- Benutzer mit der Rolle Konzernspezialist haben ab dem Datum der Erstellung einer Konzernzuordnung den Zugriff auf das neue Konzernmitglied. Dafür müssen in SYRIUS bei der Bereitstellung der Berechtigungsattribute die zukünftigen States (fachliche Historisierung) berücksichtigt werden.
- Der Zugriff ist möglich, wenn zum aktuellen Datum oder irgendwann in der Zukunft eine Konzernzuordnung existiert.
  - Beispiel: Wenn eine Firma ab 01.01.2018 in einem Konzern ist, soll der entsprechende Konzernspezialist dieser Firma schon ab dem 07.07.2017, dem Zeitpunkt der Erfassung der Zuordnung, Zugriff auf den Konzern erhalten.

3.2.9. US6: Alten und neuen Konzernspezialisten bei einem Konzernspezialisten-Wechsel zeitlich überschneidend berechtigen

Als Versicherer möchte ich, dass bei einem Wechsel des Konzernspezialisten, der alte Konzernspezialist noch bis Ende des Kalenderjahres, und der neue Konzernspezialist bereits ab der Erfassung des Wechsels Zugriff auf die Konzerndaten hat, damit der neue Konzernspezialist bereits Einblick in die Struktur des Konzerns erhält und der alte Konzernspezialist bis Ende des Kalenderjahres seine Funktion ausüben kann.

In Abbildung 3.4 wird ein Beispiel für dieses Verhalten gezeigt.

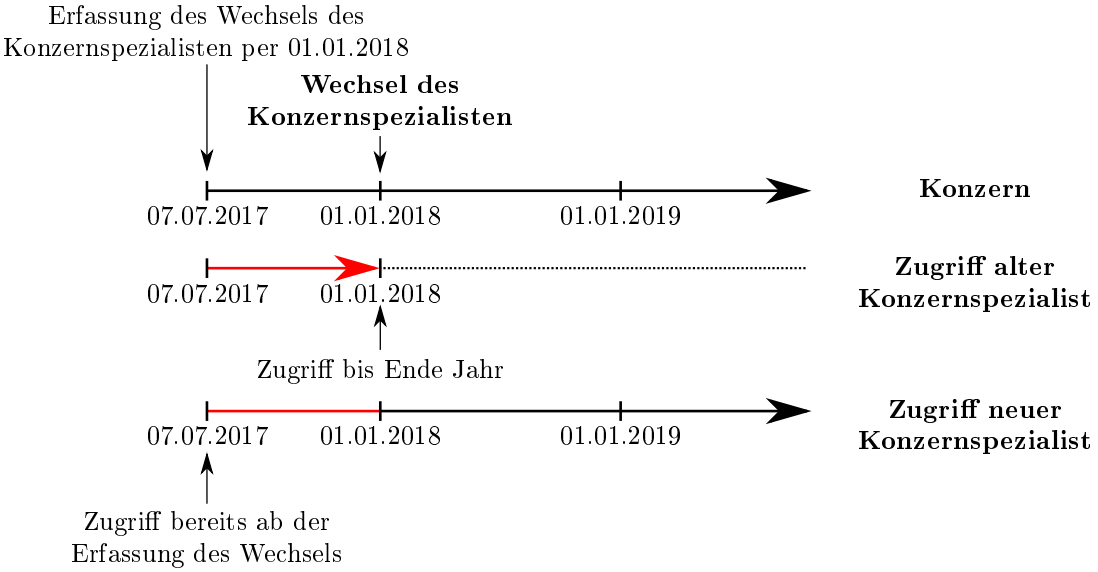

Abbildung 3.4.: Zeitlich überschneidende Berechtigung bei Konzernspezialisten-Wechsel

Akzeptanzkriterien

- Der neue Konzernspezialist erhält den Zugriff auf den Konzern und dessen Mitglieder ab dem Zeitpunkt an welchem der Wechsel in SYRIUS erfasst wurde. Dafür müssen in SYRIUS bei der Bereitstellung der Berechtigungsattribute die zukünftigen States (fachliche Historisierung) berücksichtigt werden.
- Der alte Konzernspezialist behält den Zugriff ebenfalls, bis der Wechsel fachlich gültig wird. Dies ist üblicherweise am Ende eines Kalenderjahres, kann aber auch zu anderen Zeitpunkten erfolgen.

3.2.10. US7: Technischen User für Massenverarbeitungen berechtigen

Als Operating-Verantwortlicher des Versicherers möchte ich, über einen technischen User «BATCH», Massenverarbeitungen über alle Daten in SYRIUS durchführen, damit der operative Aufwand durch die automatisierte Durchführung reduziert werden kann. Dazu muss der User «BATCH» Berechtigungen auf alle Daten in SYRIUS haben, welche er verarbeitet.

Akzeptanzkriterien

- Technischer Benutzer «BATCH» muss Lese- und Schreibrechte auf alle Daten in SYRIUS haben, ausser auf Diagnosedaten.
- Über die Benutzerzugänge von SYRIUS, wie dem Ultra-Thin Client (UTC), darf der technische User nicht verwendet werden.

3.2.11. US8: Diagnosebezogene Daten schützen

Als Krankenversicherer möchte ich, dass diagnosebezogene Daten, Informationen welche als *Sensitive Personal Information (SPI)* kategorisiert werden können und zum Beispiel Krankheitsdiagnosen beinhalten, nicht von Vertriebspartnern oder Mitarbeitern, welche diese Daten nicht für die Erfüllung ihres Auftrages benötigen, eingesehen werden können, damit die Daten nur für Personen sichtbar sind, welche für die ihnen anvertraute Aufgabe zwingend auf diese Daten angewiesen sind (zweckgebundenen Verwendung von Daten; DSG Art. 4 Abs. 3), sodass die diagnosebezogene Daten nicht an die Öffentlichkeit gelangen und Kunden das Vertrauen in ihre Versicherung nicht verlieren.

Unter diagnosebezogene Daten fallen zum Beispiel die diagnostizierten Krankheiten, die Dauer des Spitalaufenthalts, angewendete Behandlungen, und so weiter. Diese Daten sollen nur den zuständigen Sachbearbeitern, welche mit der Leistungsbearbeitung vertraut sind, angezeigt werden. Alle Mitarbeiter welche diese sensiblen Daten nicht für die

Erfüllung ihres Auftrags brauchen, sollen die Daten nicht erhalten.

Folgende BOs in SYRIUS enthalten diagnosebezogene Daten:

- Falldossier und Falldiagnose
- Leistungserfassung (Pflegeabrechnung, Taggeldabrechnung, Rentenabrechnung, Kapitalleistungen)
- Kostengutsprache
- E-Claim-Rechnung (MCD-Informationen: Minimal Clinical Dataset)

Akzeptanzkriterien

- Vertriebspartnern dürfen keine diagnosebezogene Daten angezeigt werden.
- Mitarbeiter, welche berechtigt sind an einem Fall zu arbeiten, dürfen die dazugehörigen Diagnosedaten sehen.

3.2.12. US9: Partner aufgrund ihrer OE schützen

Beschreibung

Als Versicherer möchte ich Partner und Falldossiers sowie deren Subobjekte (Vertrag, Leistungsfall) nur von Mitarbeitern der zuständigen oder einer übergeordneten OE, zum Beispiel dem Firmenhauptsitz des Versicherers, bearbeiten lassen, damit jeder Partner und jedes Falldossier nur von Mitarbeitern verändert werden kann, welche über das nötige OE-spezifische Wissen verfügen und somit kompetente Entscheidungen treffen können.

Akzeptanzkriterien

- Mitarbeiter einer OE haben Rechte auf die Leistungen ihrer OE.
- Mitarbeiter können Rechte über eine oder mehrere OEs erhalten.
- Ein Mitarbeiter einer übergeordneten OE kann die Daten der untergeordneten OEs einsehen und bearbeiten.

3.2.13. US10: Postkörbe und Aufgaben schützen

Beschreibung

Als Versicherer möchte ich einzelne Postkörbe und die darin enthaltenen Aufgaben nur von Mitarbeitern mit einer spezifischen Geschäftsrolle abarbeiten lassen, damit die Aufgaben gezielt von Mitarbeitern mit dem entsprechenden fachlichen Know-How erledigt werden. Damit kann sichergestellt werden, dass Aufgaben spezifischen Geschäftsrollen zugewiesen werden können und nur Mitarbeiter, welche diese Geschäftsrolle besitzen, diese Aufgaben sehen und bearbeiten können.

Akzeptanzkriterien

- Der Zugriff auf Postkörbe und die darin enthaltenen Aufgaben ist nur mit einer definierten Geschäftsrolle möglich.
- Ein Mitarbeiter welcher die Geschäftsrolle nicht besitzt, kann den Postkorb und die dazugehörigen Aufgaben nicht einsehen.

3.2.14. US11: Vertretungen bei Absenzen berechtigen

Beschreibung

Als Versicherer möchte ich einem Mitarbeiter über einen definierten Zeitraum alle Berechtigungen eines abwesenden Mitarbeiters zuteilen, damit der stellvertretende Mitarbeiter die Aufgaben des abwesenden Mitarbeiters über diesen Zeitraum wahrnehmen kann. Es muss sichergestellt werden, dass alle Aufgaben auch während der Abwesenheit einzelner Mitarbeiter von einem Stellvertreter ausgeführt werden können.

Akzeptanzkriterien

- Einem Mitarbeiter kann als Stellvertretung eines anderen Mitarbeiter zugewiesen werden.
- Diese Stellvertretung kann zeitlich limitiert / festgelegt werden (von/bis).
- Der Stellvertreter erhält über den definierten Zeitraum alle Berechtigungen des abwesenden Mitarbeiters.
- Ein Stellvertreter muss nicht weiter vertreten werden (keine mehrstufige Hierarchie).

3.2.15. US12: Leistungserbringer vor Mutation schützen

Beschreibung

Als Versicherer möchte ich die automatisch über eine Schnittstelle eingelesenen Daten über Leistungserbringer in SYRIUS wie zum Beispiel Ärzte, Spitäler, oder sonstige Rechnungssteller nur von einem Team von dafür zuständigen Mitarbeitern innerhalb der Firma bearbeiten lassen, damit nicht jeder beliebige Mitarbeiter die Stammdaten zu Leistungserbringern verändern kann und eine zentrale Stelle für die Änderung an den Daten vorhanden ist. Dadurch wird auch die Möglichkeit von Betrugsfällen, zum Beispiel über die Mutation von Bankverbindungen der Leistungserbringer, reduziert.

Die Daten wie Anschriften, Adressen, und Kontoverbindungen von Leistungserbringern werden in der Schweiz über eine zentrale Stelle regelmässig allen Krankenversicherungen zur Verfügung gestellt. Sollte eine Versicherung eine Information finden, welche nicht mehr aktuell ist, muss diese zentrale Stelle informiert werden. Somit erhalten alle Datenbezüger die aktuellen Daten bei der nächsten Auslieferung. Diese Stammdaten werden typischerweise etwa wöchentlich über einen Batchjob in SYRIUS importiert. Würde jeder Mitarbeiter einer Versicherung Änderungen an diesen Stammdaten vornehmen, würden veraltete Informationen von der zentralen Stelle nie als solche identifiziert und nicht angepasst. Zudem böte diese Mutationsberechtigung auch die Möglichkeit zur Änderung prozesskritischer Daten, wie zum Beispiel der Bankverbindung eines Arztes. Zahlungen könnten durch die Mutation solcher Stammdaten an beliebige Konten umgeleitet werden.

Akzeptanzkriterien

- Alle Mitarbeiter, welche Leistungen bearbeiten, müssen die Leistungserbringer sehen, damit sie wissen ob Rechnungen für Behandlungen oder Schadenfälle akzeptiert und ausbezahlt werden können.
- Die Experten müssen Daten der Leistungserbringer bearbeiten, um im Falle veralteter Daten schnell reagieren zu können.

3.2.16. US13: Vertriebspartner vor Mutation schützen

Beschreibung

Als Versicherer möchte ich Daten zu Vertriebspartnern, welche auch Mitarbeiter des Versicherers sein können, nur von einem Team, welches sich speziell mit den Vertriebspartnern auskennt, bearbeiten lassen, damit keine falschen Provisionen an die Vertriebspartner ausgelöst werden können.

Über die Vertriebspartner wird in SYRIUS auch automatisch die Provision ausgelöst. Deshalb ist es sinnvoll einen Mutationsschutz für dieses Objekt zu implementieren. Gelingt es einem Vertriebspartner einen neuen Kunden zu akquirieren kann sich dies auf seine Provision auswirken. Könnte nun jeder Mitarbeiter einen Vertriebspartner mutieren, würden vielleicht inkorrekte Provisionen ausbezahlt. Noch kritischer wird die Situation wenn der Vertriebspartner als Mitarbeiter des Versicherers arbeitet, und sich somit selbst Provisionen auszahlen könnte. Im Gespräch mit Kunden von Adcubum wurde in Erfahrung gebracht dass dieser Fall bereits aufgetreten ist. Deshalb ist dieser Schutz für die Versicherer von Relevanz.

Akzeptanzkriterien

- Vertriebspartner müssen im System von allen Mitarbeitern gefunden werden.
- Nur eine spezielle Gruppe von zuständigen Mitarbeitern kann die Daten von Vertriebspartnern verändern.

3.2.17. US14: Berechtigungsverwaltung schützen

Beschreibung

Als Versicherer möchte ich, dass die Verteilung von Benutzerrechten möglichst restriktiv geschieht und von einigen wenigen Berechtigungsadministratoren durchgeführt wird, um durchzusetzen dass Benutzer nur die Rechte erhalten welche sie für ihren Auftrag benötigen, damit keine sensiblen Informationen in falsche Hände geraten, keine Möglichkeiten zu Betrug eröffnet werden, und das DSG eingehalten wird.

Die restriktive Vergabe von Rechten und Benutzerrollen in SYRIUS kann beinhalten dass die Verteilung eines Rechts erst von einem Vorgesetzten, oder in höchster Instanz vom Datenschutzbeauftragten der Firma, bewilligt werden muss. Im DSG wird oft von «verhältnismässig» oder «zweckgebunden» im Zusammenhang mit der Verwendung von Informationen gesprochen. Dazu gehört auch eine restriktive Verteilung von Rechten auf den Daten.

Akzeptanzkriterien

- Nur Administratoren mit einer bestimmten Rolle sind in der Lage die Berechtigungen eines Benutzers zu ändern.
- Alle anderen Benutzer können die Berechtigungskonfiguration nicht verändern.

3.2.18. US15: Vorbehalte schützen

Beschreibung

Als Versicherer möchte ich, dass bei einem abgeschlossenen Vertrag allfällige Vorbehalte nur von Sachbearbeitern welche, Leistungen bearbeiten müssen, angezeigt werden, damit diese sehr persönlichen Informationen auf Basis des DSG korrekt behandelt werden und keine unberechtigten Mitarbeiter Zugriff darauf haben.

Vorbehalte werden bei einem neuen Vertragsabschluss einer Zusatzversicherung hinterlegt, wenn aufgrund der Krankheitsgeschichte des neuen Kunden gewisse Einschränkungen bei den Leistungen gemacht werden müssen. Somit schützt sich der Versicherer vor Zahlungen von Leistungen, welche aufgrund der Vorgeschichte des Kunden ein erhöhtes Risiko aufweisen. Diese Vorbehalte geben grossen Aufschluss über die Krankengeschichte eines Versicherten und enthalten Diagnosedaten. Aus diesem Grund sind sie besonders schützenswert.

Akzeptanzkriterien

- Vorbehalte dürfen nur ausgewählten Sachbearbeitern, welche mit der Leistungsbearbeitung vertraut sind angezeigt werden, sofern die Informationen für deren Auftragserfüllung notwendig sind.
- Alle anderen Sachbearbeiter und SYRIUS-Benutzer haben keinen Zugriff auf Vorbehalte.

3.2.19. US16: Leistungen aufgrund ihres Geschäftsbereichs schützen

Beschreibung

Als Versicherer möchte ich, dass Sachbearbeiter, welche auf einen Geschäftsbereich spezialisiert sind und in diesem Bereich tätig sind, nur Leistungen in ihrem Bereich sehen können, damit jeder Leistungsbearbeiter die Leistungen in seinem Spezialgebiet behandelt und jede Aufgabe mit dem grösstmöglichen Wissen im Bezug auf die zu bearbeitende Leistung ausgeführt wird.

Da das Versicherungswesen sehr komplex ist und viele verschiedene Produkte von einem Versicherer angeboten werden, müssen die Leistungsbearbeiter meist auf ein Thema spezialisiert sein. Für jeden dieser Bereiche, wie zum Beispiel Krankenversicherung, Sachversicherung oder Unfallversicherung, werden in SYRIUS üblicherweise Geschäftsbereiche gebildet. Ein Leistungsbearbeiter soll dann nur die Daten seines Geschäftsbereichs sehen. In diesem Bereich verfügen die Mitarbeiter über das nötige Expertenwissen, um

eine fachkundige Einschätzung treffen zu können.

Akzeptanzkriterien

- Ein Leistungsbearbeiter darf Leistungen und Aufgaben seines Fachbereichs sehen.
- Leistungen eines anderen Geschäftsbereichs als desjenigen des Leistungsbearbeiters sind nicht einsehbar.

3.2.20. US17: Betrag von Leistungsauszahlungen limitieren

Beschreibung

Als Versicherer bin ich daran interessiert, dass Mitarbeiter entsprechend ihrer Hierarchiestufe oder Funktion unterschiedliche Limiten um Leistungen auszuzahlen haben, oder ab einem gewissen Betrag eine Bestätigung eines Vorgesetzten eingeholt werden muss, damit der Schaden von zweifelhaften oder falsch ausgelösten Zahlungen minimiert werden kann.

Falsch bewilligte Zahlungen können auch als Betrug angesehen werden, falls der Mitarbeiter vorsätzlich den Versicherer täuschen möchte. Der Schaden aus solchen Fällen soll möglichst klein gehalten werden. Zudem kann auf diese Art ein Kontrollmechanismus umgesetzt werden, um Mitarbeiter, welche noch im Training sind, besser zu betreuen.

Akzeptanzkriterien

- Mitarbeiter können nur Zahlungen bis zu einem gewissen Betrag selbstständig auslösen.
- Die Limite ist abhängig von Funktion und Hierarchiestufe des Mitarbeiters, und kann angepasst werden.

3.2.21. US18: Sensible Bearbeitungsfälle schützen

Beschreibung

Als Versicherer bin ich daran interessiert, dass Bearbeitungsfälle eines gewissen Falltyps, zum Beispiel *Bekämpfung Versicherungsmissbrauch*, welcher beim Erstellen des Falles angegeben wird, nur von bestimmten Experten eingesehen werden können, sodass die sensiblen Informationen dieser Falltypen nur für einen möglichst kleinen Personenkreis einsehbar sind.

Bestimmte Bearbeitungsfälle enthalten besonders vertrauliche Informationen wie zum Beispiel ein vermuteter Fall von Versicherungsmissbrauch. In diesen Fällen muss sichergestellt werden, dass normale Mitarbeiter keinen Zugriff zu diesen Informationen erhalten, da so die Gefahr der Veröffentlichung der Daten geringer ist. Sollten solche vertraulichen Informationen veröffentlicht werden, könnte dies für den Kunden drastische Folgen haben, für welche er dann wiederum die Versicherung verantwortlich macht.

Akzeptanzkriterien

- Normale Mitarbeiter können Bearbeitungsfälle eines gewissen Typs nicht finden oder einsehen.
- Ein spezialisierter Mitarbeiter soll die sensiblen Fälle einsehen und bearbeiten können.

3.2.22. Ausblick

In diesem Kapitel wurden die konkreten User Stories, welche aus den Interviews mit Kunden von Adcubum erarbeitet wurden, aufgeführt. Diese Liste erhebt keinen Anspruch auf die Vollständigkeit der Schutzanforderungen an SYRIUS. Die User Stories werden verwendet um mittels eines Prototypen zu belegen, dass die Schutzanfordungen an SYRIUS mit einem neuen ABAC-basierten Autorisierungssystem umsetzbar sind. Dazu wurden mit den User Stories möglichst viele technisch verschiedene Anwendungsfälle ausgewählt. Im Kapitel 4 wird die Umsetzung der User Stories als ABAC-Policies und der entwickelte Prototyp aufgezeigt.

3.3. Nichtfunktionale Anforderungen (NFA)

Nachdem mit den fachlichen Schutzanforderungen gezeigt wurde, welche Daten in SYRIUS durch eine entsprechende Autorisierung geschützt werden müssen, sind in diesem Abschnitt die nichtfunktionalen Anforderungen an die Software aufgelistet. Adcubum stellt diese Anforderungen, um die Qualität der Autorisierungslösung sicherzustellen.

3.3.1. Performance

Die Performance ist ein wichtiges Qualitätsattribut für die neue Autorisierungslösung. Eine Autorisierungsanfrage muss ähnlich schnell wie in der bestehenden Lösung beantwortet werden können, da die Persistenzschicht von SYRIUS beim Laden der Daten solange blockiert wird, bis eine Antwort des Autorisierungssystems angekommen ist. Die Performance der bestehenden Lösung wird im Verlaufe der Arbeit gemessen. Innerhalb dieser Bachelorarbeit ist es nicht möglich die Performance des neuen Systems im Detail zu analysieren, da dafür die Anbindungen aller externen Systeme implementiert werden müssten. Dies übersteigt den Rahmen der Bachelorarbeit. Nach Absprache mit dem Industriepartner (A. Gfeller, Besprechung vom 28.02.2017) reicht es daher eine Abschätzung der Performance basierend auf der heutigen Lösung zu machen. Aufgrund der in Anhang A durchgeführten Performanceanalyse, darf der «Overhead» bzw. der Performanceverlust für die Autorisierung pro «Service-Call» maximal 24 Prozent der Gesamtzeit ausmachen. Die Details und die Herleitung zu diesem Performancekostendach befindet sich in Anhang A. Des Weiteren wurden im Kapitel 4 unter Abschnitt 4.5 Vorschläge erarbeitet, welche Auskunft darüber geben, an welchen Stellen in der vorgeschlagenen, ABAC-basierten Architektur, Performanceverbesserungen implementiert werden können um diese Performance zu erreichen.

3.3.2. Repräsentativität und Aussagekraft der fachlichen Anforderungen

Eines der Hauptziele dieser Bachelorarbeit ist die Erarbeitung der fachlichen Anforderungen an das Autorisierungssystem. Es ist daher wichtig, dass die Anforderungen repräsentativ und aussagekräftig evaluiert und dargestellt werden. Die fünf häufigsten Anforderungen sowie alle kundenspezifischen Kriterien von drei Kunden sind möglichst vollständig erfasst. Jede User Story muss innert 10 Minuten für einen Consultant verständlich sein, sodass er diese Anforderung in der bestehenden Lösung parametrieren kann. Aufgrund der dokumentierten Anforderungen können die fachlichen Anforderungen in einem Folgeprojekt komplett durch Policies abgebildet werden.

3.3.3. Einfachheit der Policy-Syntax

Die Syntax in welcher die Policies für das Autorisierungssystem erfasst werden, sollen auch für eine Person, welche nicht technisch versiert ist, einfach zu lesen und zu verstehen sein. Schon Heute wird der Objektschutz nicht von Softwareentwicklern konfiguriert, sondern durch Consultants oder Mitarbeiter des Versicherers. Die Policy-Syntax, welche in dieser Bachelorarbeit verwendet wird, soll so einfach gehalten werden, dass eine solche Person eine Policy innert 10 bis 15 Minuten verstehen kann. Des Weiteren sollen diese Personen auch in der Lage sein, eigene Policies innerhalb von 20 Minuten zu verfassen. Dazu wird Kenntnis über die Elemente der Policy-Syntax vorausgesetzt.

3.3.4. Lizenzen für Prototyp

In Bezug auf Thirdparty Libraries dürfen bei Adcubum nicht beliebige Lizenzen verwendet werden. Offene Lizenzen wie zum Beispiel die «Apache Licence» sind erlaubt, GNU General Public License v3.0 (GPL3) hingegen nicht. Es wird an dieser Stelle auf das interne Dokument «Anbindung Fremdbibliotheken» [1] von Adcubum verwiesen, welches die erlaubten Lizenzen auflistet. Bei Unklarheiten wird die Benutzung mit dem Industriepartner und dem Hersteller der Software abgeklärt. Diese Regeln galten bereits für die Vorarbeit [28], und werden auch in dieser Bachelorarbeit so gehandhabt.

3.3.5. Logging / Audit Trail

Bereits im bestehenden Objektschutz wird jede Berechtigungsabfrage und die Entscheidung, ob der Zugriff gewährt wurde oder nicht, geloggt. Auch im neuen System muss jede Berechtigungsabfrage mit folgenden Informationen geloggt werden:

- Identität des Benutzers
- Parameter und deren Werte, welche an die Schnittstelle übergeben wurden
- Entscheidung des Berechtigungssystems: Zugriff gewährt oder nicht

Diese Anforderung konnte aus der Studienarbeit [28] übernommen werden.

3.3.6. Zukunftsicherheit

SYRIUS als Versicherungssoftware hat eine, für Software, extrem lange Lebensdauer. Damit muss auch die Autorisierungslösung wenigstens einen Major-Lebenszyklus überdauern, was mindestens zehn Jahren entspricht. Anstelle von Richtlinien für verwendete Libraries soll der Ansatz verfolgt werden, gegen Schnittstellen anstelle von Implementationen zu programmieren. Somit wird die Abhängigkeit von der gewählten Implemen-

tation reduziert und die Implementation kann einfacher ausgetauscht werden. Bei dem neuen Berechtigungssystem wird zusätzlich darauf geachtet dass verwendete Konzepte, Programmiersprachen, oder Softwarelibraries eine möglichst gute Verbreitung in der IT Branche haben. Somit ist die Chance auf langfristigen Support erhöht. Es dürfen sich aus der neuen Autorisierungssoftware keine zusätzlichen Anforderungen an Technische Berater und Business Consultants ergeben.

3.3.7. Wartbarkeit

Aufgrund der langen Zeit, über welche das Autorisierungssystem verwendet werden wird, ist der Wartbarkeit der Komponente grosse Bedeutung zuzuschreiben. Da Adcubum selbst keinen Betrieb von SYRIUS für Kunden anbietet, geschieht ein Grossteil der Wartung bei den Kunden oder ihren Systemanbietern. Änderungen der Policies sollen demnach innerhalb der üblichen Wartungsfenster möglich sein. Typischerweise ist der Rhytmus dieser Fenster etwa wöchentlich. Wegen dem externen Betrieb von SYRIUS sind auch keine festgelegten «Downtimes» in den Service Level Agreements (SLAs) von Adcubum zu finden. Minor Softwareupdates am Code von SYRIUS können aus Erfahrung «Downtimes» von bis zu zwei Stunden zur Folge haben, die seltenen Major Updates bis zu zwei Tage. In diesen «Downtimes» sollte auch das Autorisierungssystem aktualisiert werden können. Externe Libraries müssen, im Sinne der Wartbarkeit, durch ein Interface gekapselt sein, damit die Implementationen einfach austauschbar sind.

Alle funktionalen sowie nichtfunktionalen Anforderungen sind durch die neue Autorisierungslösung zwingend zu erfüllen. Das folgende Kapitel *Design und Implementation* beschreibt nun den Versuch mit einem Prototyp aufzuzeigen, dass die dokumentierten Anforderungen mit einer ABAC-basierten Lösung umzusetzen sind.

4. Design und Implementation

Nachdem das letzte Kapitel die Anforderungen an die neue Autorisierungslösung erläutert hat, geht dieses Kapitel auf die Dokumentation des während dieser Bachelorarbeit entwickelten Prototypen ein. Die getroffenen Architekturentscheidungen und die für den Prototypen verwendete Policy-Syntax werden in diesem Kapitel beschrieben.

4.1. Design- und Architekturentscheidungen

Dieser Abschnitt hält die wichtigen Architekturentscheidungen, welche im Zusammenhang mit der Entwicklung des Prototypen getroffen wurden, fest.

4.1.1. Übersicht

Folgende Tabelle gibt einen Überblick über die getroffenen Design- und Architekturentscheidungen. Nachfolgend werden diese im Detail erläutert.

Tabelle 4.1.: Übersicht Design- und Architekturentscheidungen

Entscheidung	**Kurzbeschreibung**
Wahl der User Stories für den Prototypen	Für die Implementation des Prototypen mussten einige User Stories ausgewählt werden. Bei der Auswahl wurde versucht, einen möglichst repräsentativen Satz von User Stories zu finden, damit so viele technische Anforderungen wie möglich abgedeckt sind. Ähnliche User Stories wurden nicht mehrfach implementiert.

Tabelle 4.1.: Übersicht Design- und Architekturentscheidungen

Entscheidung	Kurzbeschreibung
Auswahl der Policy-Syntax	Zu einigen User Stories wurden Policies im Sinne von Attribute-Based Access Control (ABAC) definiert. Bei der Auswahl der zu benutzenden Syntax wurde darauf geachtet, dass diese weit verbreitet ist und mehrere Implementationen davon existieren. Das ist ein wichtiges Kriterium, damit durch die Wahl der Syntax nicht zwingend ein bestimmter Policy Decision Point (PDP) verwendet werden muss. Dies wiederum lässt sich mit den nichtfunktionalen Anforderungen (NFAs) begründen. Als zweites Kriterium für die Syntax der Policies wurde die NFA «Einfachheit der Policy-Syntax» genommen, da die Policies später durch Consultants oder Parametrierer gewartet werden.
Evaluation der Open Source Library für den Prototypen	Für den Prototypen wird eine Implementation des PDP benötigt. Bei der Auswahl wurde darauf geachtet, dass das gewählte Produkt den aktuellen XACML 3.0 Standard implementiert, um mit der gewählten Syntax kompatibel zu sein. Zudem soll das Produkt unter einer Lizenz verfügbar sein, welche uns die kostenlose Verwendung innerhalb der Bachelorarbeit erlaubt. Die Entscheidung fiel auf die Implementation von WSO2 [22], Balana[23]. Balana erfüllt als Open Source Software die geforderten Kriterien. Zudem ist die Community auf GitHub [17] sehr aktiv, was an einer durch Stefan Kapferer gestellen Frage und der entsprechenden Antwortzeit bestätigt wurde.

4.1.2. Wahl der User Stories für den Prototypen

Da die Umsetzung aller User Stories im Prototypen den Rahmen dieser Bachelorarbeit überstiegen hätte, mussten einige möglichst repräsentative User Stories ausgewählt werden. Die ausgewählten User Stories sind als Policies im Prototypen umgesetzt. Für die Auswahl der Stories war entscheidend, dass möglichst alle Anforderungen aus technischer Sicht abgedeckt sind. Aus diesem Grund wurden die Stories gruppiert und technisch identisch umsetzbare Stories zu einer Gruppe zusammengefasst.

Die nachfolgende Tabelle zeigt diese Gruppierung der User Stories aus dem Abschnitt 3.2. Im Anschluss an die Tabelle 4.2 werden die einzelnen Gruppen und deren technische Schwierigkeiten beschrieben.

Tabelle 4.2.: Gruppierung der User Stories für Prototyp

Gruppe	**User Stories**
#1: Ein Business Object (BO) aufgrund seines Typs oder anderer Klassifizierung schützen	• US1: Very important Person (VIP) Kunden schützen • US2: Mitarbeiter schützen • US3: Organisationseinheit schützen • US9: Partner aufgrund ihrer Organisationseinheit (OE) schützen • US10: Postkörbe und Aufgaben schützen • US16: Leistungen aufgrund ihres Geschäftsbereichs schützen • US18: Sensible Bearbeitungsfälle schützen
#2: Konzernspezialisten berechtigen	• US4: Konzernspezialist berechtigen
#3: Benutzer aufgrund zukünftiger Zustände (Historisierung) berechtigen	• US5: Konzernspezialist auf zukünftiges Konzernmitglied berechtigen • US6: Alten und neuen Konzernspezialisten bei einem Konzernspezialisten-Wechsel zeitlich überschneidend berechtigen
#4: Technische Benutzer berechtigen	• US7: Technischen User für Massenverarbeitungen berechtigen
#5: Diagnosebezogene Daten schützen	• US8: Diagnosebezogene Daten schützen • US15: Vorbehalte schützen
#6: Berechtigungen temporär übertragen	• US11: Vertretungen bei Absenzen berechtigen
#7: Für das System kritische Daten vor Mutation schützen	• US12: Leistungserbringer vor Mutation schützen • US13: Vertriebspartner vor Mutation schützen • US14: Berechtigungsverwaltung schützen • US17: Betrag von Leistungsauszahlungen limitieren

Aus der Tabelle 4.2 gehen die Gruppen hervor, welche aufgrund der technischen Ähnlichkeit der User Stories erstellt wurden. Diese Gruppen werden nun kurz beschrieben.

Gruppe #1: Ein BO aufgrund seines Typs oder anderer Klassifizierung schützen

Bei diesen User Stories geht es darum, ein BO und bestimmte referenzierte BOs, aufgrund einer Klassifizierung oder einer bestimmten Ausprägung eines Attributs zu schützen. Die

Policies für alle diese Fälle unterscheiden sich lediglich in der entsprechenden Bedingung, wie zum Beispiel ob der Partner VIP, Mitarbeiter oder OE ist. User Stories aus dieser Gruppe sind in der Praxis die häufigste Anwendung, was sich auch in den Interviews mit den Kunden herausstellte. Diese User Stories eignen sich um Konzepte wie den jetzigen Schutzpfad zu modellieren.

Gruppe #2: Konzernspezialisten berechtigen

Die Partner-Daten werden häufig aufgrund einer bestimmten Organisationsstruktur geschützt. Dies wird in der Regel mittels einer OE-Zugehörigkeit des Partners gelöst (siehe US9). Der Konzernspezialist ist aus dieser Perspektive ein Spezialfall, weil er Zugriff auf die Daten von Partnern unabhängig dieser Organisationsstruktur erhalten muss. Dies beruht auf der Tatsache, dass Konzernmitglieder verschiedenen OEs angehören können.

Gruppe #3: Benutzer aufgrund zukünftiger Zustände (Historisierung) berechtigen

Beim Hinzukommen von Konzernmitgliedern zu einem Konzern oder beim Wechsel der OE eines Konzernmitgliedes kommt es zu der Situation, dass ein Konzernspezialist Zugriff auf die Daten eines Partners erhalten muss, obwohl die Zuordnung fachlich gesehen erst in der Zukunft gültig wird. Diese zeitliche Überschneidung wird nur in den User Stories US5 und US6 behandelt, was auch die technische Schwierigkeit dieser Gruppe ist. Allerdings kann mit einem guten Attribute Engineering ein Teil Schwierigkeit der Historisierung in den Policy Information Point (PIP) ausgelagert werden.

Gruppe #4: Technische Benutzer berechtigen

Technische Benutzer müssen übergreifend aller anderen Policies berechtigt werden können. Das System muss es ermöglichen einem solchen Benutzer Zugriff auf alle Daten zu gewähren, auch wenn dies aufgrund anderer Policies nicht erlaubt wäre. Dieser Spezialfall muss separat betrachtet werden. Aus technischer Sicht müssen sich mehrere Policies und Regeln überschneiden können.

Gruppe #5: Diagnosebezogene Daten schützen

Bei den User Stories dieser Gruppe geht es um den Schutz spezieller Daten in SYRIUS, welche im Zusammenhang mit einer Diagnose stehen. Ein grosser Teil des Aufwands für die Umsetzung der User Story US8 und US15 ist, die über viele verschiedene BOs in SYRIUS verteilten Diagnosedaten zu sammeln und in einer Policy festzuhalten. Technisch birgt diese Gruppe keine Risiken, weshalb andere Gruppen eine höhere Priorität geniessen.

Gruppe #6: Berechtigungen temporär übertragen

Die temporäre Übertragung von Berechtigungen an einen anderen Benutzer, zum Beispiel bei einer Ferienvertretung, ist ein Spezialfall und muss separat betrachtet werden. Zeitliche Begrenzungen sowie die Weitergabe von Rechten einer bestimmten Person sind hier als technisches Risiko zu nennen.

Gruppe #7: Für das System kritische Daten vor Mutation schützen

Bei den Stories dieser Gruppe geht es darum, für das System prozesskritische Daten nur von bestimmten Personen bearbeiten zu lassen. Die einzelnen Stories unterscheiden sich lediglich durch die unterschiedlichen BOs und Attribute, welche geschützt werden sollen, und können daher zu einer Gruppe zusammengefasst werden. Aus der Sicht der Autorisierungsschnittstelle, welche in der Studienarbeit [28] definiert wurde, muss hier nur die Operation «WRITE» geschützt werden.

Implementierte User Stories

Aufgrund der obigen Gruppierung aus Tabelle 4.2 kann nun eine Auswahl der im Prototyp zu implementierenden User Stories getroffen werden. Welche konkrete Story innerhalb der Gruppe ausgewählt wird, ist dabei unerheblich.

Die untenstehende Tabelle 4.3 hält pro Gruppe jeweils eine bis zwei User Stories fest, welche technisch gesehen für die gesamte Gruppe repräsentativ sind. Sind diese User Stories umgesetzt, lassen sich die übrigen User Stories einer Gruppe nach dem gleichen Schema implementieren. Zusätzlich ist jede Gruppe mit einer Priorität von 1 (höchste Dringlichkeit) bis 7 (geringste Dringlichkeit) versehen. Aufgrund dieser Priorität wird entschieden, in welcher Reihenfolge die Policies implementiert werden. Die Priorität berücksichtigt die Verbreitung der User Stories bei den Kunden, und wie viele technische Spezialfälle dadurch abgedeckt sind. Im Rahmen der Bachelorarbeit werden Stories aus den vier am höchsten priorisierten Gruppen umgesetzt.

Tabelle 4.3.: Für eine Gruppe aus Tabelle 4.2 repräsentative User Stories

Gruppe	**Repräsentierende Stories**	**Priorität**
#**1:** Ein BO aufgrund seines Typs oder anderer Klassifizierung schützen	• US1: VIP Kunden schützen • US2: Mitarbeiter schützen	**1**
#**2:** Konzernspezialisten berechtigen	• US4: Konzernspezialist berechtigen	**7**

Tabelle 4.3.: Für eine Gruppe aus Tabelle 4.2 repräsentative User Stories

Gruppe	Repräsentierende Stories	Priorität
#**3:** Benutzer aufgrund zukünftiger Zustände (Historisierung) berechtigen	• US6: Alten und neuen Konzernspezialisten bei einem Konzernspezialisten-Wechsel zeitlich überschneidend berechtigen	**6**
#**4:** Technische Benutzer berechtigen	• US7: Technischen User für Massenverarbeitungen berechtigen	**2**
#**5:** Diagnosebezogene Daten schützen	• US8: Diagnosebezogene Daten schützen	**5**
#**6:** Berechtigungen temporär übertragen	• US11: Vertretungen bei Absenzen berechtigen	**3**
#**7:** Für das System kritische Daten vor Mutation schützen	• US12: Leistungserbringer vor Mutation schützen	**4**

Die zur Implementierung ausgewählten User Stories sind:

- US1: VIP Kunden schützen
- US2: Mitarbeiter schützen
- US7: Technischen User für Massenverarbeitungen berechtigen
- US11: Vertretungen bei Absenzen berechtigen
- US12: Leistungserbringer vor Mutation schützen

Die fünf implementierten User Stories aus den vier Gruppen mit der höchsten Priorität belegen die technische Machbarkeit der ABAC-Lösung. Ein Grossteil der technischen Risiken kann mit dieser Auswahl minimiert werden. Bei den User Stories aus Gruppe #1 wurde bewiesen, dass es möglich ist den Schutzpfad aus der alten Lösung in SYRIUS mit ABAC umzusetzen. Anhand der User Story US7 wird aufgezeigt, dass mehrere Policies auf identische Autorisierungsanfragen wirken können, und eine übergeordnete Policy für technische Benutzer implementiert werden kann. US11 beschäftigt sich mit zeitlich begrenzten Berechtigungen und zeigt auf, dass es möglich ist temporäre Berechtigungen für einzelne Benutzer zu erstellen und dabei Attribute der Kategorie *Environment* zu verwenden. Mit dem Mutationsschutz aus Gruppe #7 wird ein weiterer sehr häufig verwendeter Schutz implementiert.

Die Schwierigkeit bei der Implementation des Schutzes von diagnosebezogene Daten, ist das Sammeln aller Objekte und Attribute, welche unter die Diagnosedaten fallen. Beide User Stories, welche sich mit dem Konzernspezialisten beschäftigen, sind nur bei einzelnen Kunden im Einsatz. Deshalb sind sie niedriger priorisiert. Zudem besteht die Schwierig-

keit hier mehr darin, die richtigen Attribute und die Historisierung im PIP abzubilden als in der Entwicklung der passenden Policy. Da die PIPs im Rahmen dieser Bachelorarbeit nur als «Mock»-Implementation bereitgestellt werden, wird auf die Implementation dieser User Stories verzichtet.

4.1.3. Auswahl der Policy-Syntax

ABAC beschreibt lediglich ein Konzept, und legt keine Implementation oder Syntax der zu schreibenden Policies fest. Es gibt mehrere verschiedene Syntaxen oder Konzepte für die Implementation des ABAC-Paradigmas. In der Tabelle 4.4 sind Policy-Syntaxen beschrieben, welche für die Implementierung des ABAC-Paradigmas verwendet werden könnten. Bei der Auswahl der Syntaxen wurden die gängigsten Open-Source Syntaxen berücksichtigt, welche das Projektteam im Internet finden konnte.

Tabelle 4.4.: Mögliche Policy-Syntaxen

Bezeichnung	Vorteile	Nachteile
XACML	• Sehr weit verbreitet • Kann durch zusätzliche DSL vereinfacht werden • Viele PDP-Implementationen	• XML ist sehr verbos • Vergleichbar lange Ladezeiten durch grosse Policy-Files
ALFA	• Kompatibel mit XACML • Sehr einfach lesbar für Menschen • Guter Support und aktive Wartung durch Axiomatics [5]	• Plugin für Generierung von XACML nicht Open Source • Abhängigkeit von Hersteller Axiomatics [5]
RestACL[20]	• Harmoniert mit der Representational State Transfer (REST)-Schnittstelle aus der Studienarbeit [28] • JavaScript Object Notation (JSON) ist weit verbreitet • Bessere Lesbarkeit als XACML	• Keine Implementation eines PDP vorhanden • Support und Community sehr klein
SecPAL[34]	• Syntax sehr kompakt und gut lesbar	• Wird nicht mehr gewartet • Verbreitung sehr gering
SOUTEI[57]	• Ansatz der funktionalen Programmierung aus Prolog [46]	• Verbreitung sehr gering • Wird nicht weiterentwickelt oder umgesetzt

XACML

XACML ist eine auf XML basierte Syntax um Policies auszudrücken, und wird von Organization for the Advancement of Structured Information Standards (OASIS) [39] unterhalten. Der XACML 3.0 Standard [56] ist frei verfügbar und somit nicht von einem Hersteller abhängig. XACML ist im Gebiet von ABAC so weit verbreitet, dass es «de facto» als Standard angesehen werden darf. Dies ist auch der Grund, weshalb sehr viele PDP-Implementationen mit XACML-Policies funktionieren. Da die Policies in XML verfasst werden, müssen auch die bekannten Nachteile von XML akzeptiert werden. Dazu zählt vor Allem die Verbosität von XML, was diese Syntax aber für Maschinen sehr gut lesbar macht.

ALFA

Abbreviated Language For Authorization (ALFA) ist eine von Axiomatics [5] entwickelte DSL, welche die Verbosität der auf XML basierten XACML-Syntax versteckt und die Policies somit auch für Menschen besser lesbar macht. Laut Angaben des Herstellers [4] unterstützt ALFA den neusten XACML 3.0 [56] Standard. Allerdings stellte sich während der Bachelorarbeit heraus, dass die in XACML 3.0 definierten Delegation-Policies nicht umgesetzt sind. Ein Beispiel einer ALFA Policy ist in Abschnitt 4.3.5 zu finden. Mit Hilfe des *ALFA Plugin for Eclipse* [6] kann aus einer ALFA Policy automatisch XACML generiert werden. Dies ist definitiv ein Vorteil, da somit alle PDP-Implementationen welche mit XACML umgehen, auch mit ALFA verwendet werden können. Der ALFA-Standard wird inzwischen von OASIS verwaltet, und ist öffentlich. Das Eclipse-Plugin [6] von Axiomatics [5], welches den Code von ALFA zu XACML übersetzt, ist allerdings für eine kommerzielle Nutzung kostenpflichtig und würde eine Abhängigkeit zu dem Hersteller mit sich bringen. Eine Alternative zu der Verwendung des Plugins von Axiomatics [6] wäre die Entwicklung eines eigenen ALFA-XACML Übersetzers.

RestACL

Access Control Language for RESTful Services (RestACL) wurde an der Universität Furtwangen in Form eines Papers [20] veröffentlicht. Wie der Name bereits vermuten lässt, ist RestACL für RESTful Services angedacht. RestACL ist speziell dafür ausgelegt, REST-Ressourcen anhand von Policies zu berechtigen. Dabei werden die Policies, sowie die Requests und Responses in JSON geschrieben. Der grösste Unterschied zu XACML ist, dass Policies in RestACL auf REST Ressourcen angewendet werden, während XACML durch frei definierbare *Targets* bestimmt, welche Policies bei einem Request evaluiert werden. Trotz der in der Studienarbeit [28] entwickelten REST-Schnittstelle, ist es im Kontext von SYRIUS nicht die Idee einzelne REST-Ressourcen zu schützen, sondern die zu ladenden BOs. Dies würde für Policies sprechen, welche durch technologieunabhängige und dynamische *Targets* eingeschränkt werden, wie sie in XACML verwendet werden. Zudem ist

dem Projektteam keine konkrete Implementation des RestACL-Ansatzes bekannt. Somit müsste ein PDP von Grund auf neu implementiert werden. Bei der Produktevaluation gäbe es keine wirkliche Auswahl.

SecPAL

Security Policy Assertion Language (SecPAL) [34] ist eine deklarative *Security Language*, welche von Microsoft entwickelt wurde. SecPAL wurde so entwickelt, dass die Sprache möglichst nah an der natürlichen Sprache liegt, und so besonders gut lesbar ist. Neben dem Nachteil sich an einen Anbieter (Microsoft) zu binden, ist vor Allem die Tatsache, dass bei SecPAL die Entwicklung nicht mehr vorangetrieben wird, nicht zu vernachlässigen. Die letzten Einträge auf der Homepage von Microsoft [24] datieren aus dem Jahr 2009. Die Projektwebsites von Microsoft [35] zu SecPAL sind zum Teil bereits nicht mehr auffindbar.

SOUTEI

SOUTEI [49] wurde als eine Ausprägung von *Binder* [10] konzipiert. Binder wiederum ist eine Erweiterung der logischen Programmiersprache Datalog, die von Prolog [46] abstammt. Abgesehen von dem Paper von Andrew Pimlott und Oleg Kiselyov [49] lassen sich aber fast keine Informationen zu SOUTEI finden. Auf Github [17] ist eine Implementation von Michael Stone [57] veröffentlicht, welche aber seit fünf Jahren nicht mehr gewartet wird. Dies alles sind Zeichen dafür, dass sich SOUTEI nicht durchsetzen konnte, und deshalb praktisch nicht verbreitet ist.

Entscheidung

Von den in Tabelle 4.4 beschriebenen existierenden Ansätzen wurde aufgrund der NFAs in Abschnitt 3.3 die Syntax ausgewählt, welche die Anforderungen am besten abdeckt. Aufgrund der Zukunftssicherheit (siehe Abschnitt 3.3.6) mussten die Optionen SOUTEI und SecPAL verworfen werden, da beide Ansätze schon seit mehreren Jahren von den Autoren nicht mehr weiterverfolgt wurden. Bei ALFA sprach die Lizenz des Eclipse-Plugins bei einer kommerziellen Nutzung und die Bindung an einen Hersteller in Anbetracht des weiteren Projektverlaufes bei Adcubum gegen eine Wahl dieser Syntax. Im Rahmen der Bachelorarbeit konnte ALFA für die erleichterte Erstellung der Policies, welche in Abschnitt 4.3 erklärt sind, verwendet werden, da aus ALFA-Policies wie bereits erwähnt XACML generiert werden kann. Dieser Einsatz von ALFA während der Bachelorarbeit wird in Abschnitt 4.3.5 erläutert.

Die Entscheidung fiel schlussendlich auf XACML, da während der Recherche deutlich mehr Erwähnungen und Implementationen gefunden wurden, als bei anderen Syntaxen.

Die weite Verbreitung von XACML bringt den Vorteil mit sich, dass bei der Wahl des PDP, welche in Abschnitt 4.1.4 erläutert ist, auf eine grosse Auswahl verschiedener Open Source Produkte zurückgegriffen werden kann. Durch die Auswahl von XACML als Policy-Syntax bleibt die Möglichkeit bestehen, eine DSL, wie zum Beispiel ALFA, zu verwenden um das Schreiben von XML zu umgehen. Um mit dem Prototypen die technische Machbarkeit des Projekts aufzuzeigen ist es noch nicht notwendig eine solche DSL zu verwenden.

Diese Entscheidung lässt sich folgendermassen gemäss der Struktur des *Y-Template* [63] von Prof. Olaf Zimmermann zusammenfassen: «Bei der Auswahl der Policy-Syntax für eine ABAC-basierte Autorisierungskomponente für SYRIUS, mit den Anforderungen an langen Produktsupport und eine einfache Form der Policies, fiel die Entscheidung auf XACML, und gegen ALFA, RestACL, SecPAL, und SOUTEI, da durch die grosse Verbreitung von XACML die Zukunftssicherheit der Syntax am besten gewährleistet ist, wobei als Konsequenz die Verbosität von XML in Kauf genommen wurde.»

4.1.4. Evaluation der PDP Library für den Prototypen

Im Rahmen der Bachelorarbeit wurde ein Prototyp des geplanten Autorisierungssystems entwickelt. Dieser Prototyp soll Autorisierungsanfragen aufgrund der erarbeiteten XACML-Policies mit «PERMIT» oder «DENY» beantworten, also die Aufgabe eines PDP übernehmen. Für die Implementation des PDP soll dabei auf eine bestehende Open Source Software zurückgegriffen werden, da die Entwicklung eines eigenen PDP innerhalb der Bachelorarbeit vom Aufwand her nicht realistisch ist. Die Entscheidung des verwendeten PDP ist in diesem Abschnitt begründet.

Auf folgende Punkte wurde bei den evaluierten Implementationen geachtet:

- **XACML Standard**: Die neuste Version des XACML Standard ist die Version 3.0. Im Sinne der NFA Zukunftssicherheit muss der ausgewählte PDP XACML 3.0 implementieren.
- **Lizenz**: Die Lizenz muss den Richtlinien aus dem Dokument «Anbindung Fremdbibliotheken» der Adcubum [1] entsprechen.
- **Aktualität**: Als Kriterium ob der PDP aktuell ist, wird der letzte Commit auf dem Code-Repository verwendet.

In Tabelle 4.5 werden die evaluierten Implementationen eines XACML-PDP aufgezeigt. Die Auflistung beinhaltet dabei nicht alle verfügbaren PDP-Implementationen, sondern nur die während der Bachelorarbeit evaluierten Optionen.

Tabelle 4.5.: Mögliche PDP Implementationen

Bezeichnung	XACML Version	Lizenz	Letzte Änderung
SNE-XACML	3.0	LGPLv3++ [13]	03.2016
AuthZForce	3.0	GNU GPLv3 [12]	02.2017
XEngine	2.0	GNU GPLv3 [12]	12.2008
Sun XACML	2.0	Sun Microsystems [58]	08.2009
HerasAF	2.0	Apache 2.0 [15]	09.2016
Balana	3.0	Apache 2.0 [15]	02.2017
Axiomatics Policy Server	3.0	Kommerziell	Unbekannt
OpenAZ	3.0	Apache 2.0 [15]	08.2016
PicketBox XACML	2.0	Apache 2.0 [15]	05.2010

Einhaltung der «MUST»-Kriterien

Nicht alle aufgelisteten PDPs halten den für diese Bachelorarbeit definierten NFAs stand. Wie bereits erwähnt, soll der PDP im Sinne der Aktualität und der Zukunftssicherheit den neusten XACML-Standard 3.0 implementieren. Diese Anforderung ist für den Prototypen der Bachelorarbeit ein «MUST»-Kriterium, welches PDPs ohne XACML 3.0 Unterstützung nicht in Frage kommen lassen. Ebenfalls ein «MUST»-Kriterium ist die Lizenz der Software. In den Standards von Adcubum zur Verwendung von *Third-Party Libraries* [1] ist ganz klar vorgegeben, welche Lizenzen verwendet werden dürfen und welche nicht. So ist zum Beispiel, wie in Abschnitt 3.3.4 bereits beschrieben, die Apache 2.0 Lizenz [15] erlaubt, hingegen ist die Verwendung von Software welche unter der GNU General Public License [12] veröffentlicht wird nicht erlaubt. Ebenso kommen kommerzielle Produkte im Rahmen dieser Bachelorarbeit nicht in Frage. Diese Anforderungen schliessen bereits viele der in Tabelle 4.5 erwähnten Implementationen aus.

Analyse der möglichen Implementationen

Nach der Überprüfung auf die «MUST»-Kriterien aus dem letzten Abschnitt 4.1.4 bleiben noch die PDP-Implementationen aus Tabelle 4.6 zur Auswahl. In diesem Abschnitt werden diese Optionen detaillierter analysiert.

Tabelle 4.6.: Mögliche PDP-Implementationen

Bezeichnung	Beschreibung
Balana	• Entwicklung durch WSO2 [22] • Implementation in Java • Community beantwortet Anfragen aktiv • Reiner PDP mit offen einsehbarem Source Code • Vergleichsweise gut dokumentiert
OpenAZ	• Entwickelt als Apache Incubator Projekt[14] • Implementation in Java • Enthält auch einen Policy Administration Point (PAP) sowie Projekte zum Testen der Software • Projektstatus wurde im August 2016 auf «retired» gesetzt

BALANA

Balana [23] ist eine Java-Implementation eines PDP und wird von der Firma WSO2 [22] entwickelt. Balana basiert auf der XACML-Implementation von Sun Microsystems [59], unterstützt aber den neusten XACML-Standard 3.0. Auf dem GitHub [17] Repository von Balana [23] ist ersichtlich, dass die Software weiterhin aktiv gewartet wird. So wurde auch auf einen während dieser Bachelorarbeit erfassten Issue [27] sehr schnell geantwortet. Dies ist definitiv als Vorteil zu gewichten. Zudem finden sich auf der Seite von xacmlinfo.org [61] einige Posts und Blogeinträge zu Balana [23]. Dies ist mehr Dokumentation als bei vielen anderen Implementationen vorhanden ist.

OpenAZ

OpenAZ wird oder wurde als Apache Incubator Projekt [14] entwickelt. Neben einem PDP gehören zu diesem Produkt auch andere Komponenten aus der ABAC-Referenzarchitektur wie zum Beispiel ein Policy Enforcement Point (PEP) oder ein PAP. Dadurch könnten viele Komponenten aus einer Hand verwendet werden, was durchaus ein Vorteil wäre. Allerdings wurden die letzten Änderungen an dem Sourcecode Repository [18] im August 2016 vorgenommen. Seitdem ist der Status des Projektes auch auf der offiziellen Website von Apache Incubator [14] auf «retired» gesetzt. Deshalb ist die Zukunft dieses Projekts sehr fraglich, was gegen eine Verwendung im Umfeld von SYRIUS spricht.

Zusammenfassung und Entscheidung

Beide Implementationen, welche die in Abschnitt 4.1.4 erwähnten Kriterien erfüllen, haben eigene Vor- und Nachteile. Die Dokumentation ist bei Balana umfangreicher als bei OpenAZ, was sicher ein Vorteil für Balana ist. Hingegen kann OpenAZ [18] mit weiteren ABAC-Komponenten wie einem PAP oder einem PEP punkten. Diese Komponenten harmonieren gut miteinander da sie dem gleichen Projekt angehören. Ein gewichtiger Nachteil bei OpenAZ [18] ist der Projektstatus «retired» seit 2016. Dadurch stellt sich die Frage ob bei diesem Projekt die Zukunftssicherheit gemäss den NFAs aus Abschnitt 3.3 gegeben ist. Schlussendlich fiel die Entscheidung zugunsten von Balana [23] aus.

Die Entscheidung lässt sich folgendermassen in einem Satz nach dem *Y-Template* [63] beschreiben: «Bei der Auswahl der PDP-Implementation für den Prototyp dieser Bachelorarbeit, mit den Anforderungen an XACML 3.0 Support, langen Produktsupport und Lizenzen welche mit den Richtlinien von Adcubum vertretbar sind, fiel die Entscheidung auf Balana und gegen OpenAZ [18], da durch die aktive Weiterentwicklung an Balana [23] die Zukunftssicherheit der Implementation besser gewährleistet ist und die Apache 2.0 Lizenz [15] eine Verwendung bei Adcubum zulässt.»

Nachdem nun alle wichtigen Architekturentscheidungen dokumentiert sind, befasst sich der folgende Abschnitt mit der Architektur und der Umsetzung des während der Bachelorarbeit entwickelten Prototypen. Darin werden die in diesem Kapitel getroffenen Entscheidungen umgesetzt.

4.2. Architektur Prototyp

Der während dieser Bachelorarbeit entwickelte Prototyp setzt die Anforderungen und getroffenen Architekturentscheidungen aus den vorherigen Kapiteln um. Dieser Abschnitt geht genauer auf die Architektur dieses Prototyps ein.

4.2.1. Ausgangslage

Während der Studienarbeit [28] wurde bereits ein Prototyp der Autorisierungsschnittstelle entwickelt und ein Refactoring in SYRIUS vorgenommen, um diese Schnittstelle aufzurufen. Da der Prototyp dieser Bachelorarbeit auf demjenigen der Studienarbeit [28] aufbaut, wird das API-Modell der Schnittstelle an dieser Stelle nochmals vorgestellt.

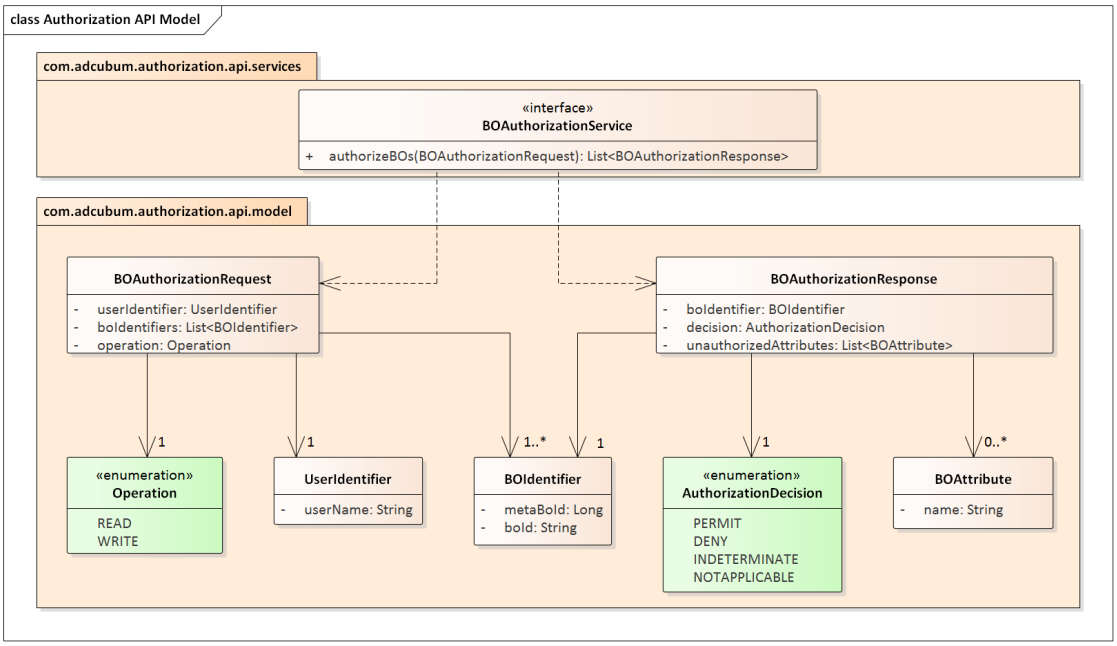

Abbildung 4.1.: API-Modell Autorisierungsschnittstelle (UML Klassendiagramm) [28]

Die RESTful HTTP basierte Autorisierungsschnittstelle implementiert das *BOAuthorizationService* Interface, welches eine Methode zur Autorisierung mehrerer BOs bereitstellt. Der Methode wird ein *BOAuthorizationRequest* als Parameter übergeben, welcher einen *UserIdentifier*, mehrere *BOIdentifier* und die *Operation* enthält. Eine Autorisierungsanfrage gilt folglich immer für einen bestimmten Benutzer und legt fest auf welche BOs, welche Operation autorisiert werden soll.

```
{
  "userIdentifier": {
    "username": "skapferer"
  },
  "boIdentifiers": [
    {
      "metaBoId": -3,
      "boId": "1234"
    },
    {
      "metaBoId": -3,
      "boId": "5678"
    }
  ],
  "operation": "READ"
}
```

Auflistung 4.1: Beispiel: BOAuthorizationRequest (JSON) [28]

Auflistung 4.1 zeigt ein Beispiel eines solchen *BOAuthorizationRequest* in der JSON-Syntax. In diesem Beispiel möchte der Benutzer «skapferer» auf zwei bestimmten BOs die *Operation* «READ» ausführen.

Auf einen *BOAuthorizationRequest* antwortet der *BOAuthorizationService* mit einer Liste von *BOAuthorizationResponse*-Objekten. Jede *BOAuthorizationResponse* enthält die Autorisierungsentscheidung (*Decision*) für ein bestimmtes BO. Die Entscheidung ist in der Regel «PERMIT» oder «DENY». Des Weiteren kann die *BOAuthorizationResponse* im Falle einer «PERMIT»-Entscheidung, eine Liste von Attributen (*unauthorizedAttributes*) enthalten, auf welche der Benutzer trotz des «PERMIT» keinen Zugriff erhalten soll (Attributschutz).

```
[
  {
    "boIdentifier": {
      "metaBoId": -3,
      "boId": "1234"
    },
    "decision": "DENY"
  },
  {
    "boIdentifier": {
      "metaBoId": -3,
      "boId": "5678"
    },
    "decision": "PERMIT",
    "unauthorized-attributes": [
      { "name": "Geburtsdatum" }
    ]
  }
]
```

Auflistung 4.2: Beispiel: BOAuthorizationResponse (JSON) [28]

Auflistung 4.2 zeigt eine mögliche *BOAuthorizationResponse*, welche auf den Beispielrequest aus Auflistung 4.1 folgen könnte. Der Benutzer erhält auf eines der BOs gar keinen und auf das Zweite einen eingeschränkten Zugriff. Der Benutzer darf auf dem zweiten BO das Attribut «Geburtstag» nicht sehen.

Für das *BOAuthorizationService*-Interface wurde während der Studienarbeit [28] lediglich eine Mock-Implementation bereitgestellt, da der Fokus auf der aufzurufenden Schnittstelle war. Die nächsten Abschnitte gehen nun auf die Weiterentwicklung des Prototypen innerhalb der Bachelorarbeit ein.

4.2.2. Übersicht Komponenten

Das Ziel der Weiterentwicklung des Prototypen innerhalb dieser Bachelorarbeit ist es zu zeigen, dass die Anforderungen aus Kapitel 3 mittels eines attributbasierten Autorisierungssystems umsetzbar sind. Dabei wird unter anderem eine neue Implementation des *BOAuthorizationService*-Interfaces aus der Studienarbeit [28] erstellt. Anstatt die Autorisierungsentscheidungen aufgrund statischer «Mock»-Daten zu treffen, basiert die Entscheidungsfindung des neuen Prototypen auf XACML-Policies. Als Implementation des PDP wird Balana [23] von WSO2 [22] verwendet, wie in Abschnitt 4.1.4 bereits im Detail erläutert wurde.

Im Kapitel 2 unter Abschnitt 2.4 wurde die ABAC-Referenzarchitektur bereits vorgestellt. Abbildung 4.2 zeigt diese Architektur nochmals im Kontext von SYRIUS und dem Prototypen dieser Bachelorarbeit. Diese Abbildung spiegelt die angestrebte Architektur wieder. Allerdings gehört die Anbindung der Fremdsysteme nicht zum Scope dieser Bachelorarbeit.

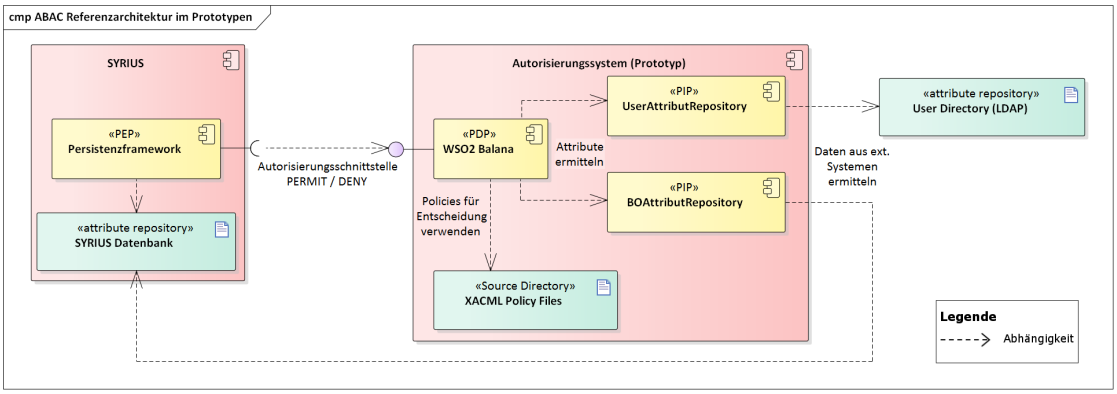

Abbildung 4.2.: ABAC-Referenzarchitekur im Prototypen (UML Komponentendiag.)

Daher wurden die PIPs, SYRIUS und das Lightweight Directory Access Protocol (LDAP)-Benutzerverzeichnis, welches das in RFC 4511 [54] beschriebene LDAP-Protokoll verwendet, im Prototypen durch «Mock»-Implementationen ersetzt. Wie Abbildung 4.3

zeigt, wurden das *UserAttributRepository* und das *BOAttributRepository* «gemocked», anstatt die Systeme tatsächlich anzubinden. Unter «Mock»-Implementationen sind in diesem Fall Java-Klassen zu verstehen, welche die für unsere Testfälle benötigten Attribute statisch im Code hinterlegt haben und zurückliefern.

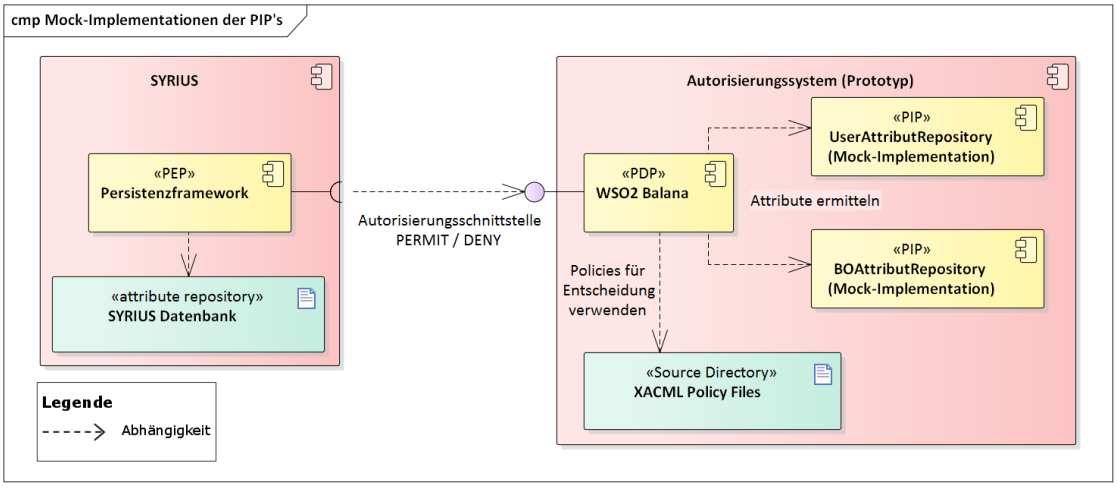

Abbildung 4.3.: «Mock»-Implementationen der PIPs (UML Komponentendiagramm)

Die XACML-Policy-Dateien wurden direkt im Java-Projekt als Ressourcen abgelegt. Ein PAP, im Sinne eines Tools mit einem Graphical User Interface (GUI) zur Verwaltung der Policies, wurde nicht implementiert. Da ein PAP lediglich die Verwaltung der Policies vereinfacht, hat er auf die Umsetzbarkeit des ABAC-basierten Konzepts keinen Einfluss.

4.2.3. Logische Sicht

Nachfolgend wird der Aufbau des Prototyps, anhand dessen wichtigster Interfaces und Klassen, aus logischer Sicht erläutert (siehe Abbildung 4.4).

Der *BalanaBOAuthorizer* stellt die neue Implementation des *BOAuthorizationService* dar und verwendet den Balana-PDP um die BOs autorisieren zu können. Dazu benötigt er einerseits die XACML-Policy-Dateien und andererseits die Attributwerte der angefragten BOs und des Benutzers. Dafür stellt Balana [23] sogenannte *AttributeFinderModules* zur Verfügung, die auch selbst implementiert werden können um eigene PIPs bereitzustellen. Dazu wird die abstrakte Klasse *AttributeFinderModule* erweitert. Wie aus Abbildung 4.4 hervorgeht, benötigt der *BalanaBOAuthorizer* zwei *AttributeFinderModules* namens «BOAttributeProvider» und «SyrUserAttributeProvider», welche jeweils die Attributwerte der BOs und SYRIUS-Benutzer ermitteln.

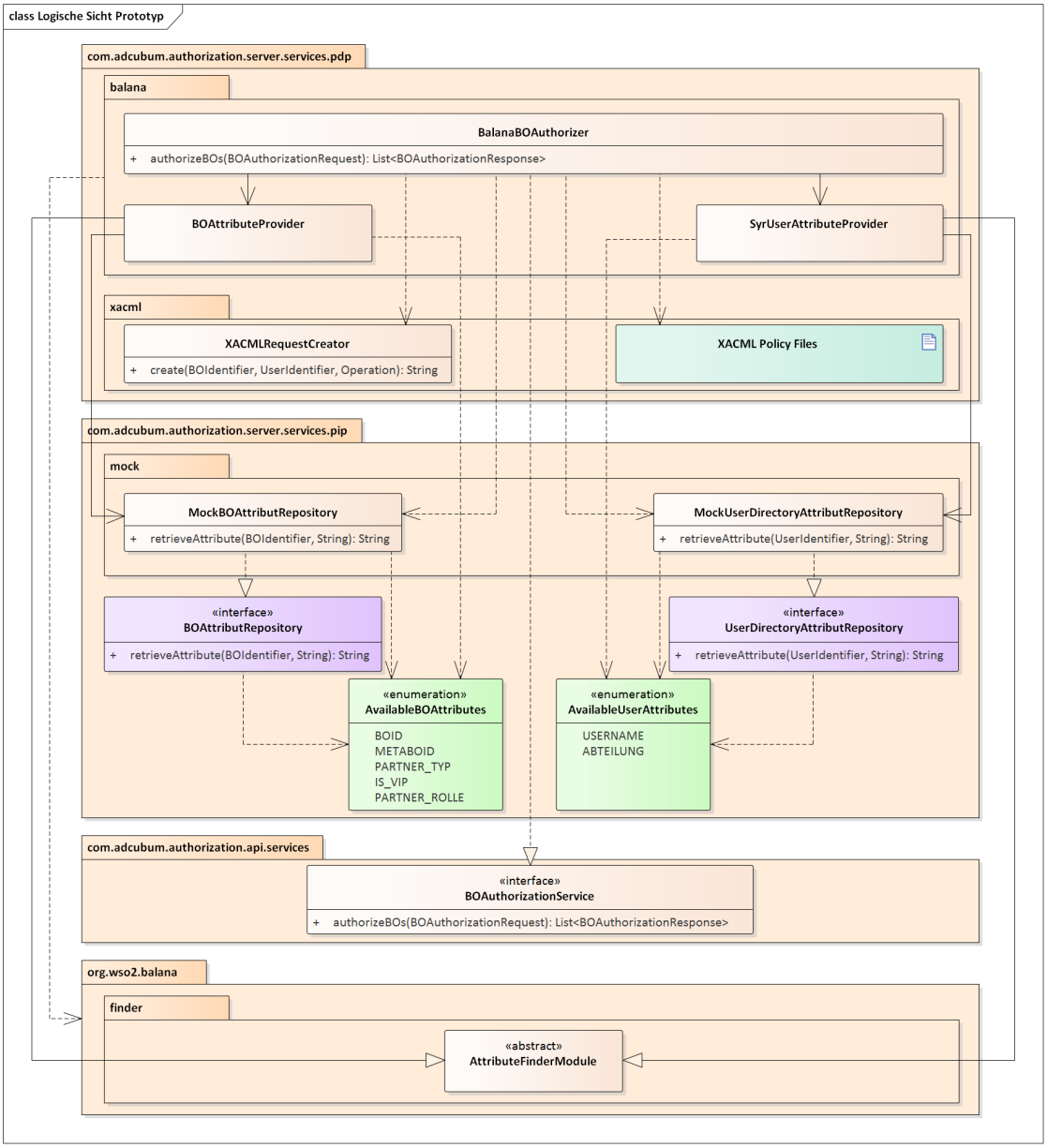

Abbildung 4.4.: Logische Sicht auf den Prototypen (UML Klassendiagramm)

Eine wichtige Anforderung an den Prototypen ist die saubere Kapselung fremder Bibliotheken, wie dies im Kapitel 3 im Abschnitt 3.3.7 Wartbarkeit festgehalten wurde. In diesem Sinne soll auch die Abhängigkeit zur verwendeten PDP-Implementation, in diesem Fall Balana [23], möglichst gering gehalten werden. Daher wurde die Implementation der Informationsbeschaffung bzw. der PIPs, nicht direkt in den *AttributeFinderModules* von Balana gemacht, sondern hinter eigenen Interfaces verborgen. Die beiden Interfaces *BOAttributRepository* und *UserDirectoryAttributRepository* kapseln diese Funktionalität

der Beschaffung von BO- und Benutzerattributen und könnten später auch von einer beliebigen anderen PDP-Implementation aufgerufen werden.

Die beiden Interfaces stellen jeweils eine simple Methode zur Verfügung, welcher entweder ein *BOIdentifier* oder ein *UserIdentifier* und das zu ermittelnde Attribut übergeben wird. Zurückgeliefert wird der Wert des entsprechenden Attributes. Beispielsweise könnte dem *UserDirectoryAttributRepository* der *UserIdentifier* «Max Müller» und das Attribut «Abteilung» übergeben werden. Der Rückgabewert wäre die entsprechende Abteilung von Max Müller, zum Beispiel «Entwicklungsabteilung».

Für den Prototypen wurden die beiden Repository-Interfaces als «Mock»-Implementation bereitgestellt, wie bereits aus Abbildung 4.3 hervorging. Diese beiden Implementationen *MockBOAttributRepository* und *MockUserDirectoryAttributRepository* geben statische Attributwerte zurück, welche für die implementierten Testfälle benötigt werden. Welche Attribute im Detail benötigt werden um die einzelnen User Stories umzusetzen, wird später im Abschnitt 4.4 Informationsbeschaffung erläutert.

Die wesentlichen Artefakte des Prototypen sind aber die XACML-Policies, da diese definieren, ob ein Benutzer Zugriff auf ein BO erhält oder nicht. Sie werden später im Abschnitt 4.3 erläutert.

4.2.4. Testing

Um einen Autorisierungsrequest evaluieren zu lassen, wird der im letzten Abschnitt in Abbildung 4.4 gezeigte *BalanaBOAuthorizer* aufgerufen. Die hinterlegten XACML-Files bestimmen für jedes BO, ob der Zugriff gewährt wird. Ob die getroffenen Entscheidungen den Erwartungen entsprechen, wurde im Prototypen über JUnit [26] Tests geprüft.

Für jede implementierte User Story wurden entsprechende Testfälle implementiert. Die Testdaten gegenüber welchen die Tests ausgeführt werden, müssen in den «Mock»- Implementationen der PIPs hinterlegt sein. Die beiden «ENUMs» *AvailableBOAttributes* und *AvailableUserAttributes*, hier in Abbildung 4.5 nochmals ersichtlich, zeigen welche Testdaten für die im Prototyp implementierten Testfälle notwendig sind.

Für die nachfolgenden Testfälle sind dabei die Attribute «Abteilung» (Subjekt) und «isVIP» (Objekt) relevant.

Insgesamt sind für die fünf implementierten Stories 23 Testfälle erstellt worden. Ein Testfall erstellt typischerweise einen *BOAuthorizationRequest* mit einem oder mehreren BOs, ruft damit den *BalanaBOAuthorizer* auf, und prüft anschliessend ob die Entscheidungen in den *BOAuthorizationResponses* den Erwartungen entsprechen.

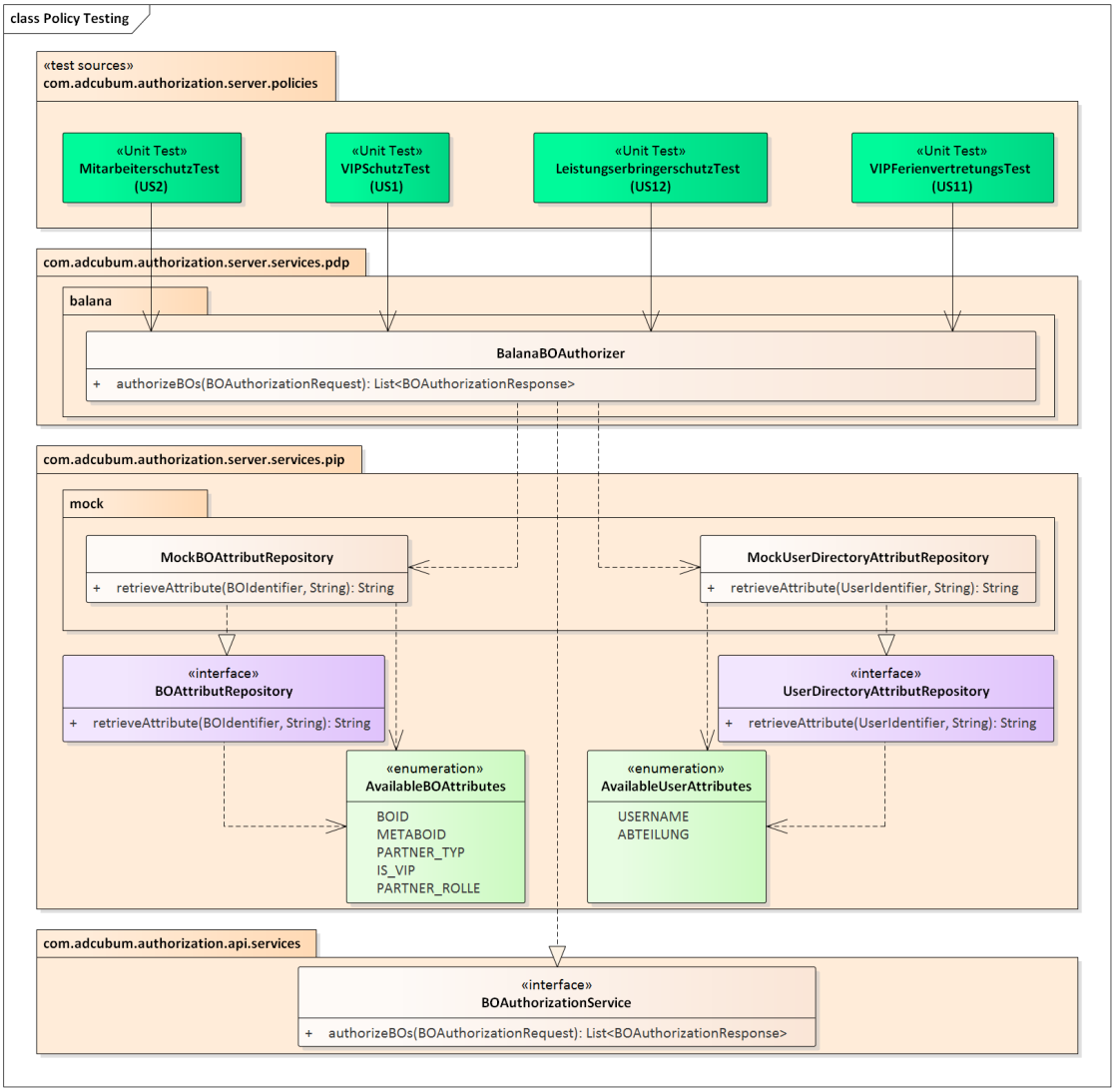

Abbildung 4.5.: Unit Tests um XACML-Policies zu testen (UML Klassendiagramm)

Auflistung 4.3 zeigt zwei Beispiele solcher Testfälle für die User Story «US1: VIP Kunden schützen». Der erste Testfall überprüft, ob ein Subjekt, ein normaler Mitarbeiter ohne Zugriffsberechtigungen auf VIPs, keinen Zugriff auf das Objekt «Patrick Superstar» (Partner) erhält. Beim zweiten Testfall hingegen, soll ein Subjekt mit dem Namen «Valter I.P. Betreuer», welcher zur Abteilung «VIPService» gehört, Schreibzugriff auf dieses Partnerobjekt erhalten. Für das Verständnis der Testfälle ist es wichtig zu wissen, dass alle Subjekte bzw. Mitarbeiter der Abteilung «VIPService» Zugriff auf VIPs erhalten sollen. Der Partner «Patrick Superstar» stellt in diesen Testfällen das Objekt dar und der PIP liefert für das Attribut *isVIP* den Wert «true», da es sich um einen VIP handelt.

```java
/**
 * Benutzer ohne spezielle Abteilung dürfen VIPs nicht einsehen.
 */
@Test
public void canProtectVIP() {
   BOAuthorizationRequest request =
         new BOAuthorizationRequestBuilder(new UserIdentifier("Peter Müller"))
         .authorizeBO(new BOIdentifier(PARTNER, "Patrick Superstar"))
         .forOperation(READ)
         .build();

   BOAuthorizationResponse response = authorizer.authorizeBOs(request).get(0);

   assertEquals(DENY, response.getDecision());
}

/**
 * Mitarbeiter der Abteilung 'VIPService' dürfen auf VIPs zugreifen.
 */
@Test
public void canWriteVIPAsBetreuer() {
   BOAuthorizationRequest request =
         new BOAuthorizationRequestBuilder(new UserIdentifier("Valter I.P. Betreuer"))
         .authorizeBO(new BOIdentifier(PARTNER, "Patrick Superstar"))
         .forOperation(WRITE)
         .build();

   BOAuthorizationResponse response = authorizer.authorizeBOs(request).get(0);

   assertEquals(PERMIT, response.getDecision());
}
```

Auflistung 4.3: Beispiel Unit Tests zur Überprüfung der VIP-Policy

Der PDP benutzt eine entsprechende XACML-Policy, in welcher diese VIP User Story abgebildet ist, damit der Test erfolgreich durchgeführt wird. Die Policies werden später im Abschnitt 4.3 erläutert. Des Weiteren muss der PIP *UserDirectoryAttributRepository* die Information bereitstellen, dass der Benutzer «Valter I.P. Bertreuer» der Abteilung «VIPService» angehört. Der PIP *BOAttributRepository* liefert die Information, dass der Partner mit dem Namen «Patrick Superstar» ein VIP ist.

4.2.5. Gesamtsicht Prototyp

Die folgende Abbildung 4.6 soll noch einmal einen Überblick über die Komponenten des Protoyps geben und dabei eine klare Abgrenzung zwischen den in der Studienarbeit [28] und den in dieser Bachelorarbeit implementierten Komponenten geben.

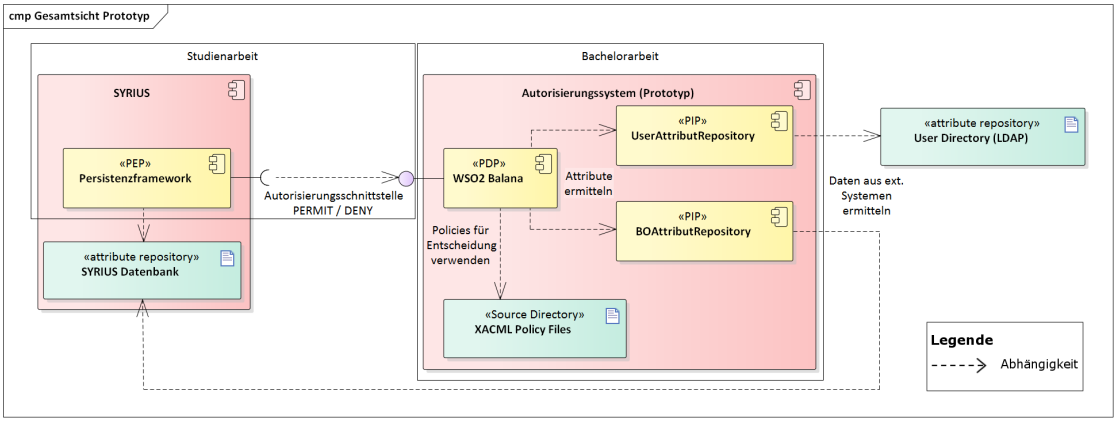

Abbildung 4.6.: Gesamtübersicht Prototyp (UML Komponentendiagramm)

In der Studienarbeit [28] wurde eine Schnittstelle entworfen und durch ein «Redesign» in der SYRIUS Persistenzschicht ein PEP implementiert, welcher diese Schnittstelle aufruft. In der Bachelorarbeit wurde nun eine Implementation eines PDP bereitgestellt und XACML-Policies implementiert um Autorisierungen durchführen zu können. Wie in der Grafik noch einmal angedeutet, wurden die PIPs nicht angebunden sondern lediglich «Mock»-Implementationen erstellt. Dies wäre nun der letzte notwendige Schritt, um den Prototyp dahingehend zu vervollständigen, dass aus SYRIUS Autorisierungen durchgeführt werden können, ohne dabei «Mock»-Daten zu verwenden.

Nachdem die Architektur nun erläutert wurde, geht der nächste Abschnitt auf die wichtigsten Artefakte aus der Weiterentwicklung des Prototypen, den Policies, ein.

4.3. Policies

Wie im Abschnitt 4.1 Design- und Architekturentscheidungen bereits erklärt, wurden die Policies für den Prototypen dieser Bachelorarbeit mit XACML umgesetzt. Bevor auf die im Prototypen umgesetzten Policies eingegangen wird, muss die Struktur von XACML und dessen Sprachmodell erklärt werden.

4.3.1. Einführung in XACML

Der XACML Standard [56], welcher von OASIS [39] gepflegt wird, definiert ein XML-Schema zur Erstellung von Sicherheitspolicies. XACML ist ausserdem als «de facto»-Standard für die Implementation von ABAC-basierten Policies bekannt [55, 21]. Die Evaluation der Policy-Syntax aus Abschnitt 4.1.3 bestätigt diese Aussage. Während der Standard [56] eine sehr grosse Anzahl Elemente umfasst, werden an dieser Stelle lediglich die wesentlichsten Elemente erklärt, welche notwendig sind um die umgesetzten Policies des Prototypen zu verstehen. Das nachfolgende Sprachmodell in Abbildung 4.7 ist dem Standard [56] von OASIS [39] entnommen. Es wurde vereinfacht, sodass nur die in dieser Arbeit verwendeten Elemente enthalten sind.

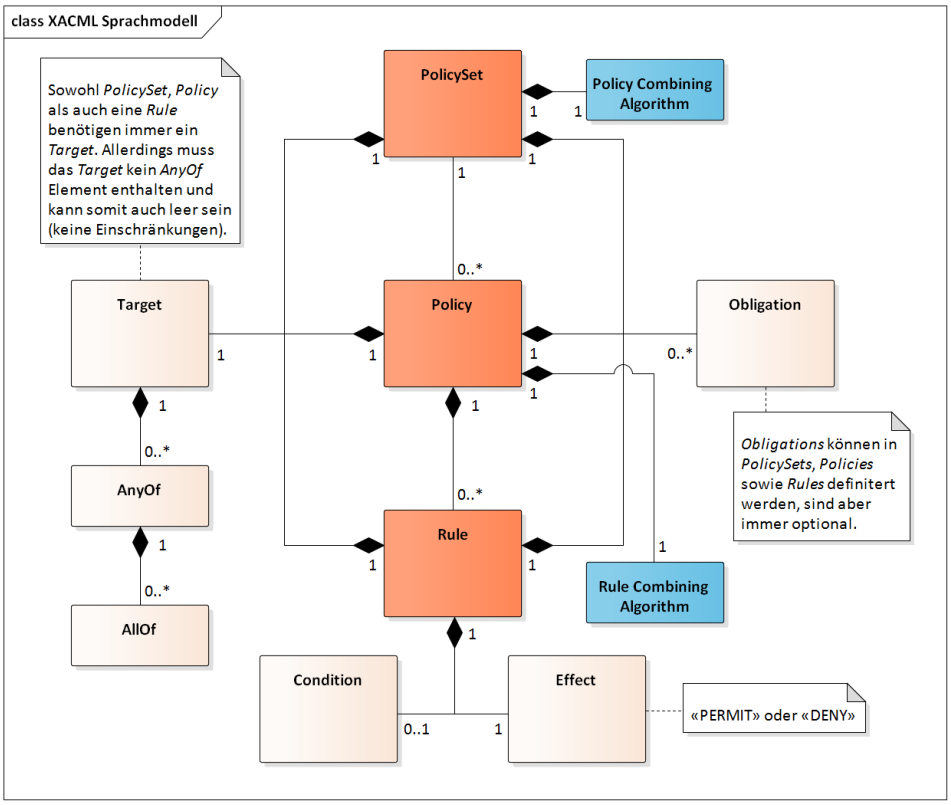

Abbildung 4.7.: Vereinfachtes XACML-Modell [56] (UML Klassendiagramm)

Die Kompositionen in Abbildung 4.7 (Ganzes-/Teile-Beziehung) zeigen an, dass die Teilelemente existenziell von den übergeordneten Elementen abhängig sind und somit auch im XML-Dokument nur diesen Hauptelementen untergeordnet werden können. Zum Beispiel kann eine *Rule* nur einer *Policy* untergeordnet werden.

Gemäss dem XACML Standard [56] von OASIS sind die Komponenten *PolicySet*, *Policy* und *Rule* die Hauptelemente des XACML Modells. Die *Rules* definieren die Zugriffsregeln innerhalb einer *Policy*. Eine *Rule* definiert eine Reihe von *Conditions* (Bedingungen) welche zutreffen müssen, damit die *Rule* zu einem bestimmten *Effect* evaluiert. Ein *Effect* ist immer entweder «PERMIT» oder «DENY». Da eine *Policy* mehrere *Rules* enthalten kann, ist es möglich, dass beim evaluieren eines XACML-Requests unterschiedliche *Effects* innerhalb einer *Policy* resultieren. In diesem Fall definiert der *Rule Combining Algorithm*, welche *Rule* stärker zu werten ist bzw. ob die *Policy* «PERMIT» oder «DENY» zurückliefert.

Ausserdem muss eine *Rule* ein *Target* enthalten. Ein *Target* gibt vor, für welche Art von Autorisierungsanfragen eine *Rule*, ein *PolicySet* oder eine *Policy* angewendet werden soll. Das *Target* definiert Bedingungen, welche auf eine Anfrage zutreffen müssen, damit der PDP das entsprechende Element berücksichtigt. Allerdings kann das *Target* auch leer sein und muss nicht zwingend eine solche Bedingung enthalten. In diesem Fall wäre die *Rule*, das *PolicySet* oder die *Policy* auf alle Anfragen anwendbar. Neben der *Rule* muss also auch ein *PolicySet* oder eine *Policy* ein *Target* definieren, damit der PDP anhand der XACML-Requests ermitteln kann, welche Policies für die Evaluation des Autorisierungsrequests verwendet werden müssen. Über die Elemente *AnyOf* und *AllOf* werden die Bedingungen definiert, von welchen innerhalb jedes *AnyOf*-Elements mindestens eines der *AllOf*-Elemente erfüllt sein muss, damit das *PolicySet*, die *Policy* oder die *Rule* angewendet wird. Wird ein XACML-Request, für welchen keine *Policies* mit passendem *Target* vorhanden sind, an den PDP geschickt, kann keine Entscheidung getroffen werden. Der PDP gibt in diesem Fall statt «PERMIT» oder «DENY», «NOT APPLICABLE» zurück.

Ein *PolicySet* ist eine Sammlung mehrerer *Policies*. Wie bei den *Rules* innerhalb einer *Policy*, kann es auch auf Ebene der *Policies* die Situation geben, in welcher die *Targets* mehrerer *Policies* zu einem XACML-Request passen. In diesem Fall entscheidet der *Policy Combining Algorithm*, welcher *Effect* der verschiedenen *Policies* zurückgeliefert wird. Sofern sich alle entsprechenden *Policies* innerhalb eines *PolicySets* befinden, kann dieser *Policy Combining Algorithm* innerhalb des *PolicySets* definiert werden. Sobald sich die überschneidenden *Targets* für einen Request allerdings über mehrere XACML-Dateien verteilen, muss der *Policy Combining Algorithm* im PDP konfiguriert werden.

Das letzte Element, welches hier behandelt wird, ist die *Obligation*. Sie bietet dem PDP die Möglichkeit, zusätzliche Informationen zu der getroffenen Entscheidung für den PEP zur Verfügung zu stellen. Eine *Obligation* gibt dem PEP eine zusätzliche Anweisung, welche er bei der Umsetzung der Autorisierungsentscheidung zwingend umsetzen muss.

4.3.2. XACML-Policies im Prototypen

Nachdem die grundlegenden Elemente von XACML erklärt wurden, kann nun ein Beispiel einer im Prototypen umgesetzten Policy gezeigt werden. Nachfolgende Tabelle zeigt alle XACML-Policy-Dateien, welche im Prototypen implementiert wurden und welche User Stories darin umgesetzt sind.

Tabelle 4.7.: Implementierte XACML-Policies und die dazugehörigen User Stories

XACML-Datei	**Repräsentierende Stories**
vipPolicySet.xml	• US1: VIP Kunden schützen • US11: Vertretungen bei Absenzen berechtigen (Beispiel für eine Vertretung eines VIP-Betreuers)
mitarbeiterPolicySet.xml	• US2: Mitarbeiter schützen
technicalUserPolicy.xml	• US7: Technischen User für Auswertungen berechtigen
leistungserbringerPolicySet.xml	• US12: Leistungserbringer vor Mutation schützen

Beispiel einer Policy aus dem Prototypen

Zur Illustration einer eigens implementierten Policy, greifen wir erneut auf die User Story «US1: VIP Kunden schützen» zurück. Das *Target* dieser Policy legt einerseits fest, für welche BOs diese Policy evaluiert werden soll und andererseits für welche *Operationen* sie zum Zug kommt.

Das *Target* dieses Beispiels ist folglich in drei *AnyOf*-Elemente eingeteilt, welche folgende Einschränkungen vornehmen:

- 1. MetaBO (erlaubte Typen der BOs)
- 2. Einschränkung auf alle BOs, welche zu einem VIP-Partner gehören.
- 3. Einschränkung der Operation.

Für die Eingrenzung der von der *Policy* betroffenen MetaBOs dient die MetaBOId, welche festlegt um welchen Typ von BO es sich handelt. In diesem Fall handelt es sich um den Partner und alle abhängigen Objekte, wie zum Beispiel Adressen oder Verträge, welche ebenfalls geschützt werden müssen. Dieser Abschnitt des *Targets* wird in Auflistung 4.4 gezeigt. Die Zeilen im XACML, welche die MetaBO-Ids festlegen, wurden entsprechend markiert.

```
<!-- Target vergleicht die MetaBoID des Requests mit den folgenden, von der Policy abgedeckten,
MetaBoIDs. -->
<xacml3:Target>
  <xacml3:AnyOf>
    <xacml3:AllOf>
      <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-equal">
        <xacml3:AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#integer">-3</xacml3:AttributeValue>
        <xacml3:AttributeDesignator
          AttributeId="http://adcubum.com/syrius/bo/metaBoId"
          DataType="http://www.w3.org/2001/XMLSchema#integer"
          Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
          MustBePresent="false" />
      </xacml3:Match>
    </xacml3:AllOf>
    <xacml3:AllOf>
      <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-equal">
        <xacml3:AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#integer">-7</xacml3:AttributeValue>
        <xacml3:AttributeDesignator
          AttributeId="http://adcubum.com/syrius/bo/metaBoId"
          DataType="http://www.w3.org/2001/XMLSchema#integer"
          Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
          MustBePresent="false" />
      </xacml3:Match>
    </xacml3:AllOf>
    <xacml3:AllOf>
      <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-equal">
        <xacml3:AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#integer">-34</xacml3:AttributeValue>
        <xacml3:AttributeDesignator
          AttributeId="http://adcubum.com/syrius/bo/metaBoId"
          DataType="http://www.w3.org/2001/XMLSchema#integer"
          Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
          MustBePresent="false" />
      </xacml3:Match>
    </xacml3:AllOf>
    <!-- weitere MetaBOIds für bessere Darstellung in Bericht entfernt -->
  </xacml3:AnyOf>
```

Auflistung 4.4: Policy-Beispiel: Target für Eingrenzung des BO-Typs

In der nächsten Auflistung 4.5 wird ein zusätzliches *AnyOf*-Element des *Targets* dieser *Policy* gezeigt, welches auf alle Partner eingrenzt, die VIPs sind.

Für dieses *Target* ist es wichtig, dass der entsprechende PIP für BO-Attribute, im Prototypen das «BOAttributRepository», dieses Attribut «isVIP» für alle BO-Typen bereitstellt, welche im obigen *AnyOf*-Element (Auflistung 4.4) angegeben wurden. Der PIP muss also die sogenannten Schutzpfade kennen und das Attribut «isVIP» für alle BOs innerhalb des Schutzpfades bereitstellen.

Die markierten Zeilen in Abbildung 4.5 zeigen das Attribut «isVIP» und dass der Wert von «isVIP» «true» sein muss.

```
<!-- Die Policy wird nur angewendet, wenn der Partner ein VIP ist. -->
  <xacml3:AnyOf>
    <xacml3:AllOf>
      <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:boolean-equal">
        <xacml3:AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#boolean">true</xacml3:AttributeValue>
        <xacml3:AttributeDesignator
          AttributeId="http://adcubum.com/syrius/bo/partner/isVIP"
          DataType="http://www.w3.org/2001/XMLSchema#boolean"
          Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
          MustBePresent="false" />
      </xacml3:Match>
    </xacml3:AllOf>
  </xacml3:AnyOf>
```

Auflistung 4.5: Policy-Beispiel: Target für Eingrenzung aller Partner welcher VIPs sind.

Zum Abschluss des *Targets* dieser Policy wird, wie bereits erwähnt, angegeben für welche *Operationen* die Policy gilt. In diesem Fall sind dies beide *Operationen* «READ» und «WRITE».

```
<!-- Aktionen READ und WRITE sind von dieser Policy betroffen. -->
  <xacml3:AnyOf>
    <xacml3:AllOf>
      <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <xacml3:AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">READ</xacml3:AttributeValue>
        <xacml3:AttributeDesignator
          AttributeId="http://adcubum.com/syrius/operation"
          DataType="http://www.w3.org/2001/XMLSchema#string"
          Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
          MustBePresent="false" />
      </xacml3:Match>
    </xacml3:AllOf>
    <xacml3:AllOf>
      <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <xacml3:AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">WRITE</xacml3:AttributeValue>
        <xacml3:AttributeDesignator
          AttributeId="http://adcubum.com/syrius/operation"
          DataType="http://www.w3.org/2001/XMLSchema#string"
          Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
          MustBePresent="false" />
      </xacml3:Match>
    </xacml3:AllOf>
  </xacml3:AnyOf>
</xacml3:Target>
```

Auflistung 4.6: Policy-Beispiel: Target für Eingrenzung der Operation.

Die restliche Policy setzt sich nun aus zwei *Rules* zusammen, welche den Zugriff auf die VIPs regeln. Die erste Regel, welche in Auflistung 4.7 abgebildet ist, legt den *Effect* «PERMIT» für alle Benutzer der Abteilung «VIPService» fest.

```
<!-- Rule berechtigt Mitarbeiter der Abteilung 'VIPService' zur Bearbeitung von VIPs -->
<xacml3:Rule
  Effect="Permit"
  RuleId="http://adcubum.com/identifier/authorization/vipPolicySet/vipPolicy/allowAccessVIP..."/>
  <xacml3:Description />
  <xacml3:Target />
  <xacml3:Condition>
    <xacml3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
      <xacml3:Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
      <xacml3:AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">VIPService</xacml3:AttributeValue>
      <xacml3:AttributeDesignator
        AttributeId="http://adcubum.com/syrius/syruser/abteilung"
        DataType="http://www.w3.org/2001/XMLSchema#string"
        Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
        MustBePresent="false" />
    </xacml3:Apply>
  </xacml3:Condition>
</xacml3:Rule>
```

Auflistung 4.7: Policy-Beispiel: Rule für alle Mitarbeiter der Abteilung «VIPService»

Die zweite *Rule* verbietet den Zugriff für alle anderen Fälle. Sie erzwingt also den *Effect* «DENY» für alle Requests welche nicht durch die erste Regel abgedeckt werden.

```
<!-- Zugriff für andere wird verweigert. -->
<xacml3:Rule
  Effect="Deny"
  RuleId="http://adcubum.com/identifier/authorization/vipPolicySet/vipPolicy/denyForOthers">
  <xacml3:Description />
  <xacml3:Target />
</xacml3:Rule>
```

Auflistung 4.8: Policy-Beispiel: Rule für restliche Mitarbeiter.

Damit diese Policy wie gewünscht evaluiert wird, ist der *Rule Combining Algorithm* entscheidend. Der *Effect* «PERMIT» der ersten *Rule* muss stärker gewichtet werden als der «DENY» der zweiten *Rule*, da beide *Rules* auf denselben Request anwendbar sind. Dazu zeigt Auflistung 4.9 den Policy-Tag, welcher die obigen *Targets* und *Rules* umschliesst.

```
<!-- Permit-Overrides: Sobald eine Rule Permit retourniert, wird der Zugriff gewährt. -->
<xacml3:Policy xmlns:xacml3="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
  PolicyId="http://adcubum.com/authorization/boPolicies/vipPolicy"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides"
  Version="1.0">
  <xacml3:Description />
  <xacml3:PolicyDefaults>
    <xacml3:XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</xacml3:XPathVersion>
  </xacml3:PolicyDefaults>

  <!-- Target und Rules: siehe Auflistungen weiter oben -->

</xacml3:Policy>
```

Auflistung 4.9: Policy-Beispiel: Policy-Tag mit *Rule Combining Algorithm*

Daraus wird ersichtlich, dass der *Rule Combining Algorithm* «*permit-overrides*» hinterlegt wurde. Dieser legt fest, dass «PERMIT»-Entscheidungen stärker gewichtet werden als «DENY». Welche *Combining Algorithms* es gibt, wird anschliessend im Abschnitt 4.3.3 detailliert erläutert.

Die gesamte VIP-Policy, welche auf den letzten Seiten in Form von XACML-Ausschnitten gezeigt wurde, kann auch anhand eines Objektdiagramms dargestellt werden. Abbildung 4.8 zeigt dieses Objektdiagramm, welches auf den XACML-Elementen der Abbildung 4.7 (Anfang dieses Abschnitts) basiert.

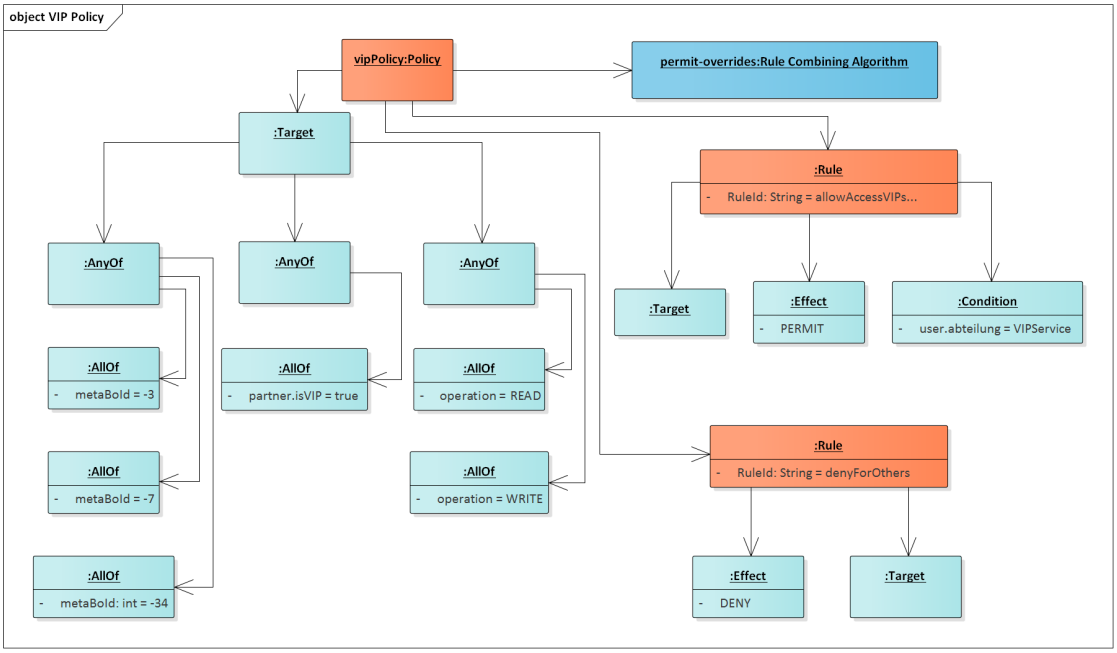

Abbildung 4.8.: VIP-Policy anhand des XACML-Modells [56] (UML Objektdiagramm)

Attributschutz

Eine weitere Anforderung, welche bereits an die Schnittstelle der Studienarbeit [28] gestellt wurde, ist der Attributschutz. In einigen Use Stories, wie zum Beispiel «US2: Mitarbeiter schützen» (siehe Abschnitt 3.2.5), wird gefordert, dass ein Benutzer ein BO zwar finden, aber nur auf einen Teil der Daten zugreifen kann.

Der PDP muss in einem solchen Fall «PERMIT» als *Effect* zurückliefern, damit der Benutzer auf das BO zugreifen kann. Allerdings muss der PDP dem PEP mitteilen können, welche Attribute er dem Benutzer nicht zeigen darf. Dafür wurden die *Obligations* (siehe Abbildung 4.7) aus dem XACML Standard [56] verwendet. Im Prototypen wurde dies anhand der «Mitarbeiter»-Policy implementiert. Die *Rule*, welche den Zugriff auf die Mitarbeiter einer Versicherung für alle Benutzer erlaubt, wird um *Obligations* erweitert, die den Zugriff auf kritische Attribute verbieten. Folgende Auflistung 4.10 zeigt diese *Rule*. Die gesperrten Attribute wurden entsprechend markiert. Des Weiteren wird eine Nachricht zurückgeliefert, welche dem PEP mitteilt, was er mit diesen Attributen tun muss.

```
<xacml3:Rule Effect="Permit"
  RuleId="http://adcubum.com/.../mitarbeiterPolicy/allowLimitedAccessToPartnerBO">
  <xacml3:Target />
  <xacml3:Condition>
  <!--Details der Rule für diese Auflistung entfernt (nicht von Bedeutung für Attributschutz).-->
  </xacml3:Condition>
  <!--Liste mit Attributen welche nicht öffentlich sein sollen, falls der Benutzer nicht
  im HR arbeitet.-->
  <xacml3:ObligationExpressions>
    <xacml3:ObligationExpression ObligationId="http://adcubum.com/.../unauthorizedAttributes"
      FulfillOn="Permit">
      <xacml3:AttributeAssignmentExpression AttributeId="http://adcubum.com/.../message">
        <xacml3:AttributeValue>The user has access to the Partner, except the given attributes.
        These should not be seen by the user.</xacml3:AttributeValue>
      </xacml3:AttributeAssignmentExpression>
      <xacml3:AttributeAssignmentExpression AttributeId="http://adcubum.com/.../attributeName">
        <xacml3:AttributeValue>Geburtstag</xacml3:AttributeValue>
      </xacml3:AttributeAssignmentExpression>
      <xacml3:AttributeAssignmentExpression AttributeId="http://adcubum.com/.../attributeName">
        <xacml3:AttributeValue>Zivilstand</xacml3:AttributeValue>
      </xacml3:AttributeAssignmentExpression>
      <xacml3:AttributeAssignmentExpression AttributeId="http://adcubum.com/.../attributeName">
        <xacml3:AttributeValue>Heimatort</xacml3:AttributeValue>
      </xacml3:AttributeAssignmentExpression>
    </xacml3:ObligationExpression>
  </xacml3:ObligationExpressions>
</xacml3:Rule>
```

Auflistung 4.10: Beispiel einer *Rule* mit Attributschutz (vereinfacht)

In diesem Beispiel aus Auflistung 4.10 werden die Attribute «Geburtstag», «Zivilstand» und «Heimatort» durch *Obligations* geschützt. Allerdings ist der PEP, in unserem Fall SYRIUS, dafür verantwortlich, diese Attribute für den Benutzer unzugänglich zu machen.

4.3.3. Strukturierung der Policies

Nachdem der letzte Abschnitt eine einzelne Policy als Beispiel gezeigt hat, geht dieser Abschnitt auf die Herausforderung der Strukturierung aller Policies innerhalb eines PDP ein. Sobald der Umfang an Policies grösser wird, nimmt die Komplexität zu und es wird notwendig, sich ein System für die Strukturierung der Policies zu überlegen. Gerade dann, wenn unterschiedliche Anwendungsfälle die gleichen BOs schützen und somit mehrere *Policies* auf einen XACML-Request anwendbar sind.

Combining Algorithms

Um eine Strategie für die Strukturierung von XACML-Policies festlegen zu können, sind die sogenannten *Combining Algorithms* entscheidend. Wie im Sprachmodell von XACML in Abbildung 4.7 bereits gezeigt wurde, gibt es hier eine Unterscheidung zwischen *Policy Combining Algorithms* und *Rule Combining Algorithms*. *Combining Algorithms* kommen immer dann zum Zug, wenn mehrere *Policies* bzw. *Rules* für eine Autorisierungsanfrage evaluiert werden und zu einer unterschiedlichen Entscheidung gelangen. Die folgende Tabelle 4.8 zeigt die in XACML standardmässig implementierten *Combining Algorithms*. Es ist auch möglich eigene Algorithmen zu implementieren, falls der Standard für die eigenen Anforderungen nicht ausreicht. Im Prototypen dieser Bachelorarbeit wurde mit den Standard-Algorithmen aus der Tabelle 4.8 gearbeitet. Bis auf eine einzige Ausnahme, dem *only-one-applicable* Algorithmus, sind alle Algorithmen sowohl als *Policy Combining Algorithm* wie auch als *Rule Combining Algorithm* einsetzbar.

Tabelle 4.8.: Combining Algorithms aus XACML Standard [56]

Algorithmus	**Beschreibung**
deny-overrides (für *Policies* & *Rules*)	Bei diesem Algorithmus wird ein «DENY» stets höher priorisiert als ein «PERMIT». Sobald eine der *Policies* oder *Rules* zu «DENY» evaluiert, wird «DENY» auch als Gesamtresultat zurückgeliefert.
permit-overrides (für *Policies* & *Rules*)	Bei diesem Algorithmus wird ein «PERMIT» stets höher priorisiert als ein «DENY». Sobald eine der *Policies* oder *Rules* zu «PERMIT» evaluiert, wird «PERMIT» auch als Gesamtresultat zurückgeliefert.
only-one-applicable (nur für *Policies* verwendbar)	Dieser Algorithmus liefert das Resultat einer Policy, falls diese Policy die einzige ist bei welcher das *Target* übereinstimmt und die Policy damit die einzige ist, welche anwendbar ist. Wenn es keine oder mehrere Policies gibt, welche in Frage kommen, liefert dieser Algorithmus «INDETERMINATE».

Tabelle 4.8.: Combining Algorithms aus XACML Standard [56]

Algorithmus	**Beschreibung**
first-applicable (für *Policies* & *Rules*)	Im Falle dieses Algorithmus werden die *Policies* in der Reihenfolge evaluiert, in welcher sie im *PolicySet* definiert sind. Das Resultat der ersten Policy, bei welcher das *Target* mit dem Request übereinstimmt und entweder einen «PERMIT» oder «DENY» zurückliefert, wird als Gesamtresultat gewertet. Im Falle der *Rules* ist die Reihenfolge innerhalb einer einzelnen *Policy* entscheidend. Das Resultat der ersten *Rule*, bei welcher das *Target* mit dem Request übereinstimmt und die *Conditions* zu «true» evaluieren, wird als Gesamtresultat gewertet. Passt bei einer *Rule* das *Target* nicht oder eine der *Conditions* evaluiert zu «false», wird die nächste *Rule* evaluiert.
ordered-deny-overrides (für *Policies* & *Rules*)	Dieser Algorithmus verhält sich gleich wie der Algorithmus *deny-overrides*, mit der Ausnahme, dass die Resultate in geordneter Reihenfolge evaluiert werden müssen. Die *Policies* werden in der Reihenfolge evaluiert, wie sie im *PolicySet* aufgeführt sind. Bei den *Rules* entscheided die Reihenfolge, wie sie in der Policy aufgeführt sind.
ordered-permit-overrides (für *Policies* & *Rules*)	Dieser Algorithmus verhält sich gleich wie der Algorithmus *permit-overrides*, mit der Ausnahme, dass die Resultate in geordneter Reihenfolge evaluiert werden müssen. Die *Policies* werden in der Reihenfolge evaluiert, wie sie im *PolicySet* aufgeführt sind. Bei den *Rules* entscheided die Reihenfolge, wie sie in der Policy aufgeführt sind.
deny-unless-permit (für *Policies* & *Rules*)	Der *deny-unless-permit* Algorithmus wird verwendet wenn ein «PERMIT» höher priorisiert werden soll als ein «DENY», «NOT APPLICABLE» oder «INDETERMINATE» aber kein gültiges Resultat sein sollen. Solange keines der Resultate «PERMIT» ist, gibt dieser Algorithmus immer «DENY» als Endergebnis zurück.
permit-unless-deny (für *Policies* & *Rules*)	Der *permit-unless-deny* Algorithmus wird verwendet wenn ein «DENY» höher priorisiert werden soll als ein «PERMIT», «NOT APPLICABLE» oder «INDETERMINATE» aber kein gültiges Resultat sein sollen. Solange keines der Resultate «DENY» ist, gibt dieser Algorithmus immer «PERMIT» als Endergebnis zurück.

Nachfolgend wird der genaue Einsatz dieser Algorithmen im Prototypen erläutert.

Strukturierung auf *Rule*-Ebene

Die *Rules* innerhalb der Policies des Prototypen basieren auf dem Prinzip, dass grundsätzlich niemand Zugriff auf die entsprechenden BOs hat, ausser eine entsprechende Regel erlaubt dies. Alle Policies sind mit einer Standard-*Rule* versehen, welche den Zugriff für alle Anfragen verbietet, die nicht explizit durch eine andere *Rule* berechtigt wurden. Auflistungen 4.11 und 4.12 zeigen dies erneut am VIP-Beispiel.

```
<!-- Zugriff für andere wird verweigert. -->
<xacml3:Rule
    Effect="Deny"
    RuleId="http://adcubum.com/identifier/authorization/vipPolicySet/vipPolicy/denyForOthers">
  <xacml3:Description />
  <xacml3:Target />
</xacml3:Rule>
```

Auflistung 4.11: *denyForOthers*-Rule

Die *Rule* «denyForOthers» sorgt dafür, dass der Zugriff grundsätzlich verboten wird. Durch andere *Rules*, wie in diesem Beispiel «allowAccessVIPservice», wird der Zugriff explizit unter bestimmten Bedingungen erlaubt.

```
<!-- Rule berechtigt Mitarbeiter der Abteilung 'VIPService' zur Bearbeitung von VIPs -->
<xacml3:Rule
    Effect="Permit"
    RuleId="http://adcubum.com/identifier/authorization/vipPolicySet/vipPolicy/allowAccessVI...">
  <xacml3:Description />
  <xacml3:Target />
  <xacml3:Condition>
    <xacml3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
      <xacml3:Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
      <xacml3:AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">VIPService</xacml3:AttributeValue>
      <xacml3:AttributeDesignator
        AttributeId="http://adcubum.com/syrius/syruser/abteilung"
        DataType="http://www.w3.org/2001/XMLSchema#string"
        Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
        MustBePresent="false"
      />
    </xacml3:Apply>
  </xacml3:Condition>
</xacml3:Rule>
```

Auflistung 4.12: *Rule* für explizite Berechtigungsvergabe an Abteilung «VIPService»

Bei einer solchen Strukturierung der *Rules* muss ein Algorithmus gewählt werden, welcher «PERMIT» höher priorisiert als «DENY». Sobald für einen Request eine *Rule* zu «PERMIT» evaluiert, soll auch das Gesamtergebnis der Policy «PERMIT» sein. Aus die-

sem Grund wird im Prototypen immer *permit-overrides* als *Rule Combining Algorithm* verwendet.

Strukturierung auf *Policy*-Ebene

Auch auf Ebene der Policies müssen Überlegungen in Bezug auf die Struktur und die *Combining Algorithms* gemacht werden. Sobald der PDP mit mehreren Policies arbeitet, kann es zu Situationen kommen, in denen für einen Request mehrere Policies evaluiert werden können. Die Anforderung aus der User Story «US7: Technischen User für Massenverarbeitungen berechtigen» spielt hier ein zentrale Rolle. Technischen Benutzern oder Administratoren soll der Zugriff auf alle Daten gewährt werden können, ohne dass die restlichen Policies berücksichtigt werden. Es gibt im Prototypen dieser Arbeit bereits aufgrund dieser User Story in jedem Fall mindestens zwei Policies, welche für einen Autorisierungsrequest evaluiert werden können bzw. das *Target* mit dem Request übereinstimmt. Im Fall der «admin»-Policy ist es sogar so, dass sie gar kein *Target* besitzt und somit für alle Requests berücksichtigt werden muss.

Für den Fall der technischen Benutzer wurde eine eigene Policy definiert, welche für bestimmte Benutzernamen in jedem Fall einen «PERMIT» liefert, unabhängig davon welches Objekt autorisiert werden soll.

```
<xacml3:Rule Effect="Permit" RuleId="http://adcubum.com/.../allowAllOperationsForTechnicalUsers">
  <xacml3:Target>
    <xacml3:AnyOf>
      <xacml3:AllOf>
        <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <xacml3:AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">admin</xacml3:AttributeValue>
          <xacml3:AttributeDesignator AttributeId="http://adcubum.com/syrius/syruser/username"
            DataType="http://www.w3.org/2001/XMLSchema#string"
            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
            MustBePresent="false" />
        </xacml3:Match>
      </xacml3:AllOf>
      <xacml3:AllOf>
        <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <xacml3:AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">batch-user</xacml3:AttributeValue>
          <xacml3:AttributeDesignator AttributeId="http://adcubum.com/syrius/syruser/username"
            DataType="http://www.w3.org/2001/XMLSchema#string"
            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
            MustBePresent="false" />
        </xacml3:Match>
      </xacml3:AllOf>
    </xacml3:AnyOf>
  </xacml3:Target>
</xacml3:Rule>
```

Auflistung 4.13: Policy zur Autorisierung technischer Benutzer

Auflistung 4.13 zeigt die einzige *Rule*, welche diese «admin»-Policy besitzt. Diese *Rule* legt fest, dass für die beiden Benutzer mit den Namen «admin» und «batch-user» in jedem Fall ein «PERMIT» evaluiert wird.

Bei der Wahl des *Policy Combining Algorithm* wurde dieselbe Strategie wie auf der Ebene der *Rules* gewählt. Es wird grundsätzlich davon ausgegangen, dass der Benutzer keine Rechte besitzt und diese explizit vergeben werden. Dieses Vorgehen entspricht auch dem Prozess bezüglich der Rechtevergabe bei den Kunden. Aus den Kundeninterviews wurde während der Anforderungsanalyse festgestellt, dass die Versicherer nach dem sogenannten «Need to Know»-Prinzip arbeiten. Das Ziel dieses Prinzip ist es, dass ein Benutzer nur exakt auf die Daten zugreifen kann, welche er für seine Arbeit benötigt. Aus diesem Grund werden bei der Rechtevergabe die Rechte explizit vergeben. Werden einem Benutzer keine Rechte zugeteilt, hat er gar keine Berechtigungen auf SYRIUS-Daten. Dieses Prinzip entspricht ausserdem dem «Principle of Least Privilege» [45], welches im Bereich der Informationssicherheit als «Good Practice» [37, 40] angesehen wird.

Entsprechend diesem Prinzip, wurde als *Policy Combining Algorithm* ebenfalls der Algorithmus *policy-combining-algorithm:permit-overrides* gewählt, da aufgrund der Strukturierung der *Rules* und des entsprechenden *Rule Combining Algorithms* davon ausgegangen werden kann, dass eine Policy nur zu «PERMIT» evaluiert, wenn explizit eine *Rule* dafür definiert wurde. Sobald mindestens eine Policy zu «PERMIT» evaluiert, soll das Ergebnis einer Autorisierungsanfrage ebenfalls «PERMIT» sein. Durch diesen Mechanismus kann mittels der obigen «admin»-Policy auch sichergestellt werden, dass ein technischer Benutzer in jedem Fall den Zugriff erhält. Unabhängig davon zu welchen *Effects* die restlichen Policies evaluieren.

Um den *Policy Combining Algorithm* festzulegen, wurde im Prototypen mit *PolicySets* gearbeitet. Diese ermöglichen es den Algorithmus festzulegen und damit den PDP aufzufordern, alle Resultate der *Policies* innerhalb des *PolicySets* mit diesem Algorithmus auszuwerten. Da es zu unübersichtlich wäre alle Policies in ein *PolicySet* zusammenzufassen, wurde pro User Story ein eigenes *PolicySet* angelegt. Damit die «admin»-Policy nicht in jedem *PolicySet* redundant implementiert werden muss, wurde auf den Mechanismus der Policy-Referenzen zurückgegriffen. Mittels Policy-Referenzen kann ein *PolicySet* eine Policy, welche in einer separaten XML-Datei liegt, über die *PolicyId* referenzieren.

Auflistung 4.14 zeigt dieses Prinzip am Beispiel des VIP-*PolicySets*. Das *PolicySet* enthält die für diese User Story spezifische Policy (Inhalt für diese Auflistung entfernt) und eine entsprechende Referenz auf die «admin»-Policy, welche sich in einer eigenen Datei befindet und so wiederverwendet werden kann.

```
<xacml3:PolicySet xmlns:xacml3="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
  PolicySetId="http://adcubum.com/identifier/authorization/vipPolicySet"
  PolicyCombiningAlgId="urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides"
  Version="1.0">

  <!-- Policy für VIP Schutz. Schützt vor Lese und Schreibzugriffen von nichtberechtigten
  Mitarbeitern. Zu testen mit UnitTest VIPSchutzTest.java-->
  <xacml3:Description>VIP-Schutz-Policy-Set</xacml3:Description>
  <xacml3:PolicySetDefaults>
    <xacml3:XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</xacml3:XPathVersion>
  </xacml3:PolicySetDefaults>
  <xacml3:Target />
  <!-- Permit-Overrides: Sobald eine Rule Permit retourniert, wird der Zugriff gewährt. -->
  <xacml3:Policy xmlns:xacml3="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
      PolicyId="http://adcubum.com/authorization/boPolicies/vipPolicy"
      RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides"
      Version="1.0">
    <!-- Inhalt der VIP-Policy für diese Auflistung entfernt.  -->
  </xacml3:Policy>

  <!-- Referenz um Admin-Policy einzubinden. Admin erhält so Berechtigungen. -->
  <xacml3:PolicyIdReference>http://adcubum.com/authorization/technicalUserPolicy
  </xacml3:PolicyIdReference>
</xacml3:PolicySet>
```

Auflistung 4.14: Policy-Referenz um Policies in anderen *PolicySets* wiederzuverwenden

Diese Lösung mit einem *PolicySet* pro User Story bietet den Vorteil, dass der Zugriff für technische Benutzer auch feingranularer gelöst werden könnte, falls es Daten gibt auf welche auch ein technischer Benutzer keinen Zugriff haben soll. Des Weiteren wäre es möglich verschiedene Policies für technische Benutzer anzulegen, wenn zum Beispiel ein «Batch»-User weniger Berechtigungen als der «admin»-User benötigt. Der Nachteil dieser Lösung ist, dass die Referenz auf die «admin»-Policy in allen vorhandenen *PolicySets* eingefügt werden muss, sofern der technische Benutzer wirklich überall den Zugriff erhalten soll. Diese Variante funktioniert allerdings nur, solange für einen XACML-Request nicht mehr als ein *PolicySet* evaluiert werden kann.

Im nächsten Abschnitt wird eine weitere Variante vorgestellt, bei welcher nicht zwingend *PolicySets* verwendet werden müssen. Der *Policy Combining Algorithm* ist dabei in der Konfiguration des PDP festgelegt. Der Vorteil dieser Variante ist, dass die Policy-Referenzen aus dem «admin»-Beispiel in Auflistung 4.14 nicht überall hinterlegt werden müssen. Dabei wäre es auch möglich mehrere *PolicySets* pro XACML-Request zu evaluieren.

«Root Policy Combining Algorithm»

Die Implementation eines PDP muss übergeordnet zu den in den *Policies* und *PolicySets* festgelegten *Combining Algorithms* einen «Root»-Algorithmus festlegen. Einerseits ist es möglich in den hinterlegten XML-Files direkt *Policies* ohne *PolicySets* zu definieren. Andererseits wäre es auch möglich, dass bei einem Request mehrere *PolicySets* evaluiert werden können, welche zu unterschiedlichen *Effects* führen. In diesen Fällen benötigt der PDP einen Algorithmus zur Auswahl des *Effects*, welcher nicht in den XACML-Dateien festgelegt werden kann. Dieser «Root Combining Algorithm» muss entsprechend im PDP definiert sein.

Die Tatsache, dass am Anfang der Implementationsphase des Prototypen nicht klar war wie dieser Algorithmus in Balana [23] konfiguriert werden kann, führte dazu, dass die Lösung mit den *PolicySets* gewählt wurde. Auf Nachfrage [27] bei der Open Source Community im Github-Projekt [17] von Balana [23], stellte sich heraus dass der «Root» Algorithmus des PDP durch die Implementation eines eigenen *PolicyFinderModules*, einer Klasse in Balana, konfiguriert werden kann. Der Vorteil dieser Variante wäre, dass die jetzigen Policy-Referenzen nicht in jedem *PolicySet* hinterlegt werden müssten und auch direkt mit *Policies* ohne *PolicySet* gearbeitet werden könnte. Der Nachteil ist allerdings, dass das Resultat abhängig von einer Konfiguration des PDP ist und das Verhalten nicht mehr vollständig durch die XACML-Dateien gesteuert wird. Die aktuell implementierte Variante basiert vollständig auf den Algorithmen in den XACML-Dateien. Solange sich die *Targets* der verschiedenen *PolicySets* nicht überschneiden und jeweils nur ein *PolicySet* pro Request in Frage kommt, ist dies die PDP-unabhängigere Variante.

4.3.4. Delegation Profiles

Eine weitere Herausforderung bei der Implementation des Prototypen war die User Story «US11: Vertretungen bei Absenzen berechtigen». Die Idee dieser Story ist es, bei der Abwesenheit eines bestimmten Benutzers, dessen Berechtigungen an einen Stellvertreter übergeben zu können. Diese Berechtigung soll aber zeitlich begrenzt sein. XACML 3.0 stellt für solche Anforderungen die Funktionalität der «Delegation Profiles» [56] bereit. Dabei erhält ein Benutzer das Recht, eigene Berechtigungen an andere Benutzer zu übergeben. Dazu muss vorerst eine Policy erstellt werden, welche dem Benutzer diese «Delegation» erlaubt. XACML 3.0 [56] kommt für die Implementation der «Delegation Profiles» mit den in Abbildung 4.7 gezeigten Sprachelementen aus. Es wird neben den Standard Attributkategorien *Object*, *Subject* und *Environment* lediglich eine neue Kategorie, *attribute-category:delegate*, benötigt. Es müssen daher an dieser Stelle keine zusätzlichen Sprachelemente eingeführt werden.

Übersicht «VIP»-Betreuer Vertretung

Im Prototyp wurde der Stellvertreter-Fall anhand des VIP-Beispiels aus Abschnitt 4.3.2 implementiert. Ein VIP-Betreuer soll in der Lage sein, seine Rechte, welche er durch die VIP-Policy erhalten hat, an andere Benutzer zu delegieren.

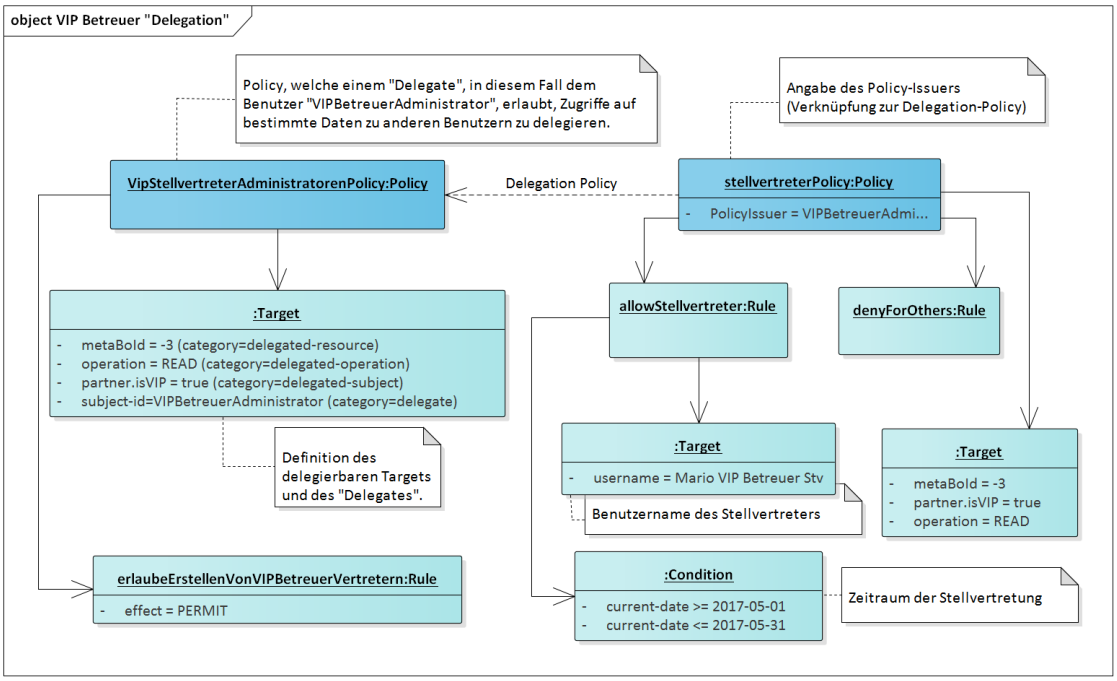

Abbildung 4.9.: Stellvertreterberechtigungen für VIP-Betreuer (UML Objektdiagramm)

Abbildung 4.9 zeigt, dass dafür in XACML eine «Delegation Policy» benötigt wird, welche es einem Benutzer erlaubt, bestimmte Rechte zu delegieren. In diesem konkreten Beispiel ist dies die «VipStellvertreterAdministratorenPolicy». Sie legt fest, dass der Benutzer «VIPBetreuerAdministrator» die «READ»-Berechtigung auf VIPs an andere Benutzer vergeben darf. Die Policy «stellvertreterPolicy», siehe Abbildung 4.9, legt dann den effektiven Stellvertreter fest. In diesem Fall fungiert der Benutzer «Mario VIP Betreuer Stv.» als Stellvertreter, welcher für den Zeitraum 01.05.2017 bis 31.05.2017 den Zugriff auf VIPs erhält. Durch den «PolicyIssuer» wird festgelegt, dass die Policy durch den Benutzer «VIPBetreuerAdministrator» ausgestellt wurde. Dadurch kann der PDP überprüfen, ob die Stellvertretung gültig ist.

Delegation Policy

Nachfolgend wird die «Delegation Policy», welche in Abbildung 4.9 als Objektdiagramm dargestellt wurde, in XACML aufgezeigt. Die Policy legt im *Target* fest, auf welche Objekte der VIP-Betreuer seine Rechte delegieren darf. Der VIP-Betreuer wird über die Attributkategorie *attribute-category:delegate* dazu berechtigt, Zugriffe auf diese Objekte zu delegieren, wie in Auflistung 4.15 ersichtlich ist.

```
<xacml3:AnyOf>
  <xacml3:AllOf>
    <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <xacml3:AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">VIPBetreuerAdministrator
        </xacml3:AttributeValue>
      <xacml3:AttributeDesignator
        AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
        DataType="http://www.w3.org/2001/XMLSchema#string"
        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:delegate"
        MustBePresent="false" />
    </xacml3:Match>
  </xacml3:AllOf>
</xacml3:AnyOf>
```

Auflistung 4.15: Delegation Profile: «Delegate User» festlegen

In diesem Beispiel aus Auflistung 4.15 wird der Benutzer «VIPBetreuerAdministrator» dazu ermächtigt, Berechtigungen für VIPs zu delegieren. Dafür muss das restliche *Target* der Policy die richtigen BOs (VIPs) und die erlaubte Operation angeben, wie in den Auflistungen 4.16 und 4.17 gezeigt wird.

```
<xacml3:AnyOf>
  <xacml3:AllOf>
    <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:integer-equal">
      <xacml3:AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#integer">-3</xacml3:AttributeValue>
      <xacml3:AttributeDesignator
        AttributeId="http://adcubum.com/syrius/bo/metaBoId"
        DataType="http://www.w3.org/2001/XMLSchema#integer"
        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:delegated:urn:oasis:names:
        tc:xacml:3.0:attribute-category:resource"
        MustBePresent="false" />
    </xacml3:Match>
  </xacml3:AllOf>
</xacml3:AnyOf>
```

Auflistung 4.16: Delegation Profile: *Target* für die «zu delegierenden» Objekte

```
<xacml3:AnyOf>
  <xacml3:AllOf>
    <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <xacml3:AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">READ</xacml3:AttributeValue>
      <xacml3:AttributeDesignator
        AttributeId="http://adcubum.com/syrius/operation"
        DataType="http://www.w3.org/2001/XMLSchema#string"
        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:delegated:urn:oasis:names:
        tc:xacml:3.0:attribute-category:action"
        MustBePresent="false" />
    </xacml3:Match>
  </xacml3:AllOf>
</xacml3:AnyOf>
<xacml3:AnyOf>
  <xacml3:AllOf>
    <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:boolean-equal">
      <xacml3:AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#boolean">true</xacml3:AttributeValue>
      <xacml3:AttributeDesignator
        AttributeId="http://adcubum.com/syrius/bo/partner/isVIP"
        DataType="http://www.w3.org/2001/XMLSchema#boolean"
        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:delegated:urn:oasis:names:
        tc:xacml:3.0:attribute-category:resource"
        MustBePresent="false" />
    </xacml3:Match>
  </xacml3:AllOf>
</xacml3:AnyOf>
```

Auflistung 4.17: Delegation Profile: *Target* für die «zu delegierenden» Objekte

Anhand einer einfachen zusätzlichen *Rule*, wie in Abbildung 4.18, wird der Benutzer «VIPBetreuerAdministrator» durch diese «Delegation»-Policy ermächtigt, Berechtigungen auf VIP-Partner an andere Benutzer zu vergeben.

```
<xacml3:Rule RuleId="erlaubeErstellenVonVIPBetreuerVertretern" Effect="Permit">
  <xacml3:Target />
</xacml3:Rule>
```

Auflistung 4.18: Delegation Profile: «Delegation»-*Rule*

Vertreter-Policy

Anschliessend wird für den Benutzer, welcher die Stellvertretung durchführen soll, eine gewöhnliche Policy («stellvertreterPolicy» in Abbildung 4.9) erstellt. Allerdings muss in dieser Policy dann der «Policy-Issuer» angegeben werden, wie in Auflistung 4.19 dargestellt.

```
<xacml3:PolicyIssuer>
  <xacml3:Attribute IncludeInResult="false" AttributeId="urn:oasis:names:tc:xacml:1.0:subject:
  subject-id">
    <xacml3:AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
      VIPBetreuerAdministrator
    </xacml3:AttributeValue>
  </xacml3:Attribute>
</xacml3:PolicyIssuer>
<xacml3:Description />
<xacml3:PolicyDefaults>
  <xacml3:XPathVersion>http://www.w3.org/TR/1999/REC-xpath-19991116</xacml3:XPathVersion>
</xacml3:PolicyDefaults>
```

Auflistung 4.19: Delegation Profile: «Policy Issuer»

Dadurch wird deklariert, dass die Rechte dieser Policy durch den Benutzer «VIPBetreuerAdministrator» vergeben wurden und der PDP kann überprüfen ob der «Issuer» die Berechtigung zur «Delegation» dieser Rechte besitzt. In dieser «Vertretungspolicy» wird neben dem «Issuer» wieder das *Target* und eine entsprechende *Rule* definiert, welche einem Stellvertreter über einen gewissen Zeitraum die Berechtigung erteilt. Das Beispiel einer solchen *Rule* aus dem Prototypen, wird in den Auflistungen 4.20 und 4.21 gezeigt.

```
<xacml3:Rule Effect="Permit"
    RuleId="http://adcubum.com/.../stellvertreterPolicySet/vipPolicy/allowStellvertreter">
  <xacml3:Description />
  <xacml3:Target>
    <xacml3:AnyOf>
      <xacml3:AllOf>
        <xacml3:Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <xacml3:AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">Mario VIP Betreuer Stv.
            </xacml3:AttributeValue>
          <xacml3:AttributeDesignator
            AttributeId="http://adcubum.com/syrius/syruser/username"
            DataType="http://www.w3.org/2001/XMLSchema#string"
            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
            MustBePresent="false" />
        </xacml3:Match>
      </xacml3:AllOf>
    </xacml3:AnyOf>
  </xacml3:Target>
```

Auflistung 4.20: Delegation Profile: *Target* für «Vertreter»-*Rule*

```
<!-- Zeitraum in der die Berechtigung angewendet wird. -->
<xacml3:Condition>
  <xacml3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <xacml3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
      <xacml3:Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:
      date-less-than-or-equal"/>
      <xacml3:AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#date">2017-05-01</xacml3:AttributeValue>
      <xacml3:AttributeDesignator
        AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date"
        DataType="http://www.w3.org/2001/XMLSchema#date"
        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment"
        MustBePresent="false" />
    </xacml3:Apply>
    <xacml3:Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
      <xacml3:Function FunctionId="urn:oasis:names:tc:xacml:1.0:function:
      date-greater-than-or-equal"/>
      <xacml3:AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#date">2017-05-31</xacml3:AttributeValue>
      <xacml3:AttributeDesignator
        AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date"
        DataType="http://www.w3.org/2001/XMLSchema#date"
        Category="urn:oasis:names:tc:xacml:3.0:attribute-category:environment"
        MustBePresent="false" />
    </xacml3:Apply>
  </xacml3:Apply>
</xacml3:Condition>
</xacml3:Rule>
```

Auflistung 4.21: Delegation Profile: *Conditions* für Zeitraum der «Vertreter»-*Rule*

In diesem spezifischen Beispiel wird dem Benutzer «Mario VIP Betreuer Stv.» die Berechtigung für den Zeitraum vom 01.05.2017 bis zum 31.05.2017 erteilt. Vor und nach diesem spezifischen Zeitraum evaluiert die entsprechende *Condition* zu «false».

Schwierigkeiten bei der Implementation mit Balana

Leider musste während der Implementation dieser User Story festgestellt werden, dass Balana [23] diese XACML 3.0 «Delegation Profiles» [56] nicht unterstützt. Ein entsprechender *Issue* [36] im Balana [23] Github-Projekt [17] wurde bereits erstellt. Der Open Source Community ist dies also bekannt. Für den Prototypen ist das Fehlen dieses Features nicht weiter problematisch, da die Ferienvertretungspolicy grundsätzlich funktioniert. Es wird lediglich der «Policy-Issuer» vom PDP nicht überprüft. Damit wird nicht sichergestellt, dass der Ersteller dieser Policy auch berechtigt ist diese «Delegation» durchzuführen. Der Zugriff für den stellvertretenden Mitarbeiter über die festgelegte temporäre Zeitperiode funktioniert aber.

Eine alternative Variante für Ferienvertretungen hat Axiomatics [5] in einem Blogeintrag [7] vorgestellt. Diese Lösung kommt ohne «Delegation Profiles» aus. Damit ist sie auch in XACML 2.0 umsetzbar. Der Vorteil der von Axiomatics [5] vorgeschlagenen Lösung

ist, dass keine zusätzlichen Policies für die einzelnen Vertreter erstellt werden müssen. Die Vertreter und die Zeitspannen für dessen Vertretungen, werden bei dieser Lösung im LDAP-Verzeichnis gepflegt und durch einen PIP ausgelesen. Der Versuch einer Implementation dieser Variante wurde während dieser Bachelorarbeit nicht gestartet. Auch wenn diese Variante auf den ersten Blick weniger Verwaltungsaufwand generiert, muss an dieser Stelle erwähnt werden, dass die Lösung mit den «Delegation Profiles» und den damit verbundenen zusätzlichen *Policies* keinen Mehraufwand im Vergleich zur aktuellen Lösung in SYRIUS bedeutet. Schon Heute wird jede Vertretung in SYRIUS manuell parametriert.

4.3.5. Alternative Policy-Syntax - ALFA

Obwohl wir durch den in Anhang B durchgeführten Benutzertest belegen konnten, dass XACML die NFA in Bezug auf die Einfachheit der Policy-Syntax erfüllt, gibt es Möglichkeiten um die Pflege der *Policies* zu erleichtern. Eine Möglichkeit um das XACML-Sprachmodell für Menschen leserlicher abzubilden wäre die Verwendung einer DSL. Wichtig ist dabei allerdings, dass aus den in der DSL geschriebenen Policies XACML generiert werden kann.

Auf Basis dieser Idee hat Axiomatics [5] die «Pseudocode»-Sprache ALFA [38] entwickelt, welche mittlerweile von OASIS [39] gepflegt wird. ALFA ist im Vergleich zu XACML wesentlich übersichtlicher und die Policies können schneller geschrieben und angepasst werden. ALFA basiert auf der selben abstrakten Syntax und somit den selben Sprachelementen wie XACML. Die Elemente aus Abbildung 4.7, welche Anfangs dieses Abschnitts gezeigt wurde, gelten somit auch für ALFA. Es ist lediglich die konkrete Syntax, welche vereinfacht wird. Axiomatics [5] bietet ausserdem ein Eclipse-Plugin [6] an, mit welchem die ALFA-Policies geschrieben und direkt in XACML-Dateien konvertiert werden können.

Das Eclipse-Plugin [6] darf für nichtkommerzielle Zwecke frei benutzt werden und konnte daher während dieser Bachelorarbeit verwendet werden. Da der Schreibaufwand in ALFA verglichen mit XACML kleiner ist, wurden die meisten Policies des Prototypen in ALFA geschrieben und das entsprechende XACML-File automatisch generiert. Da für den späteren «Make or Buy» Entscheid zu diesem Zeitpunkt noch keine Herstellerabhängigkeit geschaffen werden soll, sind die XACML-Dateien die entscheidenden Resultate dieser Arbeit.

Die folgende Auflistung 4.22 zeigt die bekannte VIP-Policy in ALFA.

```
namespace com.adcubum.authorization
{
  import com.adcubum.authorization.attributes.*
  policyset vipPolicySet {
    apply permitOverrides
    policy vipPolicy = "http://adcubum.com/authorization/boPolicies/vipPolicy" {
      target
        clause  bo.metaBoId == -3
        or      bo.metaBoId == -7
        or      bo.metaBoId == -1277
        or      bo.metaBoId == -50
        or      bo.metaBoId == -34
        or      bo.metaBoId == -104
        clause  operation.operationId=="READ"
        or      operation.operationId=="WRITE"
        clause  bo.partner.isVIP == true
      apply permitOverrides
      rule allowAccessVIPservice {
        condition syruser.abteilung == "VIPService"
        permit
      }
      rule denyForOthers {
        deny
      }
    }

    technicalUserPolicy
  }
}
```

Auflistung 4.22: VIP-Policy in ALFA

Die Attribute wie zum Beispiel *bo.metaBoId* müssen in einem separaten File gepflegt werden. Dies ist notwendig, da die Attributkategorie, der *Identifier* und der Typ des Attributs bekannt sein muss um aus der ALFA Policy wieder XACML generieren zu können. Abbildung 4.23 zeigt ein Beispiel einer solchen Attributdefinition für die MetaBO-Id.

```
attribute metaBoId {
    category = resourceCat
    id = "http://adcubum.com/syrius/bo/metaBoId"
    type = integer
}
```

Auflistung 4.23: Beispiel einer Attributdefinition in ALFA

Für Adcubum könnte eine solche alternative Policy-Syntax durchaus interessant sein, um die Benutzerfreundlichkeit und Wartbarkeit der Policies gegenüber XACML noch weiter zu steigern. Dies könnte mit einem kommerziellen Produkt, wie zum Beispiel dem Eclipse-Plugin [6] von Axiomatics [5], oder einer eigenen DSL erreicht werden. Das Entwickeln

einer eigenen DSL könnte die Benutzerfreundlichkeit bei der Pflege der Policies im Vergleich zu der Verwendung von ALFA noch weiter steigern, da diese an die Fachlichkeit des Versicherungswesens und an SYRIUS angepasst werden könnte.

Nachdem dieser Abschnitt die Policy-Syntax und die innerhalb des Prototypen entwickelten Policies erklärt hat, geht der nächste Abschnitt auf die Beschaffung der Informationen und Attribute ein, welche notwenig sind um diese Policies evaluieren zu können.

4.4. Informationsbeschaffung

Das Autorisierungssystem benötigt, wie bei den Policies im vorherigen Abschnitt bereits gesehen, Attribute um die Policies evaluieren zu können und für jeden Request eine Entscheidung zu treffen. In der ABAC-Referenzarchitektur [8, 41] spricht man von *PIPs*, oder *Attributrepositories*. Diese Attributrepositories können beliebige Softwaresysteme sein, welche Attribute zur Verfügung stellen. ABAC unterscheidet bei den Attributen zwischen den Kategorien *Objekt* und *Subjekt*. Neben den *Subjekt-* und *Objekt-* Attributen kennt das ABAC-Modell sogenannte *Umgebungsattribute* («Environment»), wie beispielsweise die Uhrzeit. Die PIPs stellen üblicherweise Attribute für eine dieser Kategorien zur Verfügung. *Subjekte* und *Objekte* werden über einen eindeutigen Schlüssel identifiziert. Die Attributrepositories, welche Attribute für die entsprechenden *Objekte* oder *Subjekte* bereitstellen, müssen diese Schlüssel kennen.

Die Anforderungsanalyse hat gezeigt, dass die nötigen Attribute um die User Stories zu implementieren, von den *Objekten* und *Subjekten* stammen. Die *Umgebung* ist für die Versicherungsfachlichkeit, abgesehen von zeitlichen Informationen wie dem aktuellen Datum, kaum relevant.

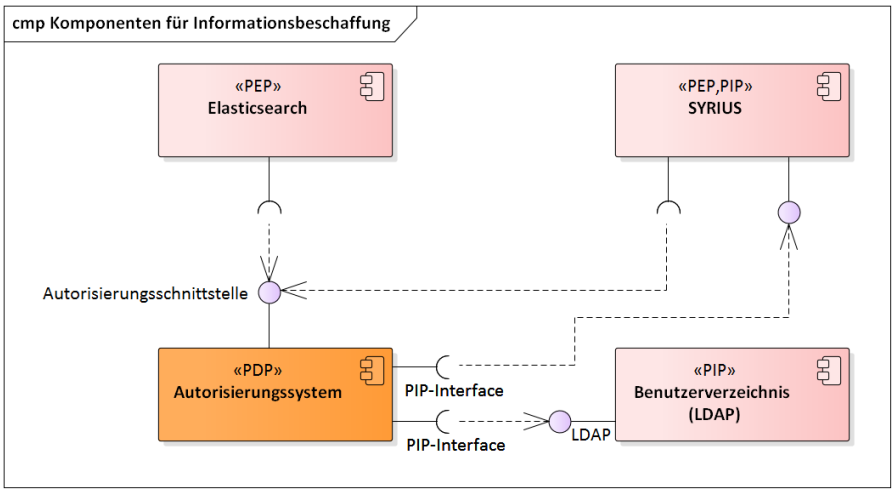

Abbildung 4.10.: PIPs für Informationsbeschaffung (UML Komponentendiagramm)

Die *Objekte* im Kontext dieser Arbeit, sind die BOs aus SYRIUS. Sie stammen somit alle aus dem selben System und werden über die BO-Id und MetaBO-Id identifiziert. In Zukunft wäre es denkbar, dass mehrere Systeme Attribute zu diesen BOs bereitstellen. Dazu muss aber jedes System über die eindeutigen Schlüssel (BO-Id / MetaBO-Id) verfügen. Die Attribute der *Subjekte* bzw. SYRIUS-Benutzer werden hingegen in einem Benutzerverzeichnis gespeichert. Bei der Anforderungsanalyse hat sich gezeigt, dass in der Praxis meist ein LDAP-Verzeichnis eingesetzt wird. Als LDAP-Verzeichnis kann eine beliebigen Implementation des LDAP-Protokolls [54] verwendet werden. Einige Implementationen

sind auf der Webseite von *LDAP.com* aufgelistet [31].

Für die Implementation der ermittelten User Stories sind demzufolge, wie in Abbildung 4.10 ersichtlich, zwei PIPs notwendig, nämlich SYRIUS und das Benutzerverzeichnis. In den nächsten zwei Abschnitten wird auf diese beiden Systeme genauer eingegangen. Dabei ist auch aufgeführt, welche Informationen diese zur Verfügung stellen müssen.

4.4.1. Benutzerverzeichnis (LDAP)

SYRIUS kann die Benutzerinformationen entweder in der eigenen Datenbank speichern, oder die Daten zu den Benutzern aus einem LDAP-Verzeichnis verwenden. Oftmals wird die Lösung mit einem LDAP-Verzeichnis von den Kunden bevorzugt, da die Benutzer innerhalb einer Versicherung für mehrere, von SYRIUS unabhängige, Softwaresysteme verwendet werden. Somit lassen sich die Informationen zentral pflegen. Dadurch müssen diese Daten nicht redundant gehalten werden. Das Autorisierungssystem bezieht aus dem LDAP-Verzeichnis alle Informationen, welche das *Subjekt*, also den Benutzer welcher auf ein BO zugreifen möchte, betreffen.

Dazu gehören die in der folgenden Tabelle 4.9 dargestellten Informationen. Konkret sind hier *Subjekt*-Attribute aufgelistet, welche in den erarbeiteten User Stories aus Kapitel 3 verwendet werden. Die Tabelle ist so gegliedert, da viele User Stories die gleichen Informationen eines *Subjektes* verwenden. Bei den Attributen der *Objekte*, sprich BOs aus SYRIUS, lässt sich diese Strukturierung nicht sinnvoll anwenden, da fast jede User Story eigene Attribute des Objekts verwendet.

Tabelle 4.9.: Informationen zum *Subjekt*

Information	**Beschreibung**	**Beispiele**	**User Stories**
Benutzer-Identifikator	Eindeutiger Schlüssel, um einen Benutzer zu identifizieren.	• Benutzername	US7 («admin»), US11
Benutzerrolle	Information darüber, welche Rollen und Funktionen der Benutzer in der Unternehmung hat.	• Sachbearbeiter/-in • Teamleiter/-in • Parametrierer/-in	US1, US3, US4, US5, US6, US7, US8, US10, US12, US14, US15, US16, US17, US18

Tabelle 4.9.: Informationen zum *Subjekt*

Information	Beschreibung	Beispiele	User Stories
Gruppen-zugehörigkeit	Information darüber, welchen Benutzergruppen der Benutzer angehört.	• Gruppe für Unterteilung geografischer Standorte • Gruppe für Unterteilung von unterschiedlichen Teams • Gruppe für Unterteilung von Experten pro Geschäftsbereich	US1, US2, US8, US9, US10, US15, US16

4.4.2. SYRIUS Datenbank

Die BOs auf welche der Benutzer zugreifen will, werden im Kontext von ABAC als *Objekte* bezeichnet. Dies können alle BOs aus SYRIUS sein, wie zum Beispiel Partner, Leistungen, OEs, oder Leistungserbringer. Alle für die Entscheidungsfindung, und somit für die entwickelten Policies, benötigten Attribute dieser Objekte, müssen dem Autorisierungssystem in Form eines PIP zur Verfügung gestellt werden.

Nachfolgend werden alle BOs und deren Attribute aufgelistet, welche für die Implementation der User Stories aus Kapitel 3 benötigt werden.

Alle beschriebenen Lösungen sind Implementationsvorschläge um eine Übersicht über die benötigten Informationen zu bekommen. Das *Attribute Engineering* [52] im Kontext von ABAC muss bei der Implementation des Autorisierungssystems wohlüberlegt durchgeführt werden, da es für jede Story verschiedene Umsetzungsvarianten gibt. Dabei muss die Wartbarkeit der entwickelten Lösung ein zentrales Interesse sein. Teilweise werden hier abgeleitete Attribute verwendet, welche nicht exakt in dieser Form im Datenmodell von SYRIUS vorhanden sind. Zum Beispiel gibt es das Attribut *Partner.isMitarbeiter* in SYRIUS nicht direkt, allerdings kann über eine Partnerrolle definiert und ermittelt werden, ob ein Partner auch ein Mitarbeiter ist. Der PIP oder der Mechanismus, welcher das *Attributrepository* des PIP befüllt, muss diese Logik des Auflösens der Partnerrolle implementieren.

Während in Tabelle 4.9 auf Seite der *Subjekte* viele User Stories die gleichen Attribute verwenden, kann diese Struktur in Tabelle 4.10 nicht beibehalten werden, da jede User Story unterschiedliche Attribute der Objekte verwendet. Deshalb ist die Tabelle nach User Stories und nicht nach Attributen gegliedert.

Tabelle 4.10.: Informationen zu den *Objekten*

User Story	**BOs**	**Attribute**
US1: VIP Kunden schützen	Partner inkl. aller abhängigen Objekte welche geschützt werden sollen.	• Partner.isVIP (*boolean*)
US2: Mitarbeiter schützen	Partner inkl. aller abhängigen Objekte welche geschützt werden sollen.	• Partner.isMitarbeiter (*boolean*), ist nicht direkt ein Attribut in SYRIUS (über Partnerrolle auflösen).
US3: Organisationseinheit schützen	Partner inkl. aller abhängigen Objekte welche geschützt werden sollen. BOs	• Partner.partnerTyp (Bedingung: *partnerTyp == 'Organisationseinheit'*)
US4: Konzernspezialist berechtigen	Konzern-BO, Partner-BOs aller Konzernmitglieder, inkl. deren abhängige Objekte welche geschützt werden sollen.	• Partner.isKonzernmitglied (*boolean*, gibt an ob ein Partner Teil eines Konzerns ist) • Partner.konzernBoId (*BOIdentifier*, Referenz zum Konzern) • Konzern.konzernspezialist (*UserIdentifier*, welche(r) angibt welche(n) Konzernspezialist(en) es zu einem Konzern gibt.)
US5: Konzernspezialist auf zukünftiges Konzernmitglied berechtigen	Konzern-BO, Partner-BOs aller Konzernmitglieder, inkl. deren abhängige Objekte welche geschützt werden sollen.	• Selbe Attribute wie bei US04 (siehe oben). • Die Attribute *Partner.isKonzernmitglied* und *Partner.konzernBoId* müssen durch den PIP bereits nach der Erstellung der Zuordnung bereitgestellt werden. Dadurch muss der zeitliche Aspekt nicht innerhalb der Policy abgehandelt werden.

Tabelle 4.10.: Informationen zu den *Objekten*

User Story	**BOs**	**Attribute**
US6: Alten und neuen Konzernspezialisten bei einem Konzernspezialisten-Wechsel zeitlich überschneidend berechtigen	Konzern-BO, Partner-BOs aller Konzernmitglieder, inkl. deren abhängige Objekte welche geschützt werden sollen.	• Selbe Attribute wie bei US04 (siehe oben). • Das Attribut *Konzern.konzernspezialist* muss in der zeitlichen Phase, in welcher der alte und der neue Konzernspezialist Zugriff haben, beide *UserIdentifier* zurückliefern. Der zeitliche Aspekt wird auch hier innerhalb der Logik des PIP abgebildet.
US7: Technischen User für Massenverarbeitungen berechtigen	Keine Daten aus SYRIUS nötig, solange die technischen Benutzer, wie im jetzigen Prototyp, Zugriff auf alle BOs haben. Die Policy basiert auf Regeln welche auf Benutzernamen oder Rollen basieren (Attribute des *Subjekts*).	• Keine Attribute aus der SYRIUS-Datenbank notwendig.
US8: Diagnosebezogene Daten schützen	In SYRIUS gibt es diverse BOs, welche diagnosebezogene Daten enthalten. Diese sind in der SYRIUS BO-Dokumentation [2] festgehalten und werden hier nicht explizit aufgelistet.	• siehe BO-Dokumentation [2]
US9: Partner aufgrund ihrer OE schützen	Partner inkl. aller abhängigen Objekte welche geschützt werden sollen.	• Partner.zugehoerigeOE (*BOIdentifier*, Referenz zu der zugehörigen OE)

Tabelle 4.10.: Informationen zu den *Objekten*

User Story	**BOs**	**Attribute**
US10: Postkörbe und Aufgaben schützen	Postkorb- und Aufgaben-(Activity) BO	• Hängt davon ab wie die Postkörbe und Aufgaben in der Versicherung organisatorisch strukturiert sind und in den Policies identifiziert werden müssen. • Möglichkeiten: *BOIdentifier*, Aufgabentyp, Bezeichnung des Postkorbs
US11: Vertretungen bei Absenzen berechtigen	Können alle BOs sein auf welche die stellvertretende Person Zugriff haben muss. Hängt daher vom fachlichen Fall ab. Bei Ferienvertretungen muss der stellvertretenden Person der Zugriff auf die unterschiedlichsten BOs gewährt werden können.	• Fachliche Attribute für den individuellen Fall. • Die Attribute *abwesendVon* und *abwesendBis* (Datum), werden im Prototyp in der Policy definiert.
US12: Leistungserbringer vor Mutation schützen	Leistungserbringer-BO und dessen abhängige Objekte, welche ebenfalls geschützt werden sollen.	• Partner.partnerTyp (Bedingung: *partnerTyp == 'Leistungserbringer'*)
US13: Vertriebspartner vor Mutation schützen	Vertriebspartner-BO und dessen abhängige Objekte, welche ebenfalls geschützt werden sollen.	• Partner.partnerRolle (Bedingung: *partnerRolle == 'Vertriebspartner'*)
US14: Berechtigungsverwaltung schützen	Alle BOs in SYRIUS welche in Zusammenhang mit Benutzer- oder Berechtigungsinformationen stehen. Zum Beispiel werden den *LDAP-Rollen* in die SYRIUS Datenbank synchronisiert.	• Eindeutiger Schlüssel (ID) der Berechtigungsobjekte.

Tabelle 4.10.: Informationen zu den *Objekten*

User Story	**BOs**	**Attribute**
US15: Vorbehalte schützen	Alle BOs in SYRIUS, welche Informationen zu Vorbehalten enhalten. Diese sind in der SYRIUS BO-Dokumentation [2] festgehalten und werden hier nicht explizit aufgelistet.	• siehe BO-Dokumentation [2]
US16: Leistungen aufgrund ihres Geschäftsbereichs schützen	Geschäftsbereich-BO, Leistungs-BOs und deren abhängige Objekte, welche ebenfalls geschützt werden müssen.	• Leistung.geschaeftsbereich (*BOIdentifier*, Referenz auf das Geschäftsbereich-BO)
US17: Betrag von Leistungsauszahlungen limitieren	Leistungs-BO auf welchem der Leistungsbetrag festgelegt wird.	• Leistung.betrag (Der vom Benutzer eingegebene Leistungsbetrag. Um diesen an das Autorisierungssystem zu übermitteln, müsste allenfalls noch die Autorisierungsschnittstelle angepasst werden.) • Der maximale Betrag, welcher ein Leistungsbearbeiter freigeben darf, muss auf dem *Subjekt* definiert werden. – Zum Beispiel: user.leistungslimite
US18: Sensible Bearbeitungsfälle schützen	*Case-Management*-BO	• case.typ (Typ des Bearbeitungsfalls) • Beispielbedingung: *typ == 'Versicherungsmissbrauch'*

Somit sollte aus diesem Kapitel die Architektur der neuen Autorisierungslösung, die Policies, und die benötigten Informationen einschliesslich der als PIP anzubindenden Systeme dokumentiert sein. Wie bei der Architektur des neuen Prototypen in Abschnitt 2.3.1 bereits angesprochen, nimmt die neue Autorisierungslösung als «Tradeoff» für die dynamischere Berechtigungsvergabe und die Entkopplung aus der SYRIUS Persistenzschicht, einen Performanceverlust in Kauf. Die im nächsten Kapitel 4.5 vorgeschlagenen Performanceoptimierungen zeigen auf, wie der Unterschied zu der bestehenden Lösung minimiert und die NFA «Performance» aus Abschnitt 3.3.1 eingehalten werden kann.

4.5. Vorgeschlagene Performanceoptimierungen

In den NFAs sind unter anderem Anforderungen an die Performance des neuen Autorisierungssystems festgehalten. Diese Aussagen wurden mithilfe eines Performancetests der bestehenden Lösung erarbeitet. Das genaue Vorgehen dieses Tests sowie die Resultate sind in Anhang A beschrieben. Die Ergebnisse des Performancetests zeigen auf, dass die bestehende Lösung bereits sehr performant arbeitet. Das hat vor allem damit zu tun, dass die Lösung mit der Datenbank keine «Remote»-Aufrufe benötigt. Da bei der neuen Lösung viel Wert auf die Entkopplung der Komponenten gelegt wurde, müssen Einbussen bei der Performance als «Tradeoff» der Entscheidung in Kauf genommen werden. In den NFAs ist deshalb ein Kostendach für die Performance der neuen Lösung definiert, damit diese Einbussen in einem akzeptierten Rahmen bleiben. In diesem Kapitel sind mögliche Performanceoptimierungen beschrieben, welche sich positiv auf die Performance der neuen Architektur auswirken, und somit helfen das angestrebte Performancekostendach mit der neuen Lösung einzuhalten.

Nr.	Titel	Auswirkung
1	Autorisierungssystem auf gleichem Node wie SYRIUS installieren	• Minimierung der Netzwerklatenz • Verkürzte Antwortzeiten
2	Entscheidungen in SYRIUS cachen	• Anzahl identischer Autorisierungsanfragen reduzieren • Reduzierter «Overhead» bei mehreren Anfragen mit gleichen Parametern
3	Mehrere Berechtigungsanfragen in einem Remote-Aufruf verarbeiten	• Anzahl benötigte «Remote»-Aufrufe reduzieren • «Overhead» bei direkt aufeinanderfolgenden Anfragen verkleinern
4	Lokales Attributrepository als PIP verwenden	• Weniger «Remote»-Aufrufe aus den PIPs • Datenstruktur des Attributrepository kann für Autorisierung optimiert werden

Tabelle 4.11.: Vorschläge für Performanceoptimierungen

Nachfolgend werden die in Tabelle 4.11 aufgelisteten Vorschläge für eine optimierte Performance näher erläutert. Neben einer Erklärung für eine Implementation sollen auch die Vorteile und Risiken aufgezeigt werden, um das Potential einzuschätzen.

4.5.1. Autorisierungssystem auf gleichem Node wie SYRIUS installieren

Die Performance der neuen Autorisierungslösung wird stark von dem gewählten Deployment abhängen. Mit einem geeigneten Deployment können die Aufrufzeiten für die Autorisierungsanfragen drastisch reduziert werden. In diesem Abschnitt werden zwei verschiedene mögliche Deploymentarchitekturen miteinander verglichen.

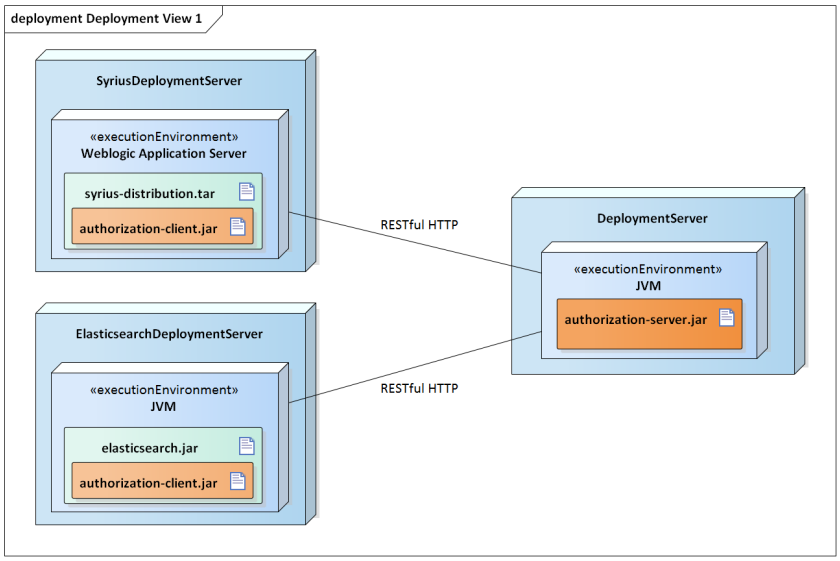

Abbildung 4.11.: Autorisierung auf eigenem Node [28] (UML Deploymentdiagramm)

In Abbildung 4.11 wird die Autorisierungskomponente auf einem eigenen «Node» ausgeführt. SYRIUS und Elasticsearch greifen über REST auf den Autorisierungsserver zu. Der Vorteil dieser Variante ist die dedizierte Umgebung, auf der die Autorisierungskomponente läuft. Die Performance wird nicht durch andere parallel laufende Programme gestört. Die Hardwareressourcen wie Central Processing Unit (CPU) und Random-access Memory (RAM) müssen nicht zwischen der Autorisierungskomponente und SYRIUS aufgeteilt werden.

Im Vergleich zu Abbildung 4.11 ist bei der Architektur in Abbildung 4.12 die Autorisierungskomponente auf dem gleichen «Node» installiert wie SYRIUS. Der grosse Vorteil dieser Umgebung ist, dass SYRIUS keine Aufrufe über das Netzwerk benötigt um Anfragen zu autorisieren. Für die Instanz von Elasticsearch ändert sich nichts. Die Schnittstellen und Protokolle bleiben gleich.

Aus Sicht der Performance und der Antwortzeiten ist die zweite Variante sicher zu bevorzugen. Durch das Deployment auf der gleichen Hardware wie SYRIUS, werden keine teuren Aufrufe über das Netzwerk benötigt. Der «Overhead» des Netzwerkprotokolls fällt dadurch geringer aus.

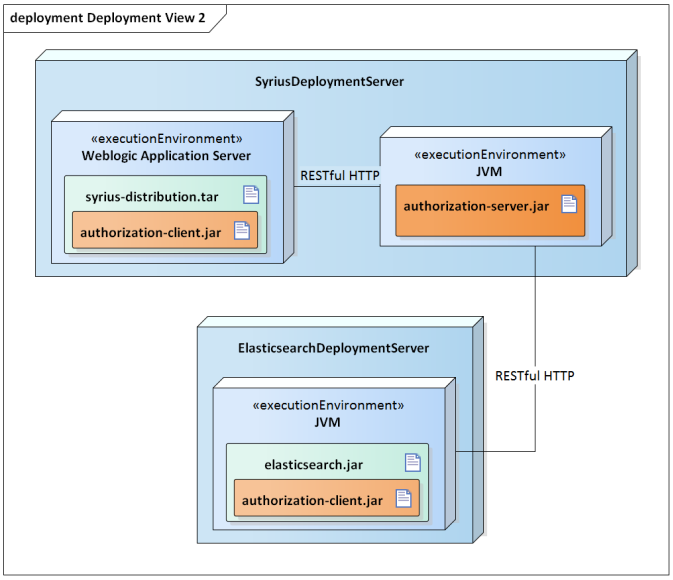

Abbildung 4.12.: Autorisierung auf geteiltem Node [28] (UML Deploymentdiagramm)

Es darf deswegen mit einer verkürzten Round Trip Time (RTT) gerechnet werden. Da diese zusätzlichen Performancekosten bei jeder Autorisierungsanfrage entstehen würden, ist die eingesparte Zeit nicht zu unterschätzen. Allerdings teilt sich die Autorisierungslösung bei diesem Deployment die Hardwareressourcen mit SYRIUS. Dies könnte potentiell einen Engpass bei vielen Autorisierungsanfragen von externen Systemen zur Folge haben.

4.5.2. Entscheidungen in SYRIUS cachen

In SYRIUS werden über eine Benutzersession verteilt, oft mehrere Zugriffe auf dieselben BOs gemacht, zum Beispiel den gleichen Partner. Diese doppelten Autorisierungsanfragen können optimiert werden. Dazu benötigt SYRIUS einen Cache für die evaluierten Autorisierungsanfragen.

Wenn die Resultate von häufig auftretenden Requests bei SYRIUS zwischengespeichert werden, könnten redundante Anfragen an die Autorisierungskomponente verhindert werden. Der SYRIUS-eigene Cache kann somit wiederkehrende Autorisierungsanfragen direkt berechtigen. Somit entfällt die komplette Entscheidungsfindung auf Seite der Autorisierungskomponente. In der jetzigen Lösung ist diese Optimierung bereits implementiert.

Dieser Mechanismus lässt sich allerdings nur anwenden, wenn die Antwort der Autorisierungskomponenten über einen längeren Zeitraum identisch ausfällt. Sobald die Berechtigungsentscheidung abhängig ist von Daten, welche sich regelmässig ändern, als Beispiel könnten hier momentane Uhrzeit oder der Status einer Aufgabe dienen, kann diese Optimierung nicht verwendet werden. Bei Autorisierungsentscheidungen, welche auf sich

häufig ändernden Daten beruhen, kann es somit vorkommen, dass bei einer Berechtigungsanfrage eine veraltete Entscheidung aus dem Cache verwendet wird, während die Autorisierungskomponente, aufgrund der aktualisierten Daten, diese Anfrage anders evaluiert hätte.

4.5.3. Mehrere Berechtigungsanfragen in einem Remote-Aufruf verarbeiten

Die Performanceanalyse der bisherigen Lösung hat ergeben, dass für einen Task, wie zum Beispiel dem Öffnen eines Partners, in SYRIUS über 100 Datenbankanfragen abgesetzt werden können. Wird nun für jede dieser SQL-Queries ein Aufruf an den Autorisierungsservice gestartet, kann die Performance der jetzigen Lösung nicht erreicht werden.

Eine mögliche Optimierung ist deshalb das Bündeln dieser Anfragen. Das Öffnen eines Partners führt laut der Performanceanalyse in Anhang A 34 SQL-Queries aus, welche durch den Objektschutz betroffen sind. Können nun alle oder ein Teil dieser Berechtigungsanfragen zu einem entfernten Aufruf an die Autorisierungskomponente zusammengefasst werden, kann dadurch die Anzahl der benötigten entfernten Aufrufe reduziert werden. Ein möglicher Ablauf mit der umgesetzten Optimierung ist in Abbildung 4.13 dargestellt.

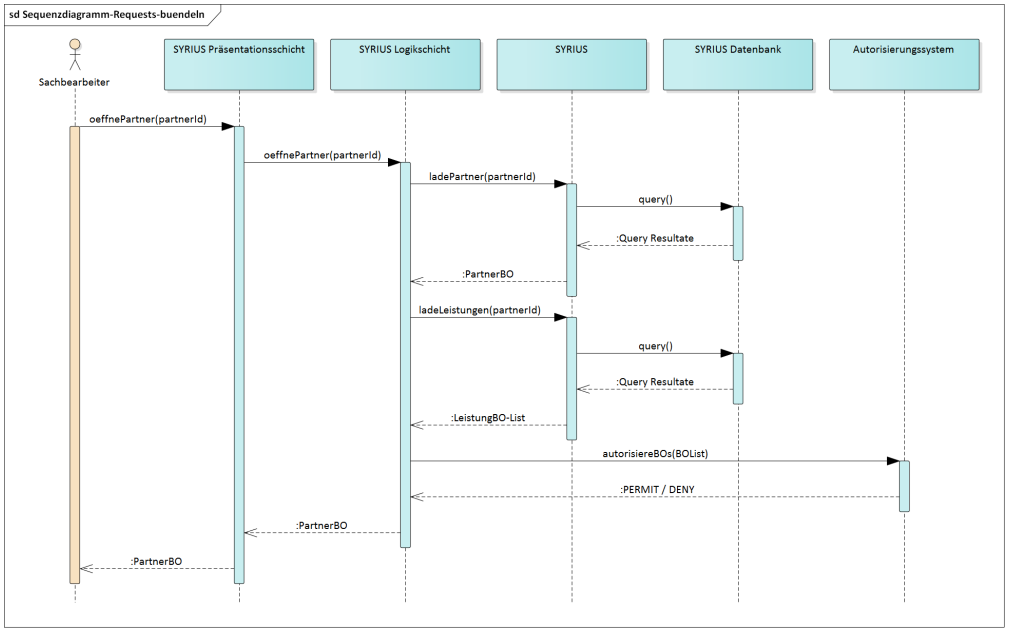

Abbildung 4.13.: «Request-Bundling» (UML Sequenzdiagramm)

Allerdings benötigt diese Optimierung eine grössere Anpassung in SYRIUS. Bis jetzt wird der Objektschutz bei jedem Aufruf aus der Persistenzschicht überprüft. Die Anfra-

gen müssen allerdings bereits in der Logikschicht zusammengefasst werden. Dies würde einen grösseren Umbau in SYRIUS verursachen. Durch die Verlagerung der Autorisierung aus der Persistenzschicht in die Logikschicht kann ein Sicherheitsrisiko entstehen, da nicht mehr jede einzelne SQL-Abfrage in der Persistenzschicht autorisiert wird. Optimal wäre eine Lösung, in welcher die Autorisierungsanfragen in der Logikschicht zusammengetragen werden, jedoch die Ausführung der Autorisierung in der Persistenzschicht bleibt. Somit kann, wie in der jetzigen Lösung, sichergestellt werden, dass jeder Datenbankzugriff autorisiert ist. Da die Optimierung auf der Seite des PEP vorgenommen wird, muss jeder PEP die Optimierung individuell umsetzen. Dies bringt je nach Anzahl verschiedener PEPs einen grossen Aufwand mit sich.

4.5.4. Lokales Attributrepository als PIP verwenden

Laut dem Referenzmodell von ABAC kann ein PDP mehrere PIPs verwenden. Wie in Kapitel 4.4 beschrieben ergeben sich für die neue Autorisierungslösung in SYRIUS zwei verschiedene PIPs, welche die vom PDP benötigten Daten liefern. Dies sind die SYRIUS Datenbank, sowie ein Benutzerverzeichnis. Diese Komponenten werden auf verschiedenen Nodes betrieben. Die Autorisierungskomponente muss nun bei jeder Autorisierungsanfrage die passenden Informationen aus den beiden oben genannten PIPs beziehen. Beide PIPs erfordern einen Aufruf über das Netzwerk. Aufrufe über das Netzwerk sind, was die Performance angeht, immer teurer als ein Aufruf innerhalb des gleichen «Node». Im Systemsequenzdiagramm in Abbildung 4.14 ist der Ablauf eines Autorisierungsrequests über die beiden PIPs dargestellt. Das Autorisierungssystem muss für die Informationen über den Partner zwingend über SYRIUS zugreifen, da direkte Zugriffe auf die SYRIUS Datenbank von fremden Systemen zu unterlassen sind.

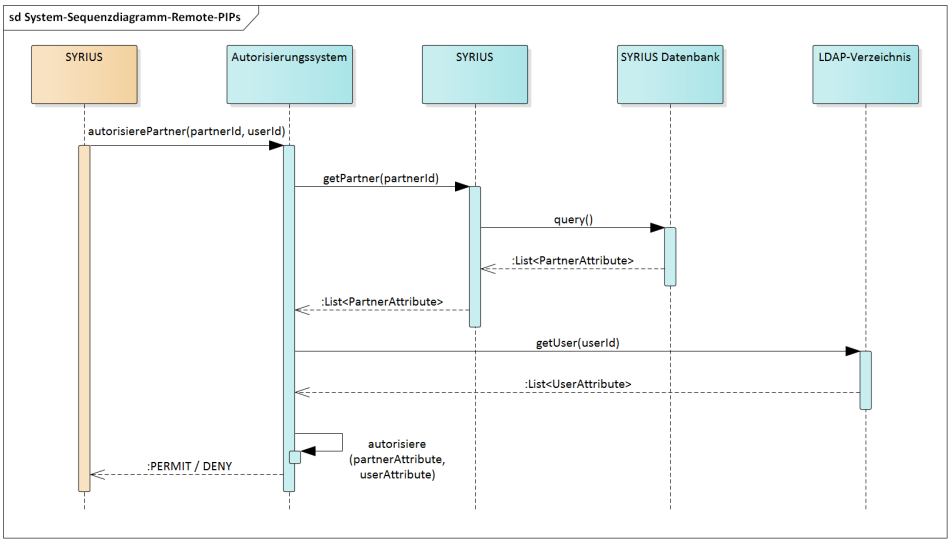

Abbildung 4.14.: Autorisierung mit verteilten *Attributrepositories* (UML Sequenzdiag.)

Eine mögliche Lösung dieses Problems ist die Erstellung eines eigenen lokalen Attributrepository, in Form einer Datenbank. Dadurch kann die Performance deutlich verbessert werden, da nun bei einer Autorisierungsanfrage anstelle der beiden Remote-Calls zu SYRIUS und dem Benutzerverzeichnis, ein Aufruf des Attributrepositories genügt. Dieses Szenario ist in Abbildung 4.15 dargestellt. Der Aufruf vom Autorisierungssystem an die eigene Datenbank kann durch ein Deployment auf dem gleichen Node als lokaler Aufruf ohne Verbindung über das Netzwerk angesehen werden. Dadurch müssen die Daten nur über das lokale *Loopback-Interface* und nicht über das LAN-Netzwerk geschickt werden.

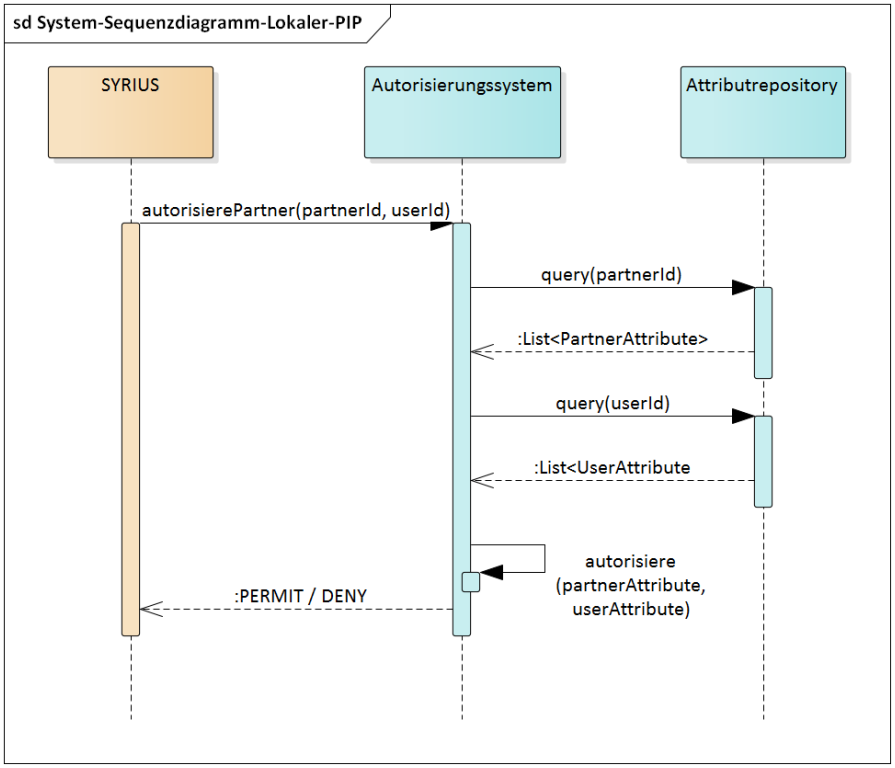

Abbildung 4.15.: Autorisierung mit lokalem *Attributrepository* (UML Sequenzdiagramm)

Ein weiterer Vorteil dieser Lösung ist die Exklusivität des Attributrepository. Einzig die Autorisierungskomponente benutzt diese Datenbank. So kann das Design der Datenstruktur genau so ausgearbeitet werden, wie dies für die Autorisierung benötigt wird.

Allerdings zieht diese Optimierung mit einem lokalen Attributrepository die Anforderung nach sich, das lokale Attributrepository mit Daten aus der SYRIUS Datenbank und dem LDAP-Verzeichnis zu befüllen und die Daten aktuell zu halten. Auf diese Thematik wird im folgenden Abschnitt eingegangen.

4.5.5. Synchronisierung eines lokalen Attributrepositories

Dieser Abschnitt geht genauer auf die vorgeschlagene Datensynchronisation aus dem Abschnitt 4.5.4 ein. Es werden zwei unterschiedliche Szenarien, *Push* und *Pull*, zur Synchronisierung der Daten erläutert. In beiden Szenarien ist es notwendig, auf allen beteiligten Nodes eine Synchronisationskomponente zu installieren, welche sich um den Abgleich der Daten kümmert. In Abbildung 4.16 ist dies in einem Komponentendiagramm aufgezeigt.

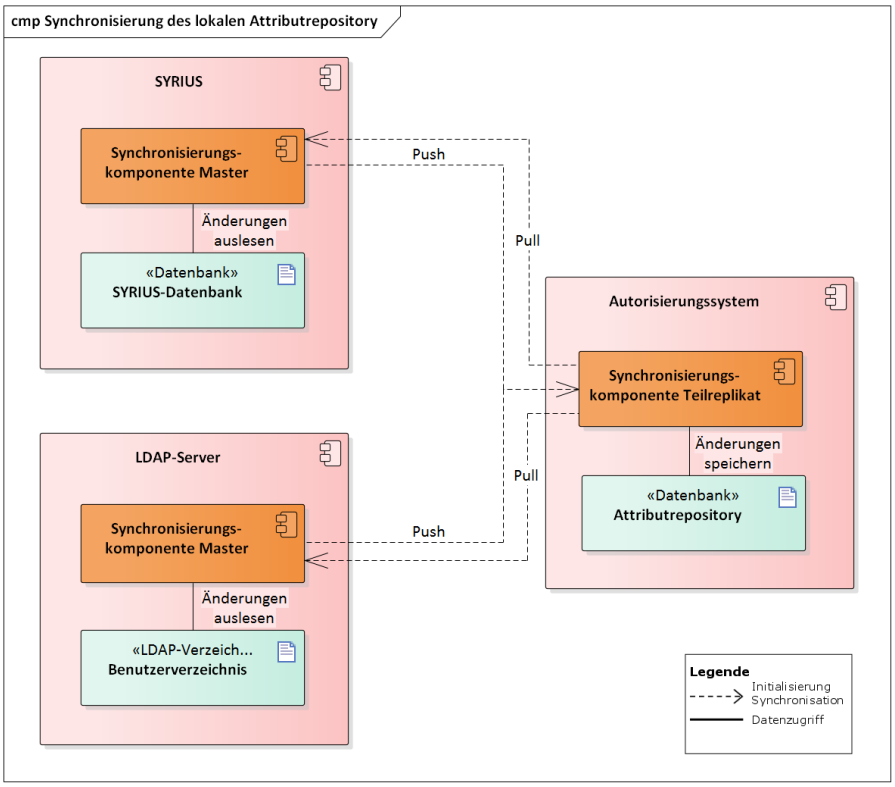

Abbildung 4.16.: Synchronisation der Datenquellen (UML Komponentendiagramm)

In den folgenden Beschreibungen wird bei den Datenquellen zwischen *Master* und *Teilreplikat* unterschieden. Als Master werden die in Kapitel 4.4 beschriebenen Datenquellen, SYRIUS Datenbank und LDAP-Verzeichnis, bezeichnet. Das in Abschnitt 4.5.4 vorgeschlagene lokale Attributrepository ist in diesem Szenario das Teilreplikat. Der Begriff «Teilreplikat» wurde deshalb gewählt, weil ein Attributrepository lediglich einen kleinen Teilbestand der Daten des Masters bei sich hält. Die Master versorgen das Teilreplikat mit den benötigten Attributen. Diese Synchronisierung geschieht nur in eine Richtung. Änderungen an den Daten werden durch Drittapplikationen, wie SYRIUS oder eine Benutzerverwaltung, nur auf den Master-Datenquellen gemacht. Das Teilreplikat wird ausschliesslich über die Synchronisierungskomponente verändert.

Push-Szenario

Bei einem Push-Szenario informiert der Master das Teilreplikat über angefallene Änderungen an den Daten. Das Teilreplikat übernimmt diese dann im eigenen Datenbestand. Der Zeitpunkt des Pushes wird vom Master bestimmt, typischerweise sobald sich die Daten des Masters ändern. Der Push kann aber auch per Timer gesteuert werden. Der Ablauf eines solchen Push-Szenarios ist in Sequenzdiagramm 4.17 aufgezeigt.

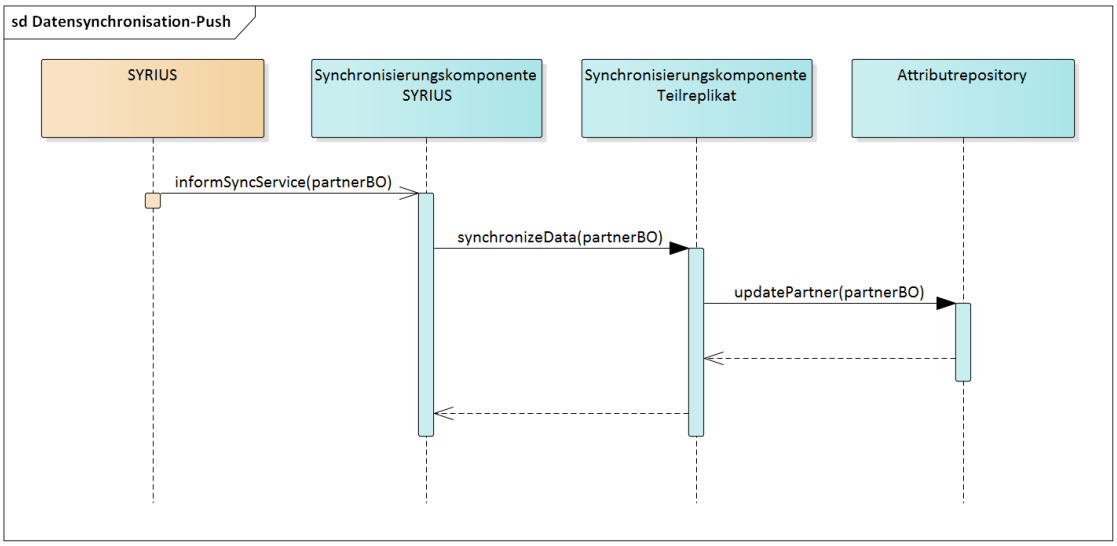

Abbildung 4.17.: Systemsequenzdiagramm des Push-Szenario (UML Sequenzdiagramm)

Dieses Szenario eignet sich besonders für Daten, welche sich häufig ändern, oder zwingend zu jeder Zeit im Teilreplikat aktuell sein müssen. Für die garantierte Aktualität der Daten muss der Preis des vergleichsweise hohen Netzwerkverkehrs in Kauf genommen werden.

Pull-Szenario

Im Pull-Szenario geht die Anfrage vom Teilreplikat aus. Das Teilreplikat fragt die beiden Master nach allfälligen Änderungen. Als Antwort können entweder die kompletten Daten, oder zum Beispiel auch nur die Differenz seit der letzten Anfrage übermittelt werden. Dies hängt von der Implementation der Synchronisierungskomponente ab. In Abbildung 4.18 ist dieses Szenario anhand eines Abgleichs der Daten aus der SYRIUS Datenbank aufgezeichnet. Der gleiche Ablauf würde bei einem Abgleich der Daten aus dem LDAP-Repository entstehen.

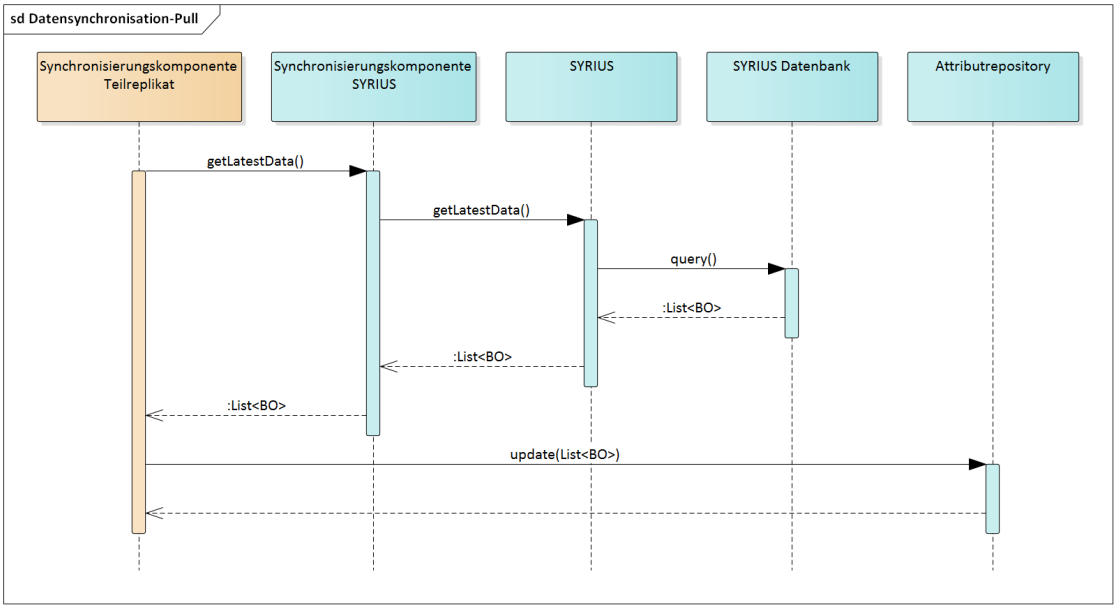

Abbildung 4.18.: Systemsequenzdiagramm des Pull-Szenario (UML Sequenzdiagramm)

Der Vorteil dieser Art der Synchronisierung ist, dass die Kontrolle beim Teilreplikat liegt. Das Teilreplikat kann steuern wie oft die Daten aktualisiert werden sollen. Dies hat aber den Nachteil dass Änderungen am Master nicht sofort übernommen werden. Dieses Szenario bietet sich daher an, wenn die Daten des Masters keine häufige Änderung erfahren.

In der Praxis ist es oft sinnvoll eine Mischung aus beiden Szenarien zu verwenden. Während Daten, welche sich häufig ändern, vorzugsweise über ein Push-Szenario synchronisiert werden, kann es in anderen Fällen Sinn machen, die Kontrolle über den Zeitpunkt der Synchronisierung mit einem Pull-Szenario bei dem Teilreplikat zu belassen. Abschliessend lässt sich keine direkte Empfehlung für eines der beiden Szenarien abgeben, da auf beide Arten die gewünschten Daten synchronisiert werden können. Die Entscheidung für oder gegen ein Szenario ist sehr stark von dem konkreten Anwendungsfall abhängig.

4.5.6. Bewertung der Optimierungen - Chancen und Risiken

Nachdem nun alle Vorschläge für Performanceoptimierungen vorgestellt wurden, gibt nachfolgende Tabelle 4.12 einen Überblick über die Chancen und Risiken der einzelnen Vorschäge, da solche Optimierungen immer Vor- und Nachteile mit sich bringen.

Tabelle 4.12.: Chancen und Risiken der Performanceoptimierungen

Optimierung	**Bewertung**
Autorisierungssystem auf gleichem Node wie SYRIUS installieren	**Chancen:** • Kein Performanceverlust aufgrund Netzwerklatenz. • Keine «Remote»-Aufrufe. **Risiken:** • System auf welchem SYRIUS läuft wird durch zusätzliches System belastet und muss evtl. ausgebaut werden. • Andere Systeme welche das Autorisierungssystem aufrufen müssen, brauchen nach wie vor «Remote»-Calls (z.B. Elasticsearch). • Dies könnte dazu führen, dass das Autorisierungssystem auf mehrere Nodes verteilt lauffähig sein muss, was eine komplexere Architektur nach sich zieht.
Entscheidungen in SYRIUS cachen	**Chancen:** • Reduzierung wiederholter Autorisierungsrequest mit gleichen Parametern und selber Antwort. **Risiken:** • SYRIUS müsste dazu wissen, welche Requests auf Daten basieren die sich nicht häufig ändern bzw. bei welchen Requests diese Optimierung durchgeführt werden darf. • Damit würde SYRIUS wieder Informationen enthalten, welche eigentlich nur das Autorisierungssystem betreffen sollen (Verletzung des «Single Responsibility» Prinzips). • Risiko, dass Entscheidungen teilweise veraltet sind.
Mehrere Berechtigungsanfragen in einem Remote-Aufruf verarbeiten	**Chancen:** • Deutliche Reduktion der Anzahl Autorisierungsanfragen. **Risiken:** • Grösserer Architekturumbau innerhalb von SYRIUS notwendig. • Autorisierung eventuell nicht mehr direkt in Persistenzschicht durchführbar.

Tabelle 4.12.: Chancen und Risiken der Performanceoptimierungen

Optimierung	Bewertung
Lokales Attributrepository als PIP verwenden	**Chancen:** • Keine «Remote»-Calls für Informationsbeschaffung. • Wesentlich performantere Entscheidungsfindung des PDP. **Risiken:** • Zusätzliche Komplexität und Aufwand für die Synchronisation der Attribute. • Risiko, dass Attribute veraltet sind.

Die vorgeschlagenen Performanceoptimierungen können helfen um die neue Architektur möglichst performant zu gestalten. Allerdings beinhalten alle Vorschläge auch gewisse Risiken. Welche Vorschläge bei der Implementation tatsächlich umgesetzt werden ist der Adcubum überlassen. Als letztes Thema behandelt dieser Bericht im folgenden Kapitel 4.6 die Möglichkeiten einer Migration des alten Systems zu dem neuen ABAC-Konzept.

4.6. Migrationsmöglichkeiten der bestehenden Berechtigungslösung

Nachdem in den letzten Kapiteln die Umsetzung des ABAC-Konzepts und mögliche Optimierungen an der Performance behandelt wurden, behandelt dieses Kapitel die Möglichkeiten einer automatisierten oder teilautomatisierten Migration der bestehenden Berechtigungslösung in SYRIUS, zu der neuen ABAC-basierten Lösung. Dazu müssen die alten Berechtigungsdefinitionen in XACML-Policies umgewandelt werden.

4.6.1. Automatische Generierung von Policies

Aus einer Role-Based Access Control (RBAC)-Berechtigung automatisch ABAC Policies zu generieren ist sehr aufwändig, und automatisiert fast nicht zu bewerkstelligen. Es gibt einen Ansatz um XACML-Policies automatisch generieren zu lassen. Dieser ist in einem wissenschaftlichen Paper von Zhongyuan Xu und Scott D. Stoller [62] beschrieben. Allerdings gibt es dazu bis jetzt keine öffentlich verfügbare Implementierung. Eine weitere Problematik, welche einer automatisierten Migration bei SYRIUS im Weg stehen würde, ist die Ausgangslage, dass die jetzige Lösung mit den Schutzobjekten und Berechtigungsdefinitionen von dem klassisches RBAC-Modell, welches nur auf Benutzern und deren Rollen basiert, abweicht. Neben den zu generierenden Policies müssen auch

die in den Policies verwendeten Attribute erarbeitet und in den PIPs bereitgestellt werden. Der ganze Prozess des Attribute Engineering, dem Erarbeiten der Attribute, müsste ebenfalls automatisiert werden. In der jetzigen Berechtigungslösung werden viele Rechte über Berechtigungsobjekte vergeben, welche mit den zu berechtigenden BOs verknüpft sind. Um aus diesen künstlich angelegten Berechtigungsinformationen auf «fachliche» ABAC-Attribute zu schliessen, braucht es viel Wissen aus der Fachdomäne und über die bestehende SYRIUS-Parametrierung. Diese Argumente sprechen deutlich für ein manuelles Attribute Engineering. Dazu kommt die individuelle Parametrierung jedes Kunden und die benötigten Attribute, welche bei jeder Kundeninstanz von SYRIUS unterschiedlich ausfallen können. Aufgrund all dieser Argumente ist eine eigene Entwicklung von PIP-Attributen und Policies einer automatischen Generierung vorzuziehen.

4.6.2. Migration der SYRIUS Standard-Berechtigungsdefinitionen

Mit der jetzigen Version von SYRIUS werden vier Standard-Berechtigungsdefinitionen mitgeliefert. Dies sind die Folgenden:

- Mitarbeiterschutz
- OE-Schutz
- Partnerschutz
- Leistungserbringerschutz

Viele Kunden verwenden diese Standard-Berechtigungsdefinitionen, und oft wird dadurch auch bereits ein grosser Teil des Objektschutzes abgedeckt. Dies bestätigte sich auch in den Interviews mit den Kunden. Da diese Berechtigungsdefinitionen bei Adcubum entwickelt wurden, gibt es die Möglichkeit bei einer Migration auf die neue ABAC-basierte Lösung, entsprechende Standard-Policies mitzuliefern. Alle für die Standard-Berechtigungsdefinitionen notwendigen Informationen, wie zum Beispiel die Information ob ein Partner als Leistungserbringer gilt, sind bereits in der Datenbank von SYRIUS vorhanden. Mit den vorhandenen Informationen lassen sich entsprechende Policies erarbeiten, welche dann als Standard mit der neuen Autorisierungslösung mitgeliefert werden können. Die für diese Standard-Policies benötigten Informationen aus der SYRIUS-Datenbank, müssen allerdings durch die manuelle Implementation eines PIP bereitgestellt werden. Zudem kommt es in der Praxis häufig vor, dass Kunden von Adcubum die mitgelieferten Berechtigungsdefinitionen anpassen. In einem solchen Fall kann die neue Standard-Policy als Vorlage genommen werden, welche dann gemeinsam mit dem Kunden angepasst werden kann.

4.6.3. Migration von kundenspezifischen Policies

Viele Kunden parametrieren mit der jetzigen Lösung von SYRIUS eigene Berechtigungsdefinitionen zusätzlich zu den von Adcubum ausgelieferten Standartdefinitionen. Die grosse Schwierigkeit bei der Migration von diesen kundenspezifischen Berechtigungsdefinitionen ist, dass oft nicht klar ist, aufgrund von welchen fachlichen Attributen die Berechtigungen vergeben werden. Oft werden die in SYRIUS zur Verfügung stehenden Berechtigungsdefinitionen verwendet. Allerdings werden die BOs dadurch über die Berechtigungsobjekte kategorisiert und nicht, wie im Konzept von ABAC vorgesehen, durch die fachlichen Attribute der BOs autorisiert. Eine Transformation der von den Kunden selbst erstellten Berechtigungsdefinitionen in ABAC-Policies benötigt sehr viel Kundenspezifisches Wissen, und kann nicht direkt von einem Kunden auf Andere übertragen werden. Somit wird von einer automatisierten Migration dieser kundenspezifischen Berechtigungsdefinitionen abgeraten.

4.6.4. Migrationsvorgehen

Aus den letzten Abschnitten geht hervor, dass eine automatisierte Migration sehr schwierig wird, und nur beschränkt einen Nutzen bringt. Aber auch mit der automatisierten Generierung von Policies und Attributen, muss die neue Autorisierungskomponente vor der Einführung bei jedem Kunden ausgiebig getestet werden, damit sichergestellt werden kann, dass die neue Implementation sich identisch verhält wie die Bestehende. Das Problem ist hierbei, dass die jetzigen Parametrierungen bei jedem Kunden unterschiedlich sind, und somit nicht vor der Auslieferung ein abschliessender Test durchgeführt werden kann.

Damit der Kunde die neue Autorisierungslösung ausgiebig testen kann, muss in SYRIUS auf eine einfache Art zwischen dem alten und dem neuen Autorisierungsmechanismus umgeschalten werden können. Somit können die effektiven Ergebnisse der beiden Lösungen direkt auf der Kundeninstanz miteinander verglichen werden. Dazu kann ein in der Studienarbeit [28] aufgegriffenes Refactoring, welches auf dem Konzept von Aspektorientierte Programmierung (AOP) basiert, verwendet und ergänzt werden. Die Idee des in der Studienarbeit erwähnten Refactoring ist, die Logik des bestehenden Objektschutzes aus der restlichen Persistenzschicht zu extrahieren und hinter einem eigenen Interface zu kapseln. In Abbildung 4.19 ist an dem Beispiel des *GenericStatementGenerator* ersichtlich wie die beiden Implementationen der Autorisierung austauschbar verwendet werden können.

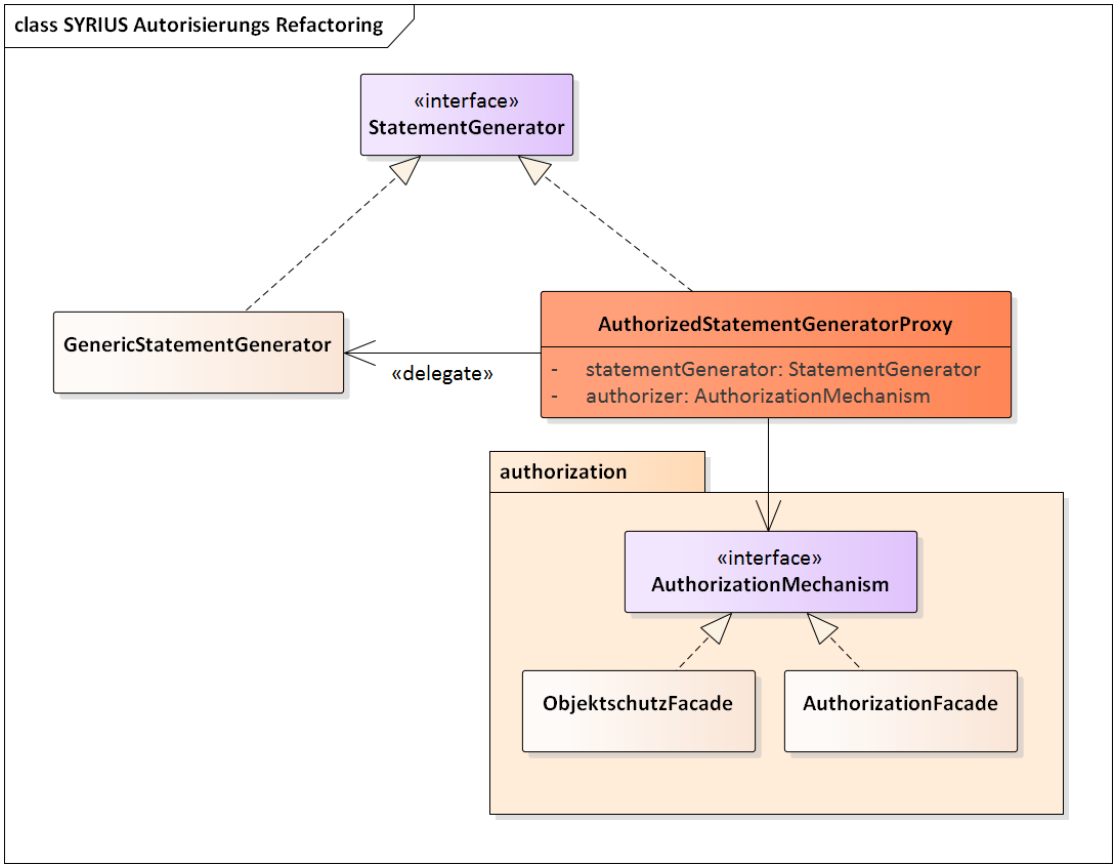

Abbildung 4.19.: «Redesign» für austauschbare Autorisierung (UML Klassendiagramm)

Durch den Einsatz des *Proxy-Patterns* [16, 60] ist es möglich den *AuthorizedStatementGeneratorProxy* anstelle des *GenericStatementGenerators* zu verwenden, und eine Implementation des austauschbaren Autorisierungsmechanismus zu hinterlegen. Der *GenericStatementGenerator* wird als «Delegate» hinterlegt und dessen Resultate mittels dem Autorisierungsmechanismus gefiltert. Damit würde die Autorisierung an einer zentralen Stelle aufgerufen und die Autorisierungsmechanismen, welche das Interface *AuthorizationMechanism* implementieren, könnten dynamisch im *AuthorizedStatementGeneratorProxy* konfiguriert werden. *ObjektschutzFacade* steht für die jetzige Implementation, *AuthorizationFacade* ist als Fassade für den Zugriff auf das neue Autorisierungssystem zu verstehen. Da beide Klassen das Interface *AuthorizationMechanism* implementieren, kann die tatsächliche Implementation in *AuthorizedStatementGeneratorProxy* beliebig ausgetauscht werden.

Benötigte Änderungen in SYRIUS

Damit ein solches Setup für die Migration möglich ist, müssen doch einige nicht ganz triviale Änderungen in SYRIUS gemacht werden. Wie bereits in der Studienarbeit [28] angetönt muss der gesamte Objektschutz zuerst von der restlichen Logik der Persistenzschicht getrennt und durch Interfaces entkoppelt werden. Dies könnte sich als Herausforderung entpuppen, da der Objektschutz bis jetzt über die Persistenzschicht verteilt an mehreren Orten umgesetzt ist.

Die Migration wie auch die Umsetzung der neuen Architektur ist Teil eines Projektes bei Adcubum. Diese Bachelorarbeit legt die Grundlagen für das weitere Gelingen dieses Projektes. Die Ergebnisdiskussion fasst nun die Resultate dieser Arbeit zusammen. Dazu bewertet das nächste Kapitel ob die zu Beginn definierten NFAs erfüllt wurden. Zuletzt gibt Abschnitt 5.4 einen Ausblick über die Weiterverwendung dieser Bachelorarbeit im Rahmen des weiterlaufenden Projektes bei Adcubum.

5. Ergebnisdiskussion

Dieses letzte Kapitel fasst die Resultate dieser Bachelorarbeit zusammen und bewertet diese. Ausserdem werden die aus der Arbeit gezogenen Schlussfolgerungen für zukünftige Folgeprojekte erläutert.

5.1. Kritische Erfolgsfaktoren dieser Bachelorarbeit

In der Aufgabenstellung, welche in Anhang C abgelegt ist, wurden die kritischen Erfolgsfaktoren der Bachelorarbeit festgelegt. Sie setzen sich aus Resultaten und einzuhaltenden nichtfunktionalen Anforderungen (NFAs) zusammen. Diese kritischen Erfolgsfaktoren werden an dieser Stelle noch einmal aufgelistet.

- Die Benutzeranforderungen werden repräsentativ und aussagekräftig evaluiert und dargestellt.
- Anforderungen können mit Policies möglichst einfach abgebildet werden. Die Sprache bzw. die Syntax ist nicht unnötig komplex und damit für Kunden und Consultants verständlich.
- Es soll abgeschätzt werden, wie die Laufzeiteigenschaften der vorgeschlagenen Informationsbeschaffungslösung aussehen werden. Ausserdem soll ersichtlich sein, an welchen Stellen die Performance, z.B. durch Caching-Mechanismen, verbessert werden soll.

Aus diesen Erfolgsfaktoren wurden am Anfang des Projekts NFAs verfasst. Die folgenden Abschnitte gehen auf die Resultate dieser Arbeit und die Erfüllung der NFAs ein. Damit wird auch die Zielerreichung in Bezug auf die oben genannten Erfolgsfaktoren bewertet.

Das Ende dieses Kapitels gibt ausserdem einen Ausblick auf zukünftige Projekte bei der Adcubum und wie die Resultate dieser Arbeit die Produktevaluation und den «Make or Buy» Endscheid unterstützen können.

5.2. Resultate der Bachelorarbeit

Die folgenden Resultate sind während der Bachelorarbeit entstanden.

5.2.1. Funktionale Anforderungen an das Autorisierungssystem

Ein grosser Anteil dieser Bachelorarbeit wurde in die Aufnahme der funktionalen Anforderungen, konkret die Schutzbegehren der Kunden von Adcubum, investiert. Daraus ist eine Liste von 18 User Stories entstanden. Diese geben einerseits Aufschluss darüber wie der heutige Objektschutz eingesetzt wird und andererseits welche Schutzbegehren mit dem neuen Autorisierungssystem umsetzbar sein müssen.

5.2.2. Policies

Ein Ziel dieser Arbeit war die Evaluation einer geeigneten Syntax, um die auf dem Attribute-Based Access Control (ABAC)-Paradigma basierenden Policies darzustellen. Mit XACML wurde eine Syntax gefunden, welche den NFAs entspricht und mit welcher sich die verschiedenen User Stories umsetzen liessen.

5.2.3. Prototyp

Durch die prototypische Umsetzung mit der Open Source Library «Balana» [23] konnte gezeigt werden, dass sich die funktionalen Anforderungen auf der Basis eines ABAC-basierten Autorisierungssystems implementieren lassen. Der Prototyp kann Autorisierungsanfragen über die RESTful HTTP Schnittstelle aus der Studienarbeit [28] entgegennehmen, und mithilfe der verfassten Policies evaluieren.

5.2.4. Aussage zu Performance

Aufgrund der in Anhang A durchgeführten Performanceanalyse konnte ein Performancekostendach definiert werden, an welchem sich die neue Autorisierungslösung messen lässt. Dadurch kann bei der Implementation sichergestellt werden, dass sich die Performance mit dem neuen System gegenüber der alten Lösung nicht verschlechtert. Die im Kapitel 4.5 vorgeschlagenen Optimierungen helfen dabei die in der NFA «Performance» definierten Ziele zu erreichen, zeigen aber auch die Aufwände und Risiken der jeweiligen Optimierung auf.

5.2.5. Migration

Während dieser Bachelorarbeit konnte die Erkenntnis gewonnen werden, dass eine automatisierte Migration der Berechtigungskonfiguration kaum zu bewerkstelligen ist. Der Wechsel des Zugriffkontrollmodells hin zu ABAC erfordert ein mit Bedacht durchgeführtes «Attribute Engineering» [52] und die manuelle Entwicklung von XACML-Policies. Diese Artefakte automatisiert zu generieren wäre mit einem grossen Programmieraufwand verbunden und würde nur mit geringer Wahrscheinlichkeit zu einem gut wartbaren Resultat führen. Auf Basis dieser Erkenntnis wurde ein Vorgehen für die Migration vorgeschlagen.

5.3. Bewertung der definierten NFAs

In Kapitel 3.3 wurden sieben NFAs definiert, welche für die Bachelorarbeit oder auch für die zukünftige Autorisierungskomponente von Relevanz sind. Alle für die Bachelorarbeit relevanten NFAs werden aufgrund des Erreichten bewertet. Bezieht sich eine NFA zudem auf die zukünftige Autorisierungskomponente, wird anstelle einer Bewertung eine Beschreibung abgegeben wie diese NFA eingehalten werden kann.

5.3.1. Repräsentativität und Aussagekraft der fachlichen Anforderungen

Bei der Ermittlung der Anforderungen wurde direkt mit den Kunden von Adcubum gesprochen. In mehreren Interviews wurden die verschiedenen Schutzbegehren der Kunden aufgenommen und in Form von User Stories dokumentiert. Damit die Repräsentativität dieser User Stories gewährleistet ist, wurden vier verschiedene Kunden befragt, welche sowohl die Standards der Adcubum als auch individuelle Objektschutzparametrierungen verwenden. Die insgesamt 18 verfassten User Stories repräsentieren die fachlichen Anforderungen der Kunden an den Schutz in SYRIUS umfassend, wenn auch nicht abschliessend. Die Repräsentativität der Anforderungen wird als **erfüllt** betrachtet, da mehrere Kunden befragt wurden und eine grosse Anzahl der in Erfahrung gebrachten User Stories bei mehreren oder sogar allen Kunden vorkamen.

Die Aussagekraft wurde mit dem in Anhang B dokumentierten Benutzertest überprüft und **erfüllt**. Der Consultant war in der Lage, die ihm vorgelegte User Story, innerhalb der von diesem NFA vorgegeben 10 Minuten zu verstehen. Dabei waren vorallem die Akzeptanzkriterien [51] hilfreich. Trotzdem gab es einige Hinweise darauf, dass die Verwendung des Templates von Cohn [9] bei den User Stories noch verbessert werden kann. Diese Resultate und Verbesserungsmöglichkeiten wurden im Anhang B entsprechend dokumentiert.

Aufgrund der Ergebnisse des Benutzertests und der Breite an aufgenommenen Benutzeranforderungen kann diese NFA als **erfüllt** angesehen werden.

5.3.2. Einfachheit der Policy-Syntax

Bei der Auswahl der Policy-Syntax in Abschnitt 4.1.3 wurde neben anderen Faktoren auch das Kriterium der Einfachheit berücksichtigt. Die mit der evaluierten Syntax, XACML, verfassten Policies wurden ebenfalls durch den Benutzertest in Anhang B überprüft und die Syntax hat die geforderten Kriterien **erfüllt**. Die Testperson war in der Lage die ihr vorgelegte XACML-Policy innerhalb von weniger als den vorgegebenen 10 bis 15 Minuten zu verstehen. Auch eine Anpassung der Policy war in weniger als 20 Minuten möglich. Die detaillierten Resultate befinden sich in Anhang B.

Trotz der Erfüllung dieser NFA wäre eine weitere Vereinfachung der Syntax durch eine zusätzliche DSL, wie z.B. Abbreviated Language For Authorization (ALFA) [38] von Axiomatics [5] denkbar, wie dies im Kapitel 4 unter Abschnitt 4.1.3 *Auswahl der Policy-Syntax* festgehalten wurde.

5.3.3. Performance

Da in Absprache mit dem Industriepartner beschlossen wurde den Prototyp mit «Mock»-Implementationen der externen Systeme zu implementieren, konnte keine aussagekräftige Performanceanalyse der neuen ABAC-Komponente gemacht werden. Mit der in Anhang A dokumentierten Performanceanalyse konnte stattdessen ein Performancekostendach festgelegt und der Wunsch des Industriepartners nach einer messbaren Obergrenze des Performance-«Overheads» erfüllt werden. Ausserdem wurden im Abschnitt 4.5 Vorschläge unterbreitet an welchen Stellen in der vorgeschlagenen Architektur Performanceverbesserungen durchgeführt werden können. Diese können dabei helfen, die Performanceanforderungen bei der Implementation des Autorisierungssystems zu erfüllen. Diese NFA gilt somit auch als **erfüllt**, auch wenn das ursprüngliche Ziel einer Abschätzung der Laufzeiteigenschaften bezüglich der Informationsbeschaffungslösung nur **teilweise erfüllt** ist. Da keine Umsysteme angebunden wurden entspricht dieses Ergebnis trotzdem den Erwartungen des Industriepartners. Diese minimale Abweichung des Erfolgskriteriums aus der Aufgabenstellung (Anhang 3) wurde vom Industriepartner befürwortet (Besprechung vom 28.02.2017).

5.3.4. Lizenzen für Prototyp

Diese NFA wurde **erfüllt**, da die Open Source Library «Balana» [23] unter der *Apache Software Licence 2.0* [15] lizenziert ist, welche gemäss dem internen Dokument der Adcubum «Anbindung Fremdbibliotheken» [1] verwendet werden darf. Ansonsten wurden, abgesehen von den in der Studienarbeit [28] bereits verwendeten, keine Fremdbibliotheken eingesetzt.

5.3.5. Logging / Audit Trail

Sämtliche Autorisierungsanfragen im Prototypen werden über Log4J geloggt. Die Log-Einträge enthalten die Identität des Benutzers, Business Object (BO)-Id und MetaBO-Id des BOs auf welches der Benutzer zugreifen wollte, sowie die Operation und die Entscheidung des Policy Decision Point (PDP) (PERMIT/DENY). Damit wurde diese NFA **erfüllt**.

5.3.6. Zukunftssicherheit

Auf die NFA Zukunftssicherheit wurde sowohl bei der Auswahl der Policy-Syntax als auch der PDP-Library für den Prototypen geachtet. Diese NFA wird ebenfalls als **erfüllt** betrachtet, da die gewählte Policy-Syntax XACML eine gute Verbreitung in der IT-Branche hat und damit die Chance auf langfristigen Support gut ist. Ausserdem wurde bei der Implementation des Prototypen darauf geachtet, dass die Abhängigkeiten zur PDP-Implementation möglichst gering ist und nur gegen Interfaces programmiert wird.

5.3.7. Wartbarkeit

Die NFA Wartbarkeit legt fest, dass die Policies innerhalb der üblichen Wartungsfenster aktualisiert werden können. Da diese in XACML-Files abgelegt sind, können sie problemlos ausgetauscht werden. Je nach Implementation des PDP können die Files zur Laufzeit neu eingelesen werden und es ensteht keine «Downtime». In den vorgegebenen «Downtimes» für Minor Softwareupdates von bis zu zwei Stunden, lässt sich die Autorisierungslösung problemlos aktualisieren. Desweiteren fordert die NFA, dass externe Libraries, im Sinne der Wartbarkeit, sauber durch ein Interface gekapselt sein sollen, damit die Implementation einfach austauschbar bleibt. Dies wurde bei der Implementation des Prototypen berücksichtigt, weswegen auch diese NFA als **erfüllt** betrachtet wird.

5.4. Ausblick

Diese Bachelorarbeit zeigt auf, dass es technisch möglich ist die bestehende Autorisierungslösung in SYRIUS mit einer ABAC-basierten, externen Komponente abzulösen. Die erarbeiteten XACML-Policies können als Beispiel für eine mögliche Umsetzung einer Schutzanforderung in ABAC dienen. Ein besonderes Augenmerk muss dabei auf das *Attribute Engineering* gelegt werden, da daraus die Datenstruktur des Attributrepository hervorgeht. Eine Herausforderung bleibt im Bezug auf die Datenstruktur sicher eine identische Abbildung des heutigen Schutzpfades, auch im Bezug auf die Effizienz der Lösung. Die Datenstruktur der Policy Information Points (PIPs) hat nicht nur Auswirkungen auf die zu schreibenden Policies sondern auch einen wesentlichen Einfluss auf die Performance der zu implementierenden ABAC-Komponente. Wie sich in der Performanceanalyse in Anhang A herausgestellt hat, ist die jetzige Lösung sehr effizient, da sich der «Overhead» eines Requests im tiefen Millisekundenbereich befindet. Es wird die Umsetzung von einigen der vorgeschlagenen Performanceoptimierungen benötigen um die in der NFA «Performance» geforderten Kriterien zu erreichen. Einige dieser Performanceoptimierungen bringen grössere «Architectural Refactorings» [33, 65], und dadurch auch gewisse Risiken in SYRIUS mit sich. Deshalb müssen zwingend die Sicherheitsrisiken von solchen Optimierungen beachtet werden. Eine weitere Erkenntnis aus dieser Bachelorarbeit ist, dass die Migration fast nicht automatisiert werden kann. Während die Standardberechtigungsdefinitionen in SYRIUS auch zukünftig mit wiederverwendbaren Policies abgedeckt werden können, müssen die kundenspezifischen Berechtigungen gemeinsam mit dem Kunden analysiert und in einer entsprechenden Policy verfasst werden. Bei der gleichzeitigen Verwendung von unterschiedlichen Policies muss zwingend auf die Struktur und das Verhalten innerhalb eines PolicySets geachtet werden. Einen Vorschlag für eine solche Policy-Struktur ist im Kapitel 4.3 beschrieben.

Diese Bachelorarbeit legt die konzeptionellen Grundlagen für die folgenden Projekte bei Adcubum und eine zukünftige Einführung eines attributbasierten Autorisierungssystem für SYRIUS. Mit den erarbeiteten funktionalen und nichtfunktionalen Anforderungen aus Kapitel 3 können während der Evaluation von ABAC-basierten Autorisierungskomponenten alle in Betracht gezogenen Produkte überprüft und entsprechend bewertet werden. Damit ist Adcubum mit den gewonnenen Resultaten dieser Arbeit in der Lage zu überprüfen, welche ABAC-Produkte die Anforderungen der Kunden erfüllen. Auf dieser Basis ist es anschliessend möglich einen «Make or Buy»-Entscheid zu fällen. Bei der Umsetzung des Projektes helfen die gewonnenen Erkenntnisse aus dieser Bachelorarbeit um technische Risiken einzugrenzen und eine möglichst performante Architektur zu finden.

A. Performanceanalyse

A.1. Motivation

Die Performance ist ein wesentlicher Bestandteil der nichtfunktionalen Anforderungen (NFAs). Damit die Anforderungen an die Performance messbar definiert werden können, wurde eine Performanceanalyse der bestehenden Lösung durchgeführt. Die Resultate dieses Tests sind in die NFA *Performance* im Abschnitt 3.3.1 eingeflossen. Damit kann sichergestellt werden dass das Performancekostendach des neuen Systems auf realistischen Zahlen basiert. Eine Performancemessung im Prototypen dieser Arbeit wäre nicht aussagekräftig, da die externen Systeme für die Informationsbeschaffung nicht integriert wurden. Aus diesem Grund wurde die Performance des Prototypen nicht gemessen.

A.2. Vorgehen

Nachfolgend wird das Vorgehen bei der Performanceanalyse erläutert. Dies beinhaltet die Auswahl der Use Cases in SYRIUS, woraus sich die Structured Query Language (SQL)-Abfragen ergeben, sowie das Vorgehen bei der anschliessenden Messung der Ausführungszeit der einzelnen SQL-Abfragen.

Die Use Cases werden in SYRIUS durchgeführt und die SQL-Abfragen protokolliert. Anschliessend kann die Performance der einzelnen Abfragen überprüft werden.

A.2.1. Use Cases

Die ausgewählten Use Cases kommen bei den Kunden sehr häufig vor und die Performance dieser Use Cases ist in der bestehenden Lösung sehr gut. Diese Performanceanalyse verwendet bewusst zwei der Use Cases bei welchen die Performance bereits im bestehenden System gut ist, damit sich die Performanceanforderungen des neuen Autorisierungssystems an den effizienten Use Cases orientieren, und somit die gute Performance bei diesen häufig vorkommenden Use Cases beibehalten werden kann. In Absprache mit einem Experten der jetzigen Objektschutzlösung von Adcubum, wurden zwei repräsentative Use Cases ausgewählt um die Performance zu messen. Beim ersten Use Case handelt es sich um die *Suche eines Partners*, beim zweiten um das *Öffnen der Partnerübersicht*.

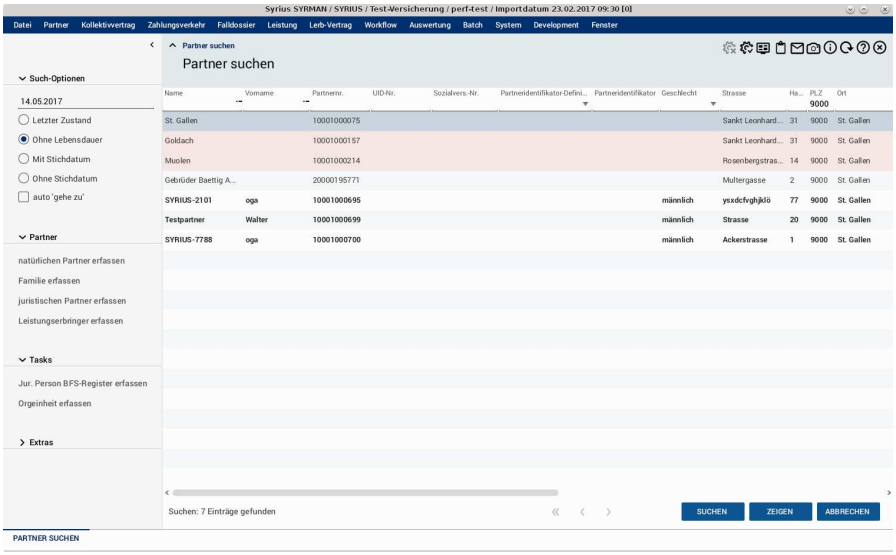

Abbildung A.1.: Suche eines Partners in SYRIUS

Diese beiden Use Cases werden in der Praxis häufig verwendet und es werden, hauptsächlich beim Öffnen der Partnerübersicht, viele Business Objects (BOs) aus der Datenbank geladen.

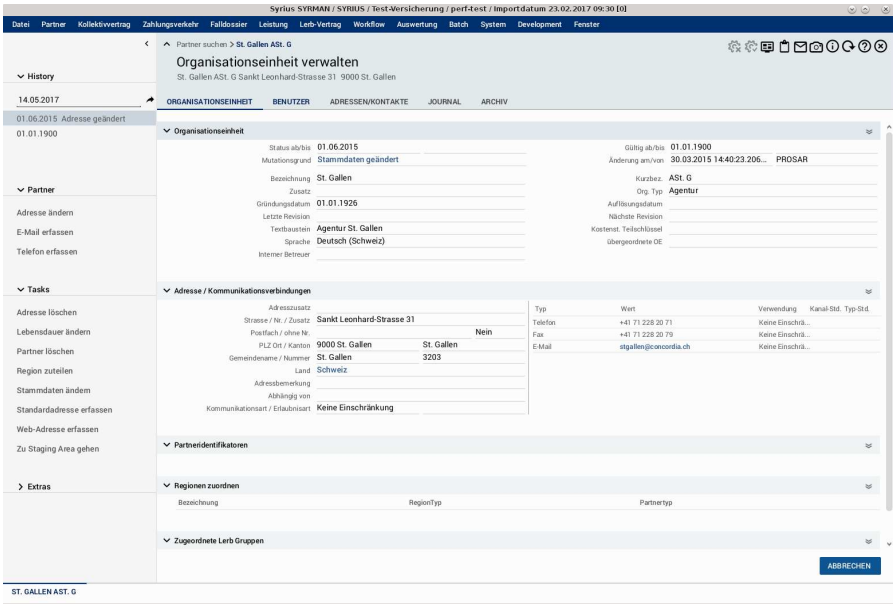

Abbildung A.2.: Anzeige eines Partners in SYRIUS (Partnerübersicht)

Ob der Test repräsentativ ist, hängt im wesentlichen von der Parametrierung des Ob-

jektschutzes und der Menge an Laufdaten in der Datenbank ab. Die Performance wird hauptsächlich von der Länge der Schutzpfade beeinflusst. Um ein möglichst realistisches Ergebnis zu erhalten, wurde die Parametrierung eines realen Kunden der Adcubum verwendet. Dieser benutzt einen Partnerschutz, welcher bei vielen Kunden im Einsatz ist und somit ein repräsentatives Resultat ermöglichen soll. Des Weiteren enthält die Datenbank einen vollen Datenbestand und somit genügend Daten für eine realistische Performanceanalyse.

Der Objektschutz basiert auf der Erweiterung der SQL-WHERE-Klausel. Damit die Performance einer einzelnen *Autorisierung* gemessen werden kann, müssen die einzelnen Datenbankabfragen auf die BOs separat betrachtet werden.

```
1  SELECT vertragId, itsPartner, ... FROM Vertrag vertrag
2          WHERE vertrag.vertragId = ...
3          AND EXISTS (
4                  SELECT partner.itsPartnerSchutz FROM Partner partner
5                          WHERE partner.BOId = vertrag.itsPartner
6                          -- Erlaubte Partnerschutzobjekte für den
7                          -- aktuellen Benutzer
8                          AND partner.itsPartnerSchutz IN
9                                  ('4','3','2','1')
10         );
```

Auflistung A.1: Objektschutz: Erweiterung der WHERE-Klausel [28]

Mit folgendem Vorgehen wurden die einzelnen Abfragen der oben erwähnten Use Cases analysiert:

- Durchführen der Use Cases aus dem SYRIUS Ultra-Thin Client (UTC).
- Gleichzeitiges aufzeichnen aller SQL-Abfragen (sql.log-Datei ist in SYRIUS Entwicklungsumgebung vorhanden).
- Ausführen und Messen der Zeit für die einzelnen Queries mit SQL-Tool.
- Ausschalten des Objektschutzes und anschliessend erneutes Aufzeichnen der einzelnen Abfragen.
- Erneutes Ausführen und Messen der Abfrage ohne WHERE-Klausel.
- Anschliessend können die Abfragezeiten der einzelnen Queries verglichen werden.

A.2.2. Messung der Ausführungszeit einer Abfrage

Die Messung der Performance einzelner SQL-Abfragen, um diese anschliessend zu vergleichen, ist nicht trivial, da die Datenbank Optimierungen vornimmt und Resultate einzelner Abfragen *cached*. Um gängige Fehler bei der Messung der Abfrageperformanz zu vermeiden, wurden vorerst einige Recherchen [25, 32, 29] durchgeführt und das Vorgehen mit einem Datenbankspezialisten der Adcubum besprochen.

Das grösste Problem bei der Messung der Ausführungszeit ist, dass die Datenbank bei mehrfacher Ausführung einer Abfrage Daten «cached» und die Ausführung dadurch schneller wird. Diesbezüglich wurde vom Datenbankspezialisten empfohlen, die Abfrage mehrfach durchzuführen, die erste Ausführung nicht zu bewerten und für die restlichen Ausführungen den Durchschnitt zu berechnen. Dieses Vorgehen wird auch in diesem Blogpost «Comparing Oracle Query Performance for Faster Applications» [25] vorgeschlagen. Dieses Verhalten, dass vor allem die erste Ausführung einer Abfrage wesentlich langsamer ist als die nachfolgenden Abfragen, konnte nach ersten Tests schnell nachvollzogen werden.

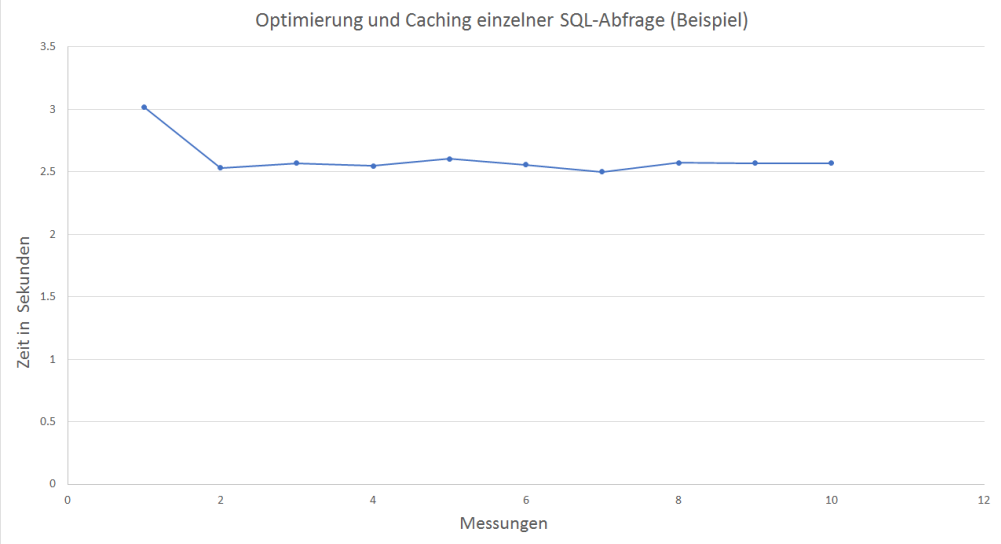

Abbildung A.3.: Ausführungszeit einer Beispiel-Abfrage nach 10 Ausführungen

Abbildung A.3 zeigt dies anhand einer Beispiel-Abfrage, welche 10 mal ausgeführt wurde. Bereits ab der zweiten Ausführung verändert sich die Ausführungszeit nur noch um 1 bis maximal 2 Prozent.

Da die Ausführungszeit der ersten Abfrage nicht repräsentativ ist, wurde folgendes Vorgehen für die Messung der Abfragezeiten festgelegt:

- Abfragen werden 6 mal ausgeführt, wobei die erste Messung ignoriert wird.

- Von den weiteren 5 Messungen wird der Durchschnitt berechnet und dieses Resultat für den Vergleich herangezogen.

Eine Alternative Variante welche ebenfalls häufig verwendet wird um die Performance verschiedener Datenbanken zu vergleichen, wie zum Beispiel in [32], ist das Testen ohne dieses Datenbank-«Warmup». Dabei wird direkt die Zeit der allerersten Ausführung der Abfrage genommen. Bei diesem Vorgehen muss allerdings nach der Ausführung jeder einzelnen Abfrage der Cache gelöscht und idealerweise der gesamte Datenbankserver neu gestartet werden. Diese Variante kam bei der Performanceanalyse dieser Arbeit nicht in Frage, da dafür nicht genügend Kontrolle über den Datenbankserver zur Verfügung stand.

Eine weitere Idee war der Vergleich der Kosten («Costs») einer Datenbankabfrage, anstatt der Ausführungszeit. Dies ist offensichtlich aber ein weit verbreiteter Fehler, da die Kosten unterschiedlicher Abfragen nicht miteinander verglichen werden können. *Tom Kyte*, Datenbankspezialist bei Oracle [42], erklärt dies sehr gut und umfangreich in einem Blogeintrag [29]. Aus diesem Grund wurde diese Idee schnell wieder verworfen und auf die oben erklärte Messung der Ausführungszeit zurückgegriffen.

A.3. Voraussetzungen und Vorbereitungen

Nachfolgende Voraussetzungen müssen für die Performanceanalyse sichergestellt werden:

- Datenbank mit guter Parametrierung und grosser Datenmenge bereitstellen.
- Datenbankserver darf zum Zeitpunkt der Analyse nicht durch andere Quereinflüsse belastet werden. Aus diesem Grund wurde die Analyse zu einer Randzeit durchgeführt, an welcher das System von keinen anderen Benutzern verwendet wurde.
- Entwicklungsumgebung mit SYRIUS-Codestand welcher zu der Datenbank passt. Wird SYRIUS direkt aus der Entwicklungsumgebung gestartet, besteht die Möglichkeit den Objektschutz auszuschalten.
- Oracle SQL Developer [44] mit Zugriff auf die Datenbank, um die Queries manuell abzusetzen und die Ausführungszeit zu messen.

A.3.1. Datenbank

Die gewählte Datenbank enthält eine Parametrierung welche von einem Kunden von Adcubum übernommen wurde. Ausserdem enthält sie eine grosse Menge an Daten (Volldatenbestand). Die Parametrierung darf nicht verändert werden und es muss sichergestellt werden, dass der Datenbankserver nicht durch andere Aufgaben belastet wird.

A.3.2. SYRIUS Entwicklungsumgebung

Um die Tests durchzuführen wurde eine Entwicklungsumgebung aufgesetzt in welcher der SYRIUS Codestand auf den Stand der entsprechenden Datenbank gebracht wurde. Wird SYRIUS aus Eclipse im *Developer*-Modus gestartet, kann zur Analyse der Abfragen die Datei *sql.log* ausgelesen werden. In dieser Log-Datei werden alle SQL-Abfragen welche auf der Datenbank ausgeführt werden geloggt.

Die *Objektschutzfacade* [28] in SYRIUS bietet die Möglichkeit, den Objektschutz ein- und auszuschalten. Über diesen Mechanismus können die Abfragen sowohl mit als auch ohne Objektschutz-WHERE-Klausel generiert werden.

A.4. Protokoll und Resultate der Testdurchführung

In diesem Abschnitt wird erläutert wie die Performanceanalyse durchgeführt wurde. Im Detail wird dies nur für den ersten Use Case beschrieben, da das Vorgehen anschliessend für jede Abfrage dasselbe ist.

A.4.1. Use Case «Partner suchen»

Für diesen Anwendungsfall wurde eine Partnersuche mit grosser Resultatanzahl durchgeführt. Konkret wurde nach allen Partnern mit dem Namen «Müller» gesucht.

In einem ersten Durchgang wird diese Suche in SYRIUS abgesetzt während der Objektschutz aktiv ist. Die Abfrage welche dabei abgesetzt wird, siehe Auflistung A.2, kann dem *sql.log* entnommen werden.

```
SELECT
  /*f57Fk+Y9o4dG80jE35Z3RA==c7fcabbd69bc2766203a3c17884bab09a05ff6f5*/
  b3_1.PKey, b3_1.BOId, b3_1.MetaBO, b3_1.Name, b3_1.Vorname, b3_1.PartnerNr, b3_1.UIDNr,
  b3_1.SozialVersNr, b3_1.Geschlecht, b3_1.PrivatStrasse, b3_1.PrivatHausnummer, b3_1.PrivatPLZ,
  b3_1.PrivatOrt, b3_1.Geburtstag, b3_1.PartnerTyp, b3_1.ExtPartnerId, b3_1.itsIntBetreuer,
  b3_1.GueltAb, b3_1.GueltBis, b3_1.itsPartnerSchutz
FROM Partner b3_1
WHERE b3_1.NL_Name LIKE 'MULLER%'
AND NOT EXISTS
  (SELECT
    /*rPVFX8M1zC4nDyZ201Gkjw==*/
    b1164_2.itsPartner
  FROM PartnerRole b1164_2,
    RoleDef b1160_3
  WHERE b1164_2.itsRoleDef      = b1160_3.BOId
  AND b1160_3.PartnerSearchable = '-10111'
  AND b3_1.BOId                 = b1164_2.itsPartner
  AND b1160_3.Replaced          = TO_TIMESTAMP('3000-1-1 0:0:0.0','YYYY-MM-DD HH24:MI:SS.FF9')
  AND b1160_3.stateEnd          = TO_DATE('3000-1-1','YYYY-MM-DD')
  AND b1164_2.Replaced          = TO_TIMESTAMP('3000-1-1 0:0:0.0','YYYY-MM-DD HH24:MI:SS.FF9')
  AND b1164_2.stateEnd          = TO_DATE('3000-1-1','YYYY-MM-DD')
  )
AND b3_1.Replaced = TO_TIMESTAMP('3000-1-1 0:0:0.0','YYYY-MM-DD HH24:MI:SS.FF9')
AND b3_1.stateEnd = TO_DATE('3000-1-1','YYYY-MM-DD')
-- Objektschutz
AND ((EXISTS
  (SELECT b3_1.itsPartnerSchutz
  FROM DUAL
  WHERE b3_1.itsPartnerSchutz NOT IN ('-2')
  )
OR NOT EXISTS
  (SELECT b3_1.itsPartnerSchutz
  FROM DUAL
  WHERE b3_1.itsPartnerSchutz IS NOT NULL
  )));
```

Auflistung A.2: Partnersuche: Abfrage mit Objektschutz

Anschliessend wird der Objektschutz in der *Objektschutzfacade* über das Flag *ObjektschutzDisabled* deaktiviert. Dafür muss SYRIUS neu kompiliert und gestartet werden. Nach einer wiederholten Durchführung der Partnersuche in SYRIUS, wird das SQL erneut der Datei *sql.log* entnommen.

```
SELECT
  /*f57Fk+Y9o4dG80jE35Z3RA==*/
  b3_1.PKey, b3_1.BOId, b3_1.MetaBO, b3_1.Name, b3_1.Vorname, b3_1.PartnerNr, b3_1.UIDNr,
  b3_1.SozialVersNr, b3_1.Geschlecht, b3_1.PrivatStrasse, b3_1.PrivatHausnummer, b3_1.PrivatPLZ,
  b3_1.PrivatOrt, b3_1.Geburtstag, b3_1.PartnerTyp, b3_1.ExtPartnerId, b3_1.itsIntBetreuer,
  b3_1.GueltAb, b3_1.GueltBis, b3_1.itsPartnerSchutz
FROM Partner b3_1
WHERE b3_1.NL_Name LIKE 'MULLER%'
AND NOT EXISTS
  (SELECT
    /*rPVFX8M1zC4nDyZ201Gkjw==*/
    b1164_2.itsPartner
  FROM PartnerRole b1164_2,
    RoleDef b1160_3
  WHERE b1164_2.itsRoleDef      = b1160_3.BOId
  AND b1160_3.PartnerSearchable = '-10111'
  AND b3_1.BOId                 = b1164_2.itsPartner
  AND b1160_3.Replaced          = TO_TIMESTAMP('3000-1-1 0:0:0.0','YYYY-MM-DD HH24:MI:SS.FF9')
  AND b1160_3.stateEnd          = TO_DATE('3000-1-1','YYYY-MM-DD')
  AND b1164_2.Replaced          = TO_TIMESTAMP('3000-1-1 0:0:0.0','YYYY-MM-DD HH24:MI:SS.FF9')
  AND b1164_2.stateEnd          = TO_DATE('3000-1-1','YYYY-MM-DD')
  )
AND b3_1.Replaced = TO_TIMESTAMP('3000-1-1 0:0:0.0','YYYY-MM-DD HH24:MI:SS.FF9')
AND b3_1.stateEnd = TO_DATE('3000-1-1','YYYY-MM-DD');
```

Auflistung A.3: Partnersuche: Abfrage ohne Objektschutz

Aus Auflistung A.3 wird ersichtlich, dass der Objektschutz in dieser Abfrage nicht mehr vorhanden ist.

Anschliessend wurde bei beiden Abfragen, wie in Abschnitt A.2.2 *Messung der Ausführungszeit einer Abfrage* erläutert, die Ausführungszeit gemessen.

Tabelle A.1.: Resultate Partner-Suche

Durchführung	**«Mit» Objektschutz**	**«Ohne» Objektschutz**	**Differenz**	**Zeitverlust durch Objektschutz in %**
#1:	2.809s	2.558s		
#2:	2.506s	2.542s		
#3:	2.518s	2.501s		
#4:	2.48s	2.523s		
#5:	2.511s	2.456s		
Durchschnitt:	2.5648s	2.516s	**48.8ms**	**1.94%**

Da die Oracle Datenbank grundsätzlich nur die ersten 50 Zeilen des *ResultSet* lädt, wurde bei diesem Use Case noch ein «ORDER BY pkey» bei beiden Abfragen hinzugefügt.

Dadurch wird der Datenbankserver gezwungen das ganze *ResultSet* zu laden, anstatt nur die ersten 50 Zeilen der Tabelle. Die Abfrage mit Objektschutz liefert 369'873 Resultate und diejenige ohne Objektschutz 369'916.

Aus der Tabelle A.1 geht hervor, dass die zusätzliche Zeit für den Objektschutz in diesem Fall **48.8 Millisekunden** benötigt. Dies entspricht einem prozentualen Performanceverlust von **1.9%** durch den Objektschutz.

A.4.2. Use Case «Partner öffnen»

Auch der Use Case «Partner öffnen» wurde mit und ohne Objektschutz durchgeführt und alle Abfragen aufgezeichnet. Da es bei diesem Use Case eine grosse Anzahl von Abfragen gibt, wird an dieser Stelle auf die Darstellung aller SQLs verzichtet. Beim Öffnen des Partners werden über 100 Abfragen ausgeführt. Aus diesen Abfragen wurden alle herausgefiltert, welche gar keinen Objektschutz beinhalten und somit identisch wären. Damit ist die Anzahl der Abfragen mit Objektschutz bei 34.

Die Messung der Ausführungszeit wurde für alle 34 Abfragen durchgeführt.

Partner öffnen		Zeit 1 (mit)	Zeit 2 (mit)	Zeit 3 (mit)	Zeit 4 (mit)	Zeit 5 (mit)	Zeit 1 (ohne)	Zeit 2 (ohne)	Zeit 3 (ohne)	Zeit 4 (ohne)	Zeit 5 (ohne)
SELECT /*slfTvl8AB	SELECT /*slfTvl8A	0.004	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.004	0.002
SELECT /*1dd16fb5	SELECT /*dec1ee	0.002	0.001	0.002	0.002	0.003	0.002	0.002	0.002	0.001	0.002
SELECT /*YKAKDatV	SELECT /*YKAKD	0.003	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002
SELECT /*iCZjFkutF	SELECT /*iCZjFku	0.01	0.013	0.01	0.01	0.009	0.007	0.007	0.008	0.008	0.008
SELECT /*m4f5FqNp	SELECT /*m4f5Fq	0.002	0.001	0.002	0.001	0.001	0.002	0.002	0.002	0.002	0.002
SELECT /*/Kl8PKmG	SELECT /*/Kl8PKr	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002
SELECT /*UKJjeIO/a	SELECT /*UKJjeIO	0.003	0.002	0.002	0.002	0.002	0.003	0.001	0.002	0.002	0.002
SELECT /*NSeVHCt	SELECT /*NSeVH	0.003	0.003	0.002	0.002	0.003	0.003	0.002	0.003	0.003	0.002
SELECT /*84xo7bXV	SELECT /*84xo7b	0.003	0.003	0.003	0.003	0.003	0.002	0.003	0.002	0.003	0.003
SELECT /*55fb2fa22	SELECT /*036b61	0.003	0.003	0.003	0.002	0.003	0.003	0.003	0.003	0.003	0.002
SELECT /*+ FIRST_R	SELECT /*+ FIRST	0.016	0.003	0.003	0.003	0.002	0.005	0.002	0.003	0.002	0.002
SELECT /*/8AbapEL	SELECT /*/8Abapl	0.003	0.003	0.002	0.003	0.002	0.002	0.001	0.002	0.002	0.002
SELECT /*e7d55a3e	SELECT /*e38c0d	0.02	0.004	0.004	0.004	0.004	0.009	0.004	0.005	0.003	0.003
SELECT /*94pvG/5Y	SELECT /*94pvG/	0.002	0.001	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002
SELECT /*HAGx8bA	SELECT /*HAGx8l	0.002	0.002	0.002	0.002	0.001	0.001	0.002	0.002	0.001	0.002
SELECT /*6873abcc	SELECT /*50df890	0.003	0.002	0.003	0.003	0.002	0.002	0.002	0.002	0.002	0.002
SELECT /*DzzAu29p	SELECT /*DzzAu2	0.02	0.023	0.003	0.003	0.003	0.007	0.009	0.003	0.003	0.003
SELECT /*cd8bbe5c	SELECT /*731516	0.003	0.003	0.003	0.004	0.003	0.004	0.003	0.003	0.003	0.003
SELECT /*swGychT7	SELECT /*swGych	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.003	0.003	0.002
SELECT /*cY91h9SO	SELECT /*cY91h9	0.002	0.002	0.003	0.002	0.002	0.002	0.002	0.002	0.002	0.002
SELECT /*/ceIetYZC	SELECT /*/ceIetY	0.002	0.003	0.002	0.002	0.002	0.002	0.001	0.002	0.002	0.001
SELECT /*qIzmzXQA	SELECT /*qIzmzX	0.003	0.002	0.001	0.002	0.002	0.002	0.002	0.002	0.001	0.002
SELECT /*w8gafis+n	SELECT /*w8gafis	0.004	0.003	0.003	0.004	0.003	0.003	0.003	0.003	0.003	0.004
SELECT /*9xUFolwv	SELECT /*9xUFol	0.003	0.004	0.003	0.003	0.003	0.003	0.003	0.004	0.003	0.003
SELECT /*357F+8q2	SELECT /*357F+8	0.027	0.006	0.006	0.007	0.005	0.012	0.007	0.007	0.006	0.006
SELECT /*+ FIRST_R	SELECT /*+ FIRST	0.017	0.007	0.002	0.001	0.002	0.004	0.005	0.004	0.002	0.002
SELECT /*p0MMgSu	SELECT /*p0MMg	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002
SELECT /*+ FIRST_R	SELECT /*+ FIRST	0.002	0.002	0.003	0.002	0.002	0.002	0.001	0.002	0.002	0.002
SELECT /*cd8bbe5c	SELECT /*731516	0.004	0.004	0.003	0.004	0.003	0.002	0.003	0.003	0.003	0.004
SELECT /*32151dba	SELECT /*c42722	0.003	0.003	0.003	0.003	0.004	0.003	0.003	0.003	0.003	0.003
SELECT /*DTG0HRT	SELECT /*DTG0H	0.002	0.002	0.003	0.003	0.002	0.002	0.002	0.002	0.002	0.002
SELECT /*up8nHBW	SELECT /*up8nHB	0.003	0.002	0.002	0.002	0.002	0.002	0.002	0.003	0.002	0.002
SELECT /*DTG0HRT	SELECT /*DTG0H	0.003	0.002	0.002	0.003	0.002	0.003	0.002	0.002	0.003	0.002
SELECT /*up8nHBW	SELECT /*up8nHB	0.002	0.002	0.002	0.002	0.002	0.001	0.003	0.001	0.002	0.002

Abbildung A.4.: Ausführungszeiten der Abfragen beim Use Case «Partner öffnen»

Aus den gemessenen Resultaten, welche in Abbildung A.4 dargestellt werden, ergeben sich folgende Durchschnittswerte.

Tabelle A.2.: Resultate Partner Öffnen

Abfrage	«Mit» Objektschutz	«Ohne» Objektschutz	Differenz	Zeitverlust durch Objektschutz in %	Anzahl Resultate
SQL#1:	0.0032s	0.003s	0.2ms	6.67%	1
SQL#2:	0.002s	0.0018s	0.2ms	11.11%	0
SQL#3:	0.0022s	0.002s	0.2ms	10%	0
SQL#4:	0.0104s	0.0076s	2.8ms	36.84%	7
SQL#5:	0.0014s	0.002s	-0.6ms	-30%	0
SQL#6:	0.002s	0.002s	0ms	0%	1
SQL#7:	0.0022s	0.002s	0.2ms	10%	0
SQL#8:	0.0026s	0.0026s	0ms	0%	2
SQL#9:	0.003s	0.0026s	0.4ms	15.38%	1
SQL#10:	0.0028s	0.0028s	0ms	0%	2
SQL#11:	0.0054s	0.0028s	2.6ms	92.86%	1
SQL#12:	0.0026s	0.0018s	0.8ms	44.44%	1
SQL#13:	0.0072s	0.0048s	2.4ms	50%	4
SQL#14:	0.0018s	0.002s	-0.2ms	-10%	0
SQL#15:	0.0018s	0.0016s	0.2ms	12.5%	0
SQL#16:	0.0026s	0.002s	0.6ms	30%	0
SQL#17:	0.0104s	0.005s	5.4ms	108%	3
SQL#18:	0.0032s	0.0032s	0ms	0%	6
SQL#19:	0.002s	0.0024s	-0.4ms	-16.67%	1
SQL#20:	0.0022s	0.002s	0.2ms	10%	1
SQL#21:	0.0022s	0.0016s	0.6ms	37.5%	0
SQL#22:	0.002s	0.0018s	0.2ms	11.11%	0
SQL#23:	0.0034s	0.0032s	0.2ms	6.25%	1
SQL#24:	0.0032s	0.0032s	0ms	0%	3
SQL#25:	0.0102s	0.0076s	2.6ms	34.21%	16
SQL#26:	0.0058s	0.0034s	2.4ms	70.59%	0
SQL#27:	0.002s	0.002s	0ms	0%	0
SQL#28:	0.0022s	0.0018s	0.4ms	22.22%	0
SQL#29:	0.0036s	0.003s	0.6ms	20%	4
SQL#30:	0.0032s	0.003s	0.2ms	6.67%	1

Tabelle A.2.: Resultate Partner Öffnen

Abfrage	**«Mit» Objektschutz**	**«Ohne» Objektschutz**	**Differenz**	**Zeitverlust durch Objektschutz in %**	**Anzahl Resultate**
SQL#31:	0.0024s	0.002s	0.4ms	20%	1
SQL#32:	0.0022s	0.0022s	0ms	0%	0
SQL#33:	0.0024s	0.0024s	0ms	0%	1
SQL#34:	0.002s	0.0018s	0.2ms	11.11%	0
Total:	0.1178s	0.095s	**22.8ms**	**24%**	
				Ø 18.26%	

Diese Resultate zeigen, dass zwar in der Summe eine Differenz von **22.8 Millisekunden** zustande kommt, die Performance einer einzelnen Abfrage durch den Objektschutz aber kaum langsamer ist. In der Regel handelt es sich hier um den Bruchteil einer Millisekunde. In der Summe liegt der Zeitverlust, über den gesamten «Service-Call» gesehen, bei **24%**. Der durchschnittliche prozentuale Zeitverlust aller einzelnen Abfragen ist mit **18.26%** ebenfalls wesentlich grösser als bei Abfragen mit grossen *ResultSets*, wie beim ersten Use Case.

A.5. Bewertung der Resultate

Aufgrund der im letzten Abschnitt gezeigten Resultate wird ersichtlich, dass der Overhead durch den Objektschutz bei den effizienteren Abfragen mit wenig Resultaten, prozentual mehr ins Gewicht fällt. Auch wenn die absoluten Zahlen im zweiten Use Case sehr klein sind, liegt der Overhead durch den Objektschutz durchschnittlich bei 18% der Ausführungszeit. Bei grösseren Datenmengen, wie beim ersten Use Case, ist die Datenbank sehr effizient. Der Overhead liegt dort nur bei knappen 2%.

A.5.1. Schlussfolgerungen

Für die zukünftige Autorisierungslösung ist es wichtig zu berücksichtigen, dass jedes einzelne BO eines *ResultSets* autorisiert werden muss. Eine Bündelung dieser Autorisierungsrequests pro Abfrage ist bereits in der Autorisierungsschnittstelle [28] vorgesehen und im Prototyp implementiert worden, sodass pro Abfrage nur ein Autorisierungsrequest an das Autorisierungssystem gesendet werden muss. Aufgrund des Resultats im zweiten Use Cases dieser Performanceanalyse muss aber darauf geschlossen werden, dass innerhalb von SYRIUS eine weitere Bündelung der einzelnen Datenbankabfragen vorgenommen werden muss, um eine vergleichbare Performance wie in der jetzigen Lösung erreichen zu können. Die sehr kleinen absoluten Zeitverluste zeigen auf, dass mit separaten Autorisierungsrequests für alle 34 Abfragen dieses Use Cases, die Performance mit der neuen Architektur nicht erreicht werden kann. Aus diesem Grund wird es nicht ausreichend sein, die Autorisierung innerhalb von SYRIUS in der Persistenzschicht zu behandeln. Um die Datenbankabfragen besser bündeln zu können, werden Anpassungen in höheren Schichten der SYRIUS-Architektur notwendig sein. Auf diese Performanceoptimierung wurde im Kapitel 4, Abschnitt 4.5 bereits eingegangen.

Eine weitere Herausforderung dürfte die Autorisierung grosser *ResultSets*, wie im ersten Use Case dieser Performanceanalyse, werden. Trotz dieser grossen Datenmenge, liegt der Zeitverlust lediglich bei knapp 2%. Allerdings muss die Datenbank in der jetzigen Lösung, trotz des *Paging* in SYRIUS (maximal 60 BOs pro Page), aufgrund der WHERE-Klauseln alle Records der abgefragten Tabelle berücksichtigen, was zu dem absoluten Wert von satten 48 Millisekunden führt. Eine mögliche Lösung um diesen Wert mit der neuen Architektur zu erreichen, könnte die Implementation eines Paging-Mechanismus bei der Autorisierung sein, da nicht alle 600'000 BOs sofort in SYRIUS angezeigt werden müssen. Da die Resultate einer Datenbankabfrage mit der neuen Lösung im Nachhinein gefiltert werden müssen, kann der Paging-Mechanismus der Datenbank bei Suchabfragen sowieso nicht mehr benutzt werden.

A.5.2. Performancekostendach

Aufgrund der bereits erwähnten Resultate und Schlussfolgerungen dieser Performanceanalyse, kann der maximal erlaubte Zeitverlust für die Autorisierung nicht pro BO festgelegt werden. Aus den obigen Resultaten kann aber ein Kostendach pro Use Case, oder technisch formuliert «pro Service-Call», festgelegt werden. Aufgrund der Resultate dieser Performanceanalyse wird der maximale Overhead auf **24 Prozent** pro «Service-Call» festgelegt. Es muss dabei erwähnt werden, dass das Verhältnis von Use Case zu Service-Call in diesem Fall «1-zu-1» ist. Für das Suchen oder Öffnen eines Partners wird jeweils genau ein Service aufgerufen.

B. Benutzertest zur Validierung von NFAs

B.1. Einleitung

Die beiden nichtfunktionalen Anforderungen (NFAs) «Repräsentativität und Aussagekraft der fachlichen Anforderungen» und «Einfachheit der Policy-Syntax» (siehe Abschnitt 3.3) enthalten Kriterien, welche von den Benutzern des Systems abhängig sind. Deshalb werden die erwähnten NFAs mit einem Benutzertest validiert. Die Massnahmen welche aufgrund der Testresultate getroffen werden, sind im Abschnitt B.6 aufgelistet.

Folgendes Kriterium wurde in der NFA «Repräsentativität und Aussagekraft der fachlichen Anforderungen» festgelegt:

> **Testkriterium 1 (TK1):** «Jede User Story muss innert 10 Minuten für einen Consultant so verständlich sein, dass er diese Anforderung in der bestehenden Lösung parametrieren kann.»

Des weiteren sollen folgende Anforderungen an die «Einfachheit der Policy-Syntax» erfüllt werden:

> **Testkriterium 2 (TK2):** «Schon heute wird der Objektschutz nicht von Softwareentwicklern konfiguriert, sondern durch Consultants oder Mitarbeiter des Versicherers. Die Policy-Syntax welche in dieser Bachelorarbeit verwendet wird, soll so einfach gehalten werden, dass eine solche Person die Policy innert 10 bis 15 Minuten verstehen kann.»

> **Testkriterium 3 (TK3):** «Des weiteren sollen diese Personen auch in der Lage sein, eigene Policies innerhalb von 20 Minuten zu verfassen. Dazu wird Kenntnis über die XACML-Elemente vorausgesetzt.»

Die Erfüllung dieser Kriterien wurde durch die nachfolgenden Aufgaben, in einem Benutzertest mit einem Consultant der Adcubum, verifiziert. Die Testperson verfügt über fünf Jahre Erfahrung im Bereich der jetztigen Autorisierungslösung und der Parametrierung von SYRIUS. Mit Softwareentwicklung und Anforderungen kennt die Testperson sich ebenfalls aus. Attribute-Based Access Control (ABAC) und XACML sind neue Begriffe und werden deshalb vor dem Benutzertest kurz erläutert.

Der Benutzertest ist nicht repräsentativ, da er nur mit einer Testperson durchgeführt wurde. Ziel des Testes ist es, aufzuzeigen dass es, innerhalb der geschätzten Zeit aus den NFAs, möglich ist die geforderten Aufgaben zu erledigen.

B.2. Ausgangslage

Um ein aussagekräftiges Resultat zu erhalten muss die Testperson über ein Grundwissen zu XACML und SYRIUS verfügen. Diese Anforderung ist auch in der jeweiligen NFA festgehalten. Die Testperson erhält einen kurzen Überblick über das Ziel des Projekts, ein externes Autorisierungssystem zu implementieren und die Autorisierung aus SYRIUS zu lösen. Zusätzlich wird anhand eines Unified Modelling Language (UML) Klassendiagramms die generelle Syntax von XACML aufgezeigt. Dabei wird grob erklärt welchen Zweck die Elemente wie *Policies, Targets* oder *Rules* haben.

B.3. Testaufbau und Rahmen

Der Benutzertest wurde am 02.05.2017 in den Räumlichkeiten der Adcubum in St. Gallen durchgeführt. Alle Tests wurden innerhalb eines Nachmittags (14:30 - 18:00) durchgeführt. Nach einer zehnminütigen Einführung in XACML und die Testziele erhielt die Testperson Aufgabe 1 vorgelegt. Nach Aufgabe 1 (15:00) musste der Test für zwei Stunden unterbrochen werden, und wurde um 17:00 mit den Aufgaben 2 und 3 fortgesetzt. Alle Aufgaben wurden der Testperson mitsamt der Aufgabenstellung auf Papier vorgelegt. Aufgabe 1 musste die Testperson an einem Flipchart lösen. Neben den ausgedruckten Aufgaben und Informationen wurden keine weiteren Hilfsmittel wie das Internet oder Dokumentationen verwendet.

B.4. Aufgaben

B.4.1. Aufgabe 1: User Story verstehen

Ziel der Aufgabe

Mit dieser Aufgabe soll überprüft werden, ob der Consultant eine User Story innerhalb von 10 Minuten verstehen kann (TK1), und in der Lage wäre die Anforderung in der jetzigen Lösung zu parametrieren. Die Auswahl der User Story ist auf «US6: Alten und neuen Konzernspezialisten bei einem Konzernspezialisten-Wechsel zeitlich überschneidend berechtigen» (siehe Abschnitt 3.2.9) gefallen, da dies eine der wenigen User Stories ist, welche der Consultant noch nicht kennen sollte. Die Repräsentativität dieses Tests

ist entsprechend grösser, wenn die Testperson die Fachlichkeit der User Story noch nicht kennt.

Nachfolgend wird die referenzierte User Story aus Abschnitt 3.2.9 noch einmal dargestellt.

> **US6: Alten und neuen Konzernspezialisten bei einem Konzernspezialisten-Wechsel zeitlich überschneidend berechtigen**
>
> «Als Versicherer möchte ich dass bei einem Wechsel des Konzernspezialisten, der alte Konzernspezialist noch bis Ende Jahr und der neue Konzernspezialist bereits ab der Erfassung des Wechsels Zugriff auf die Konzerndaten hat, damit der neue Konzernspezialist bereits Einblick in die Struktur des Konzerns erhält und der alte Konzernspezialist bis Ende des Jahres seine Funktion ausüben kann.»
>
> **Akzeptanzkriterien**
>
> - Der neue Konzernspezialist erhält den Zugriff auf den Konzern und dessen Mitglieder ab dem Zeitpunkt an welchem der Wechsel in SYRIUS erfasst wurde. Dafür müssen in SYRIUS bei der Bereitstellung der Berechtigungsattribute die zukünftigen States (fachliche Historisierung) berücksichtigt werden.
> - Der alte Konzernspezialist behält den Zugriff ebenfalls, bis der Wechsel fachlich gültig wird. Dies ist üblicherweise bis Ende des Jahres.

Aufgabenstellung

Dem Consultant wird die obige User Story vorgelegt. Nachdem er sie durchgelesen hat, soll sein Verständnis für die fachliche Problematik überprüft werden. Anhand einer Grafik soll aufgezeigt werden, welche Benutzer zu welchem Zeitpunkt Zugriff auf die Konzerndaten hat.

Folgende Abbildung B.1, welche der originalen User Story hinzugefügt ist, wurde für diesen Test entfernt bzw. der Testperson nicht gezeigt:

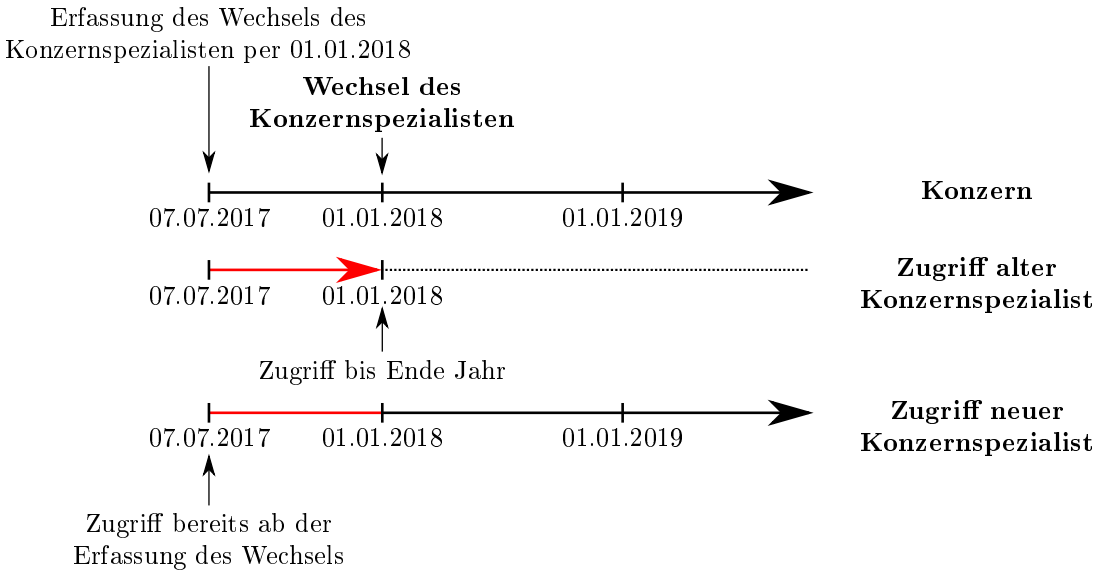

Abbildung B.1.: Zeitlich überschneidende Berechtigung bei Konzernspezialisten-Wechsel

Stattdessen wird ein leeres Gerüst vorgegeben, mit welchem die Testperson den Sachverhalt erklären soll:

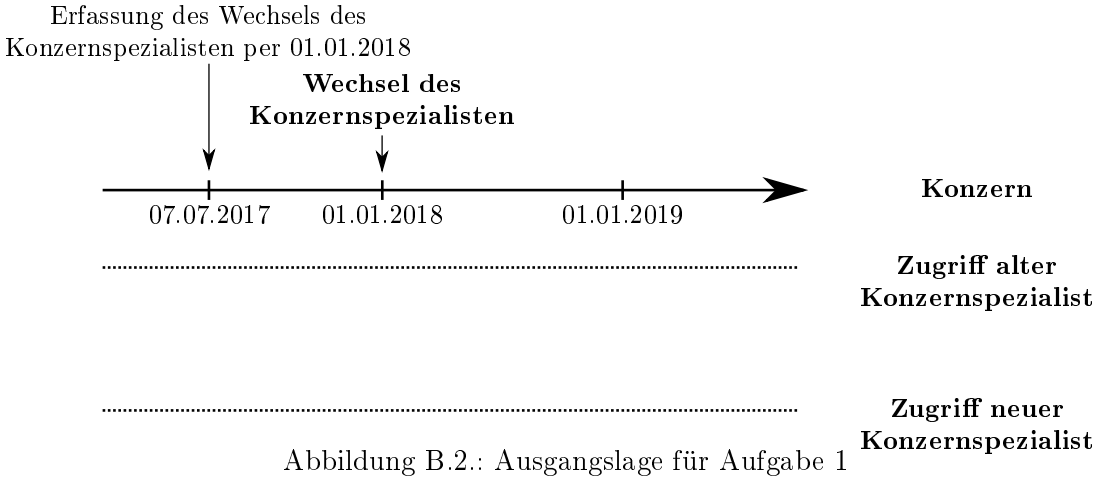

Abbildung B.2.: Ausgangslage für Aufgabe 1

Fragestellung:

- Welcher Benutzer hat zu welchem Zeitpunkt Zugriff auf den Konzern?
- In welcher Zeitspanne haben beide Benutzer den Zugriff?

Beobachtungen

- Die Testperson erkennt die Story sofort aus eigenen Projekten wieder.
- Es ist der Testperson schnell klar, um was es bei dieser Story geht, auch aufgrund seines Vorwissens.
- Inhaltliche Fragen zu der User Story tauchen auf:
  - Es ist nicht ganz klar welche Daten zu den «Konzerndaten» gehören.
  - Der zeitliche Aspekt ist nicht ganz klar. Zum Beispiel zu welchem Zeitpunkt die Abfrage abgesetzt wird.
  - Die Bezeichnung «bis Ende Jahr» irritiert, da nicht klar ist auf was sich das bezieht.
- Die Testperson muss die Story, trotz sehr gutem Vorwissen, mehrmals lesen um zu verstehen wie dies gemeint ist.
- Die langen Sätze (Cohn Template) wirken auf den Benutzer teils unverständlich.
- Aufgrund der Akzeptanzkriterien klären sich einige Fragen bei der Testperson.

Resultat

Nach mehrfachem Lesen kann die Testperson die gestellte Aufgabe innerhalb von sieben Minuten lösen. Somit wäre das Kriterium aus den NFAs erfüllt. Zu beachten ist, dass die Testperson ein gutes Hintergrundwissen zu der vorgelegten User Story hatte, und dies ganz klar bei der Erarbeitung der Lösung half. Die Akzeptanzkriterien wurden von der Testperson als sehr hilfreich empfunden, wogegen die Satzlänge der Story eher verkomplizierend wirkte.

Abbildung B.3 zeigt das von der Testperson vorgeschlagene Resultat. Aufgrund der Irritation bei dem Ausdruck «bis Ende Jahr», war sich die Testperson nicht ganz sicher ob der Zugriff des alten Konzernspezialisten bis Ende 2018 oder weiter hinaus vorhanden ist. Nach genauem Nachlesen der Akzeptanzkriterien wurde es allerdings klar.

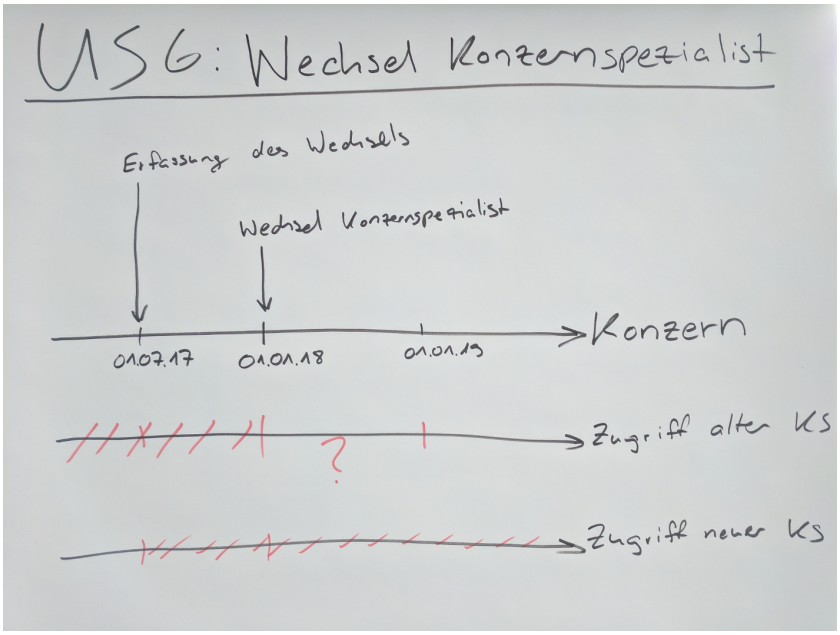

Abbildung B.3.: Benutzertest: Resultat Aufgabe 1

B.4.2. Aufgabe 2: Policy verstehen

Ziel der Aufgabe

Mittels dieser Aufgabe soll überprüft werden ob der technische Consultant eine Policy innerhalb von 10 bis 15 Minuten verstehen kann (TK2).

Verwendet wird die Policy der User Story «US1: Very important Person (VIP) Kunden schützen», welche in Abschnitt 3.2.4 beschrieben ist, da die Fachlichkeit dahinter nicht sehr komplex ist und dem Consultant bereits bekannt ist. Damit wird sichergestellt, dass der Test lediglich das Verständnis der Policy-Syntax und nicht der Fachlichkeit überprüft.

Die Policy sieht folgendermassen aus (Abbildungen B.1 und B.2):

```
<?xml version="1.0" encoding="UTF-8"?>
  <xacml3:Policy
     PolicyId="http://adcubum.com/authorization/boPolicies/vipPolicy"
    RuleCombiningAlgId="rule-combining-algorithm:permit-overrides"
    Version="1.0">
    <xacml3:Description />
    <xacml3:PolicyDefaults>
      <xacml3:XPathVersion>REC-xpath-19991116</xacml3:XPathVersion>
    </xacml3:PolicyDefaults>
    <xacml3:Target>
      <xacml3:AnyOf>
        <xacml3:AllOf>
          <xacml3:Match MatchId="function:integer-equal">
            <xacml3:AttributeValue
              DataType="integer">-3</xacml3:AttributeValue>
            <xacml3:AttributeDesignator
              AttributeId="http://adcubum.com/syrius/bo/metaBoId"
              DataType="integer"
              Category="attribute-category:resource"
              MustBePresent="false" />
          </xacml3:Match>
        </xacml3:AllOf>
      </xacml3:AnyOf>
      <xacml3:AnyOf>
        <xacml3:AllOf>
          <xacml3:Match MatchId="function:string-equal">
            <xacml3:AttributeValue
              DataType="string">READ</xacml3:AttributeValue>
            <xacml3:AttributeDesignator
              AttributeId="http://adcubum.com/syrius/operation"
              DataType="string"
              Category="attribute-category:action"
              MustBePresent="false" />
          </xacml3:Match>
        </xacml3:AllOf>
        <xacml3:AllOf>
          <xacml3:Match MatchId="function:string-equal">
            <xacml3:AttributeValue
              DataType="string">WRITE</xacml3:AttributeValue>
            <xacml3:AttributeDesignator
              AttributeId="http://adcubum.com/syrius/operation"
              DataType="string"
              Category="attribute-category:action"
              MustBePresent="false" />
          </xacml3:Match>
        </xacml3:AllOf>
      </xacml3:AnyOf>
      <xacml3:AnyOf>
        <xacml3:AllOf>
          <xacml3:Match MatchId="function:boolean-equal">
            <xacml3:AttributeValue
              DataType="boolean">true</xacml3:AttributeValue>
            <xacml3:AttributeDesignator
              AttributeId="http://adcubum.com/syrius/bo/partner/isVIP"
              DataType="boolean"
              Category="attribute-category:resource"
              MustBePresent="false" />
          </xacml3:Match>
        </xacml3:AllOf>
      </xacml3:AnyOf>
    </xacml3:Target>
```

Auflistung B.1: XACML-Policy des Benutzertests (Teil 1)

```
    <xacml3:Rule
        Effect="Permit"
        RuleId="allowAccessVIPservice">
      <xacml3:Description />
      <xacml3:Target />
      <xacml3:Condition>
        <xacml3:Apply FunctionId="function:string-is-in" >
          <xacml3:Apply FunctionId="function:string-one-and-only" >
            <xacml3:AttributeDesignator
              AttributeId="http://adcubum.com/syrius/syruser/abteilung"
              DataType="string"
              Category="subject-category:access-subject"
              MustBePresent="false" />
          </xacml3:Apply>
          <xacml3:Apply FunctionId="function:string-bag" >
            <xacml3:AttributeValue
              DataType="string">VIPService</xacml3:AttributeValue>
            <xacml3:AttributeValue
              DataType="string">Geschäftsleitung</xacml3:AttributeValue>
          </xacml3:Apply>
        </xacml3:Apply>
      </xacml3:Condition>
    </xacml3:Rule>
    <xacml3:Rule Effect="Deny"
        RuleId="denyForOthers">
      <xacml3:Description />
      <xacml3:Target />
    </xacml3:Rule>
  </xacml3:Policy>
```

Auflistung B.2: XACML-Policy des Benutzertests (Teil 2)

Vorwissen und Hilfsmittel

- Die Elemente von XACML (*Rules, Target, Policy*) sind der Testperson bekannt oder werden erklärt.
- Benötigte Stammdaten in Form von Tabellen werden der Testperson abgegeben.

Aufgabenstellung

Dem Consultant wird die obige Policy zusammen mit einer Entscheidungsmatrix vorgelegt. Aufgrund der XACML-Policy soll die Testperson nun in der Matrix alle Kombinationsfelder mit P für PERMIT oder D für DENY markiert werden.

Die nachfolgende Abbildung B.4 zeigt eine Matrix, welche die Szenarien abbildet. Jede Zeile in Kombination mit einer beliebigen Spalte, stellt ein Szenario dar, in welchem ein bestimmter Benutzer auf ein bestimmtes Business Object (BO) zugreifen möchte. Die Testperson soll für jeden Fall anhand der Policy bestimmen können, ob der Zugriff gewährt (P = PERMIT) wird oder nicht (D = DENY). Ausser den Stammdaten und der

XACML-Policy sind keine Hilfsmittel zu verwenden. Während des Tests ist die Testperson auf sich alleine gestellt.

Testtabelle Benutzertest 02.05.2017						
In dieser Matrix sollen für jeden Benutzer (Spalten) eingetragen werden welche Objekte (Zeilen) er einsehen kann.						
P = PERMIT						
D = DENY						
	Peter Müller	Claudia HR	Valter I.P. Betreuer	admin		
PARTNER - Patrick Superstar						
VERTRAG - VIP_1_Vertrag_1						
VORBEHALT - VIP_1_Vorbehalt_1						
PARTNER - Mitarbeiter1						
ADRESSE - VIP_2_Adresse_1						

Abbildung B.4.: Leere Benutzertest Matrix

Stammdaten

Folgende Daten werden der Testperson zur Verfügung gestellt um die Matrix auszufüllen. Diese Daten enthalten alle notwendigen Informationen in einer sehr kompakten Form.

Stammdaten User

Hier sind die Daten zu den Benutzern eingetragen.

Username	Abteilung
Peter Müller	Taggeld
Claudia HR	HR
Valter I.P. Betreuer	VIPService
admin	Technische User

Abbildung B.5.: Stammdaten des Benutzertest

Stammdaten BOs

Hier sind alle benötigten Stammdaten zu den BO zu finden. Beziehungen zwischen BOs werden über die BOId und den entsprechenden Fremdschlüssel (FK) gelöst

MetaBo-ID	BoId	Partnerrolle	VIP
-3	Patrick Superstar	Langjähriger Kunde	ja
-3	Peter Müller	Mitarbeiter	nein
		Partner_FK	
-34	VIP_1_Vertrag_1	Patrick Superstar	
-34	Mitarbeiter_1_Vertrag_1	Peter Müller	
-34	VIP_1_Vertrag_2	Patrick Superstar	
		Partner_FK	
-7	VIP_1_Adresse_1	Patrick Superstar	
-7	Mitarbeiter_1_Adresse_1	Peter Müller	
-7	VIP_2_Adresse_1	Peter Müller	
		Partner_FK	
-1277	VIP_1_Behandlung_1	Patrick Superstar	
-1277	Mitarbeiter_1_Behandlung_1	Peter Müller	
		Partner_FK	
-50	VIP_1_Leistungsfall_1	Patrick Superstar	
-50	Mitarbeiter_1_Leistungsfall_1	Peter Müller	
		Partner_FK	
-104	VIP_1_Vorbehalt_1	Patrick Superstar	
-104	Mitarbeiter_1_Vorbehalt_1	Peter Müller	

Abbildung B.6.: Stammdaten des Benutzertest

Beobachtungen

- Die Testperson zeigt sofort eine Affinität zu XML und filtert die relevanten Informationen.
- Die Testperson verschafft sich sehr schnell eine Übersicht, wo was geregelt ist.
- Es ist schnell klar auf welche MetaBOs das *Target* angewendet wird.
- Die Komparatoren wie *string-is-in* oder *string-one-and-only* sind irritierend.
- Die Testperson trifft die Annahme, dass alles was nicht in die Rule fällt, in einem PERMIT resultiert.
- Bedenken bezüglich der Wartberkeit mit riesigen XACML Files werden kundgetan.
- Frage der Testperson wie abhängige Objekte wie Leistungen an das *isVIP* Attribut kommen.

Resultat

Testtabelle Benutzertest 02.05.2017						
In dieser Matrix sollen für jeden Benutzer (Spalten) eingetragen werden welche Objekte (Zeilen) er einsehen kann.						
P = PERMIT						
D = DENY						
	Peter Müller	Claudia HR	Valter I.P. Betreuer	admin		
PARTNER - Patrick Superstar	D	D	P	D		
VERTRAG - VIP_1_Vertrag_1	D	D	D	D		
VORBEHALT - VIP_1_Vorbehalt_1	D	D	D	D		
PARTNER - Mitarbeiter1	D	D	D	D		
ADRESSE - VIP_2_Adresse_1	D	D	D	D		

Abbildung B.7.: Benutzertest: Erwartetes Resultat Aufgabe 2

Abbildung B.8.: Benutzertest: Erhaltenes Resultat Aufgabe 2

Diese Aufgabe wurde von der Testperson in der Hälfte der geschätzten Zeit erledigt. In 5 Minuten hatte der Testkandidat die Matrix ausgefüllt. Die Lösung der Testperson ist in Abbildung B.8 ersichtlich. Im Vergleich zu der erwarteten Lösung in Abbildung B.7, weichen einige Felder voneinander ab. Dieser Zustand ist auf die Annahme der Testperson, dass nicht vom *Target* betroffene Requests automatisch *PERMIT* zurückgeben, zurückzuführen. Nach Auflösung der falschen Annahme korrigierte die Testperson ihre Antworten im Gespräch selbstständig auf die zu erwartenden Werte. Das Testkriterium 2 ist somit erfüllt.

B.4.3. Aufgabe 3: Policy modifizieren

Ziel der Aufgabe

Mit dieser Aufgabe soll abschliessend überprüft werden, ob der Consultant in der Lage ist, innerhalb von 20 Minuten eine Policy zu modifizieren (TK3). Diese Aufgabe erfordert ein tieferes Verständnis der Policy und der Syntax als Aufgabe 2.

Hierfür wird, wie bei Aufgabe 2, die Policy der User Story «US1: VIP Kunden schützen» verwendet. Die Testperson ist somit bereits vertraut mit der Policy und muss sich nicht zuerst noch in eine neue Policy einlesen. Um den Testfall für die Testperson einfach zu halten, reichen Notizen auf den Sourcecodeblättern. Es wird nicht verlangt dass die Testperson XML Code von Hand schreibt. Dies wäre eine Fleissaufgabe und sagt über das Verständnis der Testperson nichts aus. Die Zeit welche gebraucht würde, um den XACML-Code von Hand zu schreiben, wird bei der Auswertung der Aufgabe berücksichtigt.

Vorwissen und Hilfsmittel

- Die Elemente von XACML (*Rules, Target, Policy*) sind der Testperson bekannt oder werden erklärt.
- Benötigte Stammdaten in Form von Tabellen werden der Testperson abgegeben.
- Die verwendete Policy entspricht der Aufgabe 2.

Aufgabenstellung

Die Testperson soll in dieser Aufgabe die XACML-Policy selbst ergänzen. Auf dem bereits in Aufgabe 2 abgegebenen Sourcecode sollen die Änderungen des Testszenarios eingetragen werden. Das Szenario, welches umgesetzt werden muss, wird der Testperson in Form eines Textes abgegeben. Ziel dieser Aufgabe ist es, festzustellen ob der Consultant aufgrund des abgegebenen Textes in der Lage ist, eine bestehenden Policy entsprechend zu modifizieren.

Szenarien:

- Aufgrund von strukturellen Änderungen beim Versicherer werden neu alle Mitarbeiter aus der Abteilung «Human Resources» (HR) dazu berechtigt, Einsicht in alle VIP Partner zu erhalten.
- Gleichzeitig sollen die Abteilung «Human Resources» und «VIP Service» neu Einsicht in alle Leistungen, welche die MetaBOId von *-50* haben, von VIPs erhalten. Die Leistungen verfügen über eine Referenz zum Partnerobjekt und können somit das gleiche Attribut verwenden wie die Partnerobjekte selbst. Diese Anpassungen sollen an der vorhandenen Syntax vorgenommen werden.

Beispiellösungen:

Um der Abteilung «Human Resources» (HR) zugriff auf alle VIP Partner zu geben muss der sogenannte *Bag-of-String* um den Begriff **«HR»** erweitert werden. In der folgenden Auflistung B.3 ist dies umgesetzt.

```
<xacml3:Condition>
    <xacml3:Apply FunctionId="function:string-is-in" >
        <xacml3:Apply FunctionId="function:string-one-and-only" >
        <xacml3:AttributeDesignator
            AttributeId="http://adcubum.com/syrius/syruser/abteilung"
            DataType="string"
            Category="subject-category:access-subject"
            MustBePresent="false" />
        </xacml3:Apply>
        <xacml3:Apply FunctionId="function:string-bag" >
            <xacml3:AttributeValue
                DataType="string">VIPService</xacml3:AttributeValue>
            <xacml3:AttributeValue
                DataType="string">Geschäftsleitung</xacml3:AttributeValue>
            <xacml3:AttributeValue
                DataType="string">HR</xacml3:AttributeValue>
                </xacml3:Apply>
        </xacml3:Apply>
</xacml3:Condition>
```

Auflistung B.3: Ergänztes *Bag-of-String* der VIP-Policy

Die Einsicht auf die Leistungen der VIPs kann über das *Target* der Policy gelöst werden. In dem *<Any-Of>* Konstrukt muss auf die MetaBO-ID der Leistungen (-50) überprüft werden. In der Auflistung B.4 ist eine mögliche Lösung aufgezeigt.

```
<xacml3:AnyOf>
    <xacml3:AllOf>
        <xacml3:Match MatchId="function:integer-equal">
            <xacml3:AttributeValue
                DataType="integer">-3</xacml3:AttributeValue>
            <xacml3:AttributeDesignator
                AttributeId="http://adcubum.com/syrius/bo/metaBoId"
                DataType="integer"
                Category="attribute-category:resource"
                MustBePresent="false" />
        </xacml3:Match>
    </xacml3:AllOf>
    <xacml3:AllOf>
        <xacml3:Match MatchId="function:integer-equal">
            <xacml3:AttributeValue
                DataType="integer">-50</xacml3:AttributeValue>
            <xacml3:AttributeDesignator
                AttributeId="http://adcubum.com/syrius/bo/metaBoId"
                DataType="integer"
                Category="attribute-category:resource"
                MustBePresent="false" />
        </xacml3:Match>
    </xacml3:AllOf>
</xacml3:AnyOf>
```

Auflistung B.4: Verändertes *Target* mit Überprüfung auf Leistungen

Beobachtungen

- Die Aufgaben werden innerhalb einer Minute aus den Szenarien extrahiert.
- Die Testperson findet nach einer weiteren Minute den richtigen Ort um eine Abteilung hinzuzufügen.
- Nach drei Minuten ist die Anpassung an der Targetdefinition erfolgt.
- Bei der Targetdefinition ist der Testperson nicht klar, ab welcher Zeile das *Target* definiert wird.
- Es tritt Verwirrung auf bei den Integer-Equal Komparatoren und ob dort auch bei der zu schreibenden Erweiterung Integer verwendet werden.
- Die Testperson bringt von sich aus den Fall der Schutzpfade ein. Die Testperson fragt, wie die vom Partner abhängigen Objekte das *isVIP* Attribut des Partners erhalten.
- Die Testperson äussert bedenken bezüglich der Fehlerfindung, wenn viele solcher Policies im Einsatz sind.

Resultat

Die geforderten Mutationen an der Policy konnten aus der bestehenden Policy abgeleitet werden. Deshalb kann aus der gelösten Aufgabe nicht geschlossen werden, dass die Testperson selbst eine komplett neue Policy verfassen kann. Laut Aussagen der Testperson ist dies nicht der Fall. Was zu dieser Fähigkeit fehlt, ist das vertiefte Wissen über XACML und das XACML-API. Für den Benutzertest war dieses Wissen keine Voraussetzung. Insgesamt kann TK3 in diesem Test als erfüllt angesehen werden, da die Testperson die geforderten Änderungen in der Policy innerhalb der Zeitlimite der NFA vorgenommen hat. Bei dieser Einschätzung wurde die Zeit, welche für das Schreiben der XML von Hand nötig wäre, berücksichtigt.

Die Resultate der Testperson sind in den nachfolgenden Abbildungen B.9, B.10, und B.11 ersichtlich. Die Aufgabe wurde auf mehreren Seiten gelöst, weshalb das Resultat in drei Abschnitte unterteilt wurde.

```
<?xml version="1.0" encoding="UTF-8"?>
  <xacml3:Policy
     PolicyId="http://adcubum.com/authorization/boPolicies/vipPolicy"
    RuleCombiningAlgId="rule-combining-algorithm:permit-overrides"
    Version="1.0">
    <xacml3:Description />
    <xacml3:PolicyDefaults>
      <xacml3:XPathVersion>REC-xpath-19991116</xacml3:XPathVersion>
    </xacml3:PolicyDefaults>
    <xacml3:Target>
      <xacml3:AnyOf>
        <xacml3:AllOf>
          <xacml3:Match MatchId="function:integer-equal">
            <xacml3:AttributeValue
              DataType="integer">-3</xacml3:AttributeValue>
            <xacml3:AttributeDesignator
              AttributeId="http://adcubum.com/syrius/bo/metaBoId"
              DataType="integer"
              Category="attribute-category:resource"
              MustBePresent="false" />
          </xacml3:Match>
        </xacml3:AllOf>
      </xacml3:AnyOf>
      <xacml3:AnyOf>
        <xacml3:AllOf>
```

Abbildung B.9.: Benutzertest: Resultat Aufgabe 3

```
26          <xacml3:Match MatchId="function:string-equal">
27            <xacml3:AttributeValue
28              DataType="string">READ</xacml3:AttributeValue>
29            <xacml3:AttributeDesignator
30              AttributeId="http://adcubum.com/syrius/operation"
31              DataType="string"
32              Category="attribute-category:action"
33              MustBePresent="false" />
34          </xacml3:Match>
35        </xacml3:AllOf>
36        <xacml3:AllOf>
37          <xacml3:Match MatchId="function:string-equal">
38            <xacml3:AttributeValue
39              DataType="string">WRITE</xacml3:AttributeValue>
40            <xacml3:AttributeDesignator
41              AttributeId="http://adcubum.com/syrius/operation"
42              DataType="string"
43              Category="attribute-category:action"
44              MustBePresent="false" />
45          </xacml3:Match>
46        </xacml3:AllOf>
47      </xacml3:AnyOf>
48      <xacml3:AnyOf>
49        <xacml3:AllOf>
50          <xacml3:Match MatchId="function:boolean-equal">
51            <xacml3:AttributeValue
52              DataType="boolean">true</xacml3:AttributeValue>
53            <xacml3:AttributeDesignator
54              AttributeId="http://adcubum.com/syrius/bo/partner/isVIP"
55              DataType="boolean"
56              Category="attribute-category:resource"
57              MustBePresent="false" />
58          </xacml3:Match>
59        </xacml3:AllOf>
60      </xacml3:AnyOf>
61    </xacml3:Target>
62    <xacml3:Rule
63        Effect="Permit"
64        RuleId="allowAccessVIPservice">
65      <xacml3:Description />
66      <xacml3:Target />
67      <xacml3:Condition>
68        <xacml3:Apply FunctionId="function:string-is-in" >
69          <xacml3:Apply FunctionId="function:string-one-and-only" >
```

Abbildung B.10.: Benutzertest: Resultat Aufgabe 3

```
70            <xacml3:AttributeDesignator
71              AttributeId="http://adcubum.com/syrius/syruser/abteilung"
72              DataType="string"
73              Category="subject-category:access-subject"
74              MustBePresent="false" />
75          </xacml3:Apply>
76          <xacml3:Apply FunctionId="function:string-bag" >
77            <xacml3:AttributeValue
78              DataType="string">VIPService</xacml3:AttributeValue>
79            <xacml3:AttributeValue
80              DataType="string">Geschäftsleitung</xacml3:AttributeValue>
81          </xacml3:Apply>
82        </xacml3:Apply>
83      </xacml3:Condition>
84    </xacml3:Rule>
85    <xacml3:Rule Effect="Deny"
86        RuleId="denyForOthers">
87      <xacml3:Description />
88      <xacml3:Target />
89    </xacml3:Rule>
90  </xacml3:Policy>
```

Abbildung B.11.: Benutzertest: Resultat Aufgabe 3

B.5. Bewertung

B.5.1. Gesamteindruck und Feedbacks des Consultants

Die User Stories sind mit genauem Lesen nachvollziehbar. Obwohl das Template für User Stories von Cohn [9] dem Stand der Technik entspricht und weit verbreitet ist, wirkten sich die gut strukturierten aber sehr langen Sätze negativ auf die Verständlichkeit aus. Mit Hilfe der Akzeptanzkriterien konnte die Testperson bei Unklarheiten ihre Annahmen validieren. Die XACML Syntax wirkt auf jemanden ohne Softwareentwickler-Hintergrund auf den ersten Blick komplex. Allerdings fand die Testperson mit einem guten Auge die wichtigen Informationen auch ohne XACML-Kenntnisse. Für eine detaillierte Analyse der Policy ist Wissen über den Aufbau der XACML-Syntax unabdingbar. Die im Anschluss an die Aufgaben kurz gezeigte Policy, welche mit dem Abbreviated Language For Authorization (ALFA)-Plugin geschrieben wurde, beurteilte die Testperson als sehr viel lesbarer im Vergleich mit XACML. Der Consultant war sehr interessiert an der Lösung, und meinte dass Kunden die neuen Möglichkeiten der Autorisierungslösung schätzen werden.

Gleichzeitig äusserte die Testperson auch gewisse Bedenken:

- Für Kunden könnte mehr Aufwand entstehen bei der Wartung der separaten Komponente.
- Die gewonnene Dynamik der ABAC-Policies könnte die Kunden dazu verleiten, noch komplexere Szenarien abzubilden als dies heute möglich ist. Dies könnte sich negativ auf die Wartbarkeit auswirken.
- Bei zu vielen Files und Policies könnte die Wartung schwieriger werden als bei der jetzigen Lösung.

B.5.2. Erfüllung der Testkriterien

Tabelle B.1.: Erfüllung der Testkriterien

Kriterium	Status	Bemerkungen
TK1	Erfüllt	• Lange Sätze eher irritierend • Achten auf Verwendung von Beispielausdrücken («bis Ende Jahr») • Akzeptanzkriterien sind sehr hilfreich
TK2	Erfüllt	• Grundwissen in XACML benötigt • Braucht ein Auge für wichtige Informationen • Deutlich weniger Zeit benötigt als 10 Minuten
TK3	Erfüllt	• Innerhalb der vorgegebenen Zeit gelöst • Benötigt Wissen zu einzelnen XACML Elementen

B.6. Massnahmen

Der Benutzertest brachte viele neue Erkenntnisse. Alle drei Testkriterien wurden durch die Testperson erfüllt, was für die gute Definition der NFAs spricht. Trotzdem gab es Beobachtungen, welche zu den folgenden Massnahmen geführt haben:

- Fachbegriffe aus der Versicherungsbranche werden im Glossar erklärt um die Verständlichkeit der User Stories zu erhöhen.
- Lange Sätze in den User Stories könnten zugunsten der schnelleren Verständlichkeit in kürzere Sätze aufgeteilt werden. Auf diese Massnahme wird verzichtet, um die Struktur des Templates beizubehalten.
- Um riesige XML-Dateien zu verhindern wird pro Policy eine XML-Datei erstellt.

C. Aufgabenstellung

Die folgenden beiden Abschnitte sind Auszüge aus der originalen Aufgabenstellung, welche am Anfang des Projekts definiert wurde.

C.1. Ausgangslage

SYRIUS ist ein ERP-System für das Versicherungswesen mit hohen Sicherheitsanforderungen. Die Datenzugriffsautorisierung ist in SYRIUS in der Persistenzschicht gelöst. Während der Studienarbeit Ärchitectural Refactoring der Data Access Security"wurde analysiert, an welchen Stellen die Persistenzschicht angepasst werden muss, um ein externes Autorisierungssystem aufzurufen. Ausserdem wurde in der Studienarbeit eine REST-basierte Autorisierungsschnittstelle entworfen.

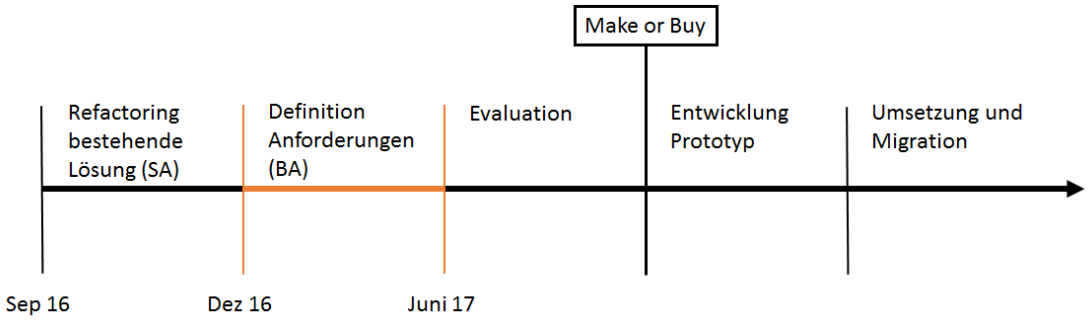

Abbildung C.1.: Zeitplan der Adcubum AG für die Einführung des neuen Systems

Nun sollen innerhalb dieser Bachelorarbeit die Anforderungen an das neue attributbasierte Autorisierungssystem und sein grundlegender fachlich-technischer Aufbau erarbeitet werden.

C.2. Ziele der Arbeit und Liefergegenstände

Im Prozess der Umstellung von einem rollenbasierten auf ein attributbasiertes Berechtigungssystem, sollen in dieser Arbeit die Anforderungen und konzeptionellen Grundlagen erarbeitet werden, um später eine Produktevaluation durchführen zu können und zu einem «Make or Buy» Entscheid zu gelangen. Diese Anforderungen und Konzepte sollen prototypisch umgesetzt werden.

Im Zuge des Redesigns sollen die Anforderungen an den Schutz der Daten, und somit auch an das neue System möglichst repräsentativ und konkret festgehalten und beschrieben werden. Auf Basis der definierten Anforderungen soll erarbeitet werden, welche Informationen von externen Datenquellen in der Decision-Engine des zu entwickelnden Systems benötigt werden und wie diese effizient verfügbar gemacht werden können. Weiterführend soll erarbeitet werden, wie die Anforderungen in Form von Policies syntaktisch formuliert werden können.

Zusätzlich soll an einem konkreten Use Case aufgezeigt werden, ob und wie die Anforderungen an das System mit einer attributbasierten Autorisierung realisierbar sind. Dafür soll ein Prototyp implementiert werden. Der Prototyp bezieht die Daten der externen Systeme über Mock-Schnittstellen. Für den weiteren Projektverlauf soll zudem die Migrationsfähigkeit, betreffend einer automatisierten oder teilautomatisierten Migration der bestehenden Lösung auf die neue attributbasierte Abbildung, aufgezeigt werden.

Die kritischen Erfolgsfaktoren für diese Arbeit wurden wie folgt definiert:

- Die Benutzeranforderungen werden repräsentativ und aussagekräftig evaluiert und dargestellt.
- Anforderungen können mit Policies möglichst einfach abgebildet werden. Die Sprache bzw. die Syntax ist nicht unnötig komplex und damit für Kunden und Consultants verständlich.
- Es soll abgeschätzt werden, wie die Laufzeiteigenschaften der vorgeschlagenen Informationsbeschaffungslösung aussehen werden. Ausserdem soll ersichtlich sein, an welchen Stellen die Performance, z.B. durch Caching-Mechanismen, verbessert werden kann.

Literaturverzeichnis

[1] Adcubum AG. Anbindung Fremdbibliotheken. `https://confluence.adcubum.com/display/SYRFRONT/Anbindung+Fremdbibliotheken`. [Adcubum INTERN; Abgerufen am 21.03.2017].

[2] Adcubum AG. SYRIUS BO-Dokumentation. Kann in jeder SYRIUS-Instanz geöffnet werden. Siehe Instanzenübersicht: `https://syrcon.internal.adcubum.com`. [Adcubum INTERN; Abgerufen am 15.05.2017].

[3] Axiomatics. Attribute Based Access Control (ABAC). `https://www.axiomatics.com/attribute-based-access-control/`. [Online; Abgerufen am 28.05.2017].

[4] Axiomatics. Axiomatics Developer Tools. `https://www.axiomatics.com/product/developer-tools-and-apis/`. [Online; Abgerufen am 24.05.2017].

[5] Axiomatics. Axiomatics is the leader in dynamic authorization solutions. `https://www.axiomatics.com/`. [Online; Abgerufen am 14.05.2017].

[6] Axiomatics. Axiomatics releases free plugin for the Eclipse IDE to author XACML3.0 policies. https://www.axiomatics.com/news/axiomatics-releases-free-plugin-for-the-eclipse-ide-to-author-xacml3-0-policies/. [Online; Abgerufen am 24.05.2017].

[7] Axiomatics. Going on vacation, how can I implement delegation in XACML? `https://www.axiomatics.com/blog/going-on-vacation-how-can-i-implement-delegation-in-xacml/`. [Online; Abgerufen am 14.05.2017].

[8] Axiomatics. XACML Reference Architecture. `https://www.axiomatics.com/blog/xacml-reference-architecture/`. [Online; Abgerufen am 23.05.2017].

[9] Mike Cohn. *User Stories Applied: For Agile Software Development*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[10] John DeTreville. Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, SP '02, pages 105–, Washington, DC, USA, 2002. IEEE Computer Society.

[11] Elastic. Elasticsearch. `https://www.elastic.co/`. [Online; Abgerufen am 27.05.2017].

[12] Free Software Foundation. GNU General Public License, version 3. `http://www.gnu.org/licenses/gpl.html`. [Online; Abgerufen am 26.05.2017].

[13] Free Software Foundation. Gnu lesser general public license v3.0. `http://www.gnu.org/licenses/lgpl.html`. [Online; Abgerufen am 26.05.2017].

[14] The Apache Software Foundation. Apache Incubator. `http://incubator.apache.org/projects/openaz.html`. [Online; Abgerufen am 24.05.2017].

[15] The Apache Software Foundation. Apache software license 2.0. `https://www.apache.org/licenses/LICENSE-2.0`. [Online; Abgerufen am 26.05.2017].

[16] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[17] Inc. GitHub. Github. `https://github.com`. [Online; Abgerufen am 24.05.2017].

[18] Inc. GitHub. Mirror of Apache OpenAZ. `https://github.com/apache/incubator-openaz`. [Online; Abgerufen am 24.05.2017].

[19] https://jan.newmarch.name. HOPP: Half Object plus Protocol. `https://jan.newmarch.name/distjava/hopp/lecture.html`, 2007. [Online; Abgerufen am 26.05.2017].

[20] Marc Hueffmeyer and Ulf Schreier. Restacl: An access control language for restful services. In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*, ABAC '16, pages 58–67, New York, NY, USA, 2016. ACM.

[21] Marc Hüffmeyer and Ulf Schreier. *Analysis of an Access Control System for RESTful Services*, pages 373–380. Springer International Publishing, Cham, 2016.

[22] WSO2 Inc. WSO2. `http://wso2.com/`. [Online; Abgerufen am 17.05.2017].

[23] WSO2 Inc. WSO2 Balana Implementation. `https://github.com/wso2/balana`. [Online; Abgerufen am 17.05.2017].

[24] Microsoft Jason Hogg. SecPAL – Access Control for Grid Computing Environments. `https://blogs.msdn.microsoft.com/thehoggblog/2007/04/20/secpal-access-control-for-grid-computing-environments/`. [Online; Abgerufen am 25.05.2017].

[25] Shirish Joshi. Comparing Oracle Query Performance for Faster Applications. `http://www.devx.com/dbzone/Article/40778/0/page/3`. [Online; Abgerufen am 25.05.2017].

[26] JUnit. JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks. `http://junit.org`. [Online; Abgerufen am 20.05.2017].

[27] Stefan Kapferer. WSO2 Balana, Github Issue: «No possibility to change the root PolicyCombiningAlgorithm». `https://github.com/wso2/balana/issues/70`. [Online; Abgerufen am 24.05.2017].

[28] Stefan Kapferer. Architectural Refactoring der Data Access Security. Semester Thesis, University of Applied Sciences HSR, Rapperswil Switzerland, 2016. Publication: `https://eprints.hsr.ch/564/`.

[29] Tom Kyte. Consider Cost or Time - trying to compare the COST of two queries. `https://asktom.oracle.com/pls/asktom/f?p=100:11:0::::P11_QUESTION_ID:313416745628`. [Online; Abgerufen am 25.05.2017].

[30] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[31] LDAP.com. Choosing an LDAP Server. `https://www.ldap.com/choosing-an-ldap-server`. [Online; Abgerufen am 14.05.2017].

[32] Mark Massias. Performance Comparison of Intersystems Caché and Oracle in a Data Mart Application. `http://www.intersystems.com/assets/datamart_wp-ee478edf530b40311ef506615c0da74d.pdf`. [Online; Abgerufen am 25.05.2017].

[33] Michael Stal. Software architecture refactoring. In Tutorial, in The International Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA). `http://stal.blogspot.ch/2007/01/architecture-refactoring.html`, 2008.

[34] Microsoft. Security Policy Assertion Language. `https://secpal.codeplex.com/`. [Online; Abgerufen am 24.05.2017].

[35] Microsoft. SecPAL Project Homepage. `http://research.microsoft.com/projects/secpal`. [Online; Abgerufen am 25.05.2017].

[36] Milstein Munakami. WSO2 Balana, Github Issue: «Delegation profile with Balana». `https://github.com/wso2/balana/issues/25`. [Online; Abgerufen am 24.05.2017].

[37] Nate Lord. What is the Principle of Least Privilege (POLP)? A Best Practice for Information Security and Compliance. https://digitalguardian.com/blog/what-principle-least-privilege-polp-best-practice-information-security-and-compliance, 2017. [Online; Abgerufen am 07.06.2017].

[38] OASIS. Abbreviated Language for Authorization Version 1.0. `https://www.oasis-open.org/committees/download.php/55228/alfa-for-xacml-v1.0-wd01.doc`. [Online; Abgerufen am 24.05.2017].

[39] OASIS. OASIS - Advancing open standards for the information society.

https://www.oasis-open.org/. [Online; Abgerufen am 21.05.2017].

[40] National Institute of Standards and Technology NIST. Generally Accepted Principles and Practices for Security Information Technology Systems. `http://csrc.nist.gov/publications/nistpubs/800-14/800-14.pdf`, 1996. [NIST Special Publication 800-14].

[41] National Institute of Standards and Technology NIST. Guide to Attribute Based Access Control (ABAC) Definition and Considerations. `http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf`, 2014. [NIST Special Publication 800-162].

[42] Oracle. Ask The Oracle Masters (AskTOM). `https://asktom.oracle.com/pls/asktom/f?p=100:1:::::::`. [Online; Abgerufen am 25.05.2017].

[43] Oracle. Oracle - Integrated Cloud Applications and Platform Services. `https://www.oracle.com/index.html`. [Online; Abgerufen am 28.05.2017].

[44] Oracle. Oracle SQL Developer. `http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index-097090.html`. [Online; Abgerufen am 17.05.2017].

[45] Open Web Application Security Project (OWASP). Principle Of Least Privilege. `https://www.owasp.org/index.php/Least_privilege`, 2009. [Online; Abgerufen am 07.06.2017].

[46] A. Ed-Dbali P. Deransart and L. Cervoni. *Prolog: The Standard*. Springer-Verlag Berlin Heidelberg, 1996.

[47] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis. Microservices in practice, part 1: Reality check and service design. *IEEE Software*, 34(1):91–98, Jan 2017.

[48] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis. Microservices in practice, part 2: Service integration and sustainability. *IEEE Software*, 34(2):97–104, Mar 2017.

[49] Andrew Pimlott and Oleg Kiselyov. *Soutei, a Logic-Based Trust-Management System*, pages 130–145. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[50] Torsten Priebe, Wolfgang Dobmeier, Björn Muschall, Günther Pernul, and Lehrstuhl Für Wirtschaftsinformatik I. ABAC – Ein Referenzmodell für attributbasierte Zugriffskontrolle. In *In Proceedings Sicherheit 2005*, 2005.

[51] Chris Rupp and die SOPHISTen. *Requirements-Engineering und -Management*. Carl Hanser Verlag GmbH & Co. KG, 2014.

[52] Ravi Sandhu. The authorization leap from rights to attributes: Maturation or cha-

os? In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, SACMAT '12, pages 69–70, New York, NY, USA, 2012. ACM.

[53] Santésuisse. Webseite des Verbandes Santésuisse. `https://www.santesuisse.ch/`. [Online; Abgerufen am 14.05.2017].

[54] Jim Sermersheim. Lightweight Directory Access Protocol (LDAP): The Protocol. RFC 4511, RFC Editor, June 2006.

[55] Jefferson O. Silva, Eduardo M. Guerra, and Clovis T. Fernandes. *An Extensible and Decoupled Architectural Model for Authorization Frameworks*, pages 614–628. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[56] OASIS Standard. eXtensible Access Control Markup Language (XACML) Version 3.0. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`. [Online; Abgerufen am 20.05.2017].

[57] Michael Stone. Soutei-Implementation. `https://github.com/mstone/soutei`. [Online; Abgerufen am 25.05.2017].

[58] Inc. Sun Microsystems. Sun microsystems license. `http://sunxacml.sourceforge.net/license.txt`. [Online; Abgerufen am 26.05.2017].

[59] Inc. Sun Microsystems. Sun's xacml implementation. `http://sunxacml.sourceforge.net`. [Online; Abgerufen am 27.05.2017].

[60] Wolfgang Giersche. APPLICATION SECURITY - FROM A DEVELOPER'S POINT OF VIEW). `https://github.com/smurve/hsr2015/raw/master/SpringSecurity/Security.pdf`, 2015. [Online; Abgerufen am 28.05.2017].

[61] xacmlinfo.org. XACML for Authorization. `http://xacmlinfo.org/category/balana/`. [Online; Abgerufen am 24.05.2017].

[62] Zhongyuan Xu and Scott D. Stoller. Mining attribute-based access control policies. *CoRR*, abs/1306.2401, 2013.

[63] U. Zdun, R. Capilla, H. Tran, and O. Zimmermann. Sustainable Architectural Design Decisions. https://www.infoq.com/articles/sustainable-architectural-design-decisions. [Online; Abgerufen am 25.05.2017].

[64] Olaf Zimmermann. Architectural Refactoring for the Cloud (ARC). `https://www.ifs.hsr.ch/index.php?id=12044&L=4`, 2016. [Online; Abgerufen am 27.05.2017].

[65] Olaf Zimmermann. Architectural refactoring for the cloud: a decision-centric view on cloud migration. *Computing*, 99(2):129–145, 2017.

Glossar

Dieses Glossar enthält alle in dieser Arbeit verwendeten Fachbegriffe. Dies sind sowohl technische Fachbegriffe, als auch fachliche Begriffe aus dem Versicherungswesen. Begriffe, welche spezifisch im SYRIUS Umfeld und dessen Domänenmodell verwendet werden, sind mit «*(SYRIUS)*» markiert.

ABAC Attribute-Based Access Control (ABAC) ist ein Zugriffskontroll-Paradigma [41, 50, 3] bei welchem Zugriffsrechte über die Attribute der zu schützenden Objekte und der Benutzer kontrolliert werden. 1–3, 9–12, 15, 20, 24, 35, 36, 38, 40, 44–46, 48, 50, 51, 54, 55, 61, 84, 86, 94, 100–102, 106–108, 110, 123, 140, 148, 153–155, 158

ALFA Abbreviated Language For Authorization (ALFA) [4] ist eine von Axiomatics [5] entwickelte DSL, um Policies in XACML zu verfassen. Während der ALFA-Standard öffentlich ist, benötigt das Plugin um Policies aus ALFA in XACML zu konvertieren für kommerzielle Nutzung eine entsprechende Lizenz. 46–48, 81–83, 108, 139, 148, 155, 160

Attribute Engineering Der Begriff *Attribute Engineering* [28, 52] wird im Kontext von ABAC häufig für das Entwickeln der Attribute, welche nötig sind um eine Policy für einen bestimmten fachlichen Fall abzubilden, verwendet. Wie beim Programmieren gibt es hier in der Regel verschiedene Lösungsvarianten und es ist das Ziel eine Lösung zu finden, welche möglichst gut wartbar ist. 42, 86, 101

Aufgabe *(SYRIUS)* Eine Aufgabe wird in SYRIUS auch als *Activity* bezeichnet. Anfallende Aufgaben wie zum Beispiel «Falldossier bearbeiten», «Kostengutschrift nachbearbeiten», oder «Rechnung bearbeiten» werden zwingend einem Postkorb zugewiesen. 28, 30, 33, 34, 41, 92, 148, 150, 152, 153

Balana Balana ist eine Open Source Java-Implementation eines Policy Decision Point (PDP) für XACML. Das Projekt wurde von WSO2 [22] entwickelt, und unterstützt den neusten XACML 3.0 Standard. 40, 49–51, 75, 80

Batchjob *(SYRIUS)* Ein *Batchjob* ist eine immer wieder anfallende Aufgabe, welcher automatisiert ausgeführt wird. Als Beispiel dienen Auswertungen von Datenbanken, oder regelmässige Imports von Stammdaten in SYRIUS. 31

Berechtigungsadministrator Ein Berechtigungsadministrator ist ein Angestellter des Versicherers und kümmert sich um die Rechteverteilung in SYRIUS. Er verteilt be-

nötigte Rechte in SYRIUS an Mitarbeiter des Versicherers und überprüft ob die vergebenen Rechte noch benötigt werden. 32

BO *(SYRIUS)* Business Objects (BOs) bezeichnen in SYRIUS gespeicherte Dateneinträge in der Datenbank wie zum Beispiel Partner, Leistung oder weitere Objekte aus dem Datenmodell von SYRIUS. Der Typ des BOs ist über die MetaBOID definiert. 5, 6, 12, 17, 18, 23, 29, 41–43, 46, 52–55, 57, 63, 64, 68, 69, 71, 77, 84–90, 92, 101, 102, 109, 112, 113, 122, 130, 143, 149, 151–155, 160

Datenschutzbeauftragter Der Datenschutzbeauftragte einer Versicherung stellt sicher dass die Datenschutzgesetze und Bestimmungen eingehalten werden. Die Auslegung des Datenschutzgesetz (DSG) ist meistens Teil seiner Aufgabe, wenn es um die Verteilung von Benutzerrechten in SYRIUS geht. 32

diagnosebezogene Daten Synonym für Diagnosedaten. 28, 29, 44, 88

Diagnosedaten Diagnosedaten verweisen auf eine Menge von Daten aus SYRIUS, welche mit einer Krankheitsdiagnose zusammenhängen. Diese Daten sind sehr sensibel, da daraus auf die Gesundheit des Kunden geschlossen werden kann. Deshalb sind Diagnosedaten als Sensitive Personal Information (SPI) anzusehen. Ein Synonym für Diagnosedaten ist der Ausdruck *diagnosebezogenen Daten*. Beide Begriffe sind in der Krankenversicherungsbranche sehr verbreitet. Welche BOs und Attribute in SYRIUS genau geschützt werden, kann der BO-Dokumentation [2] entnommen werden. 28, 29, 33, 42, 44, 149

DSL Auszug Wikipedia: Eine domänenspezifische Sprache (englisch domain-specific language, kurz DSL) oder anwendungsspezifische Sprache ist eine formale Sprache, die zur Interaktion zwischen Menschen und digital arbeitenden Computern (Computersprache) für ein bestimmtes Problemfeld (die sogenannte Domäne) entworfen und implementiert wird. Beim Entwurf einer DSL wird man bemüht sein, einen hohen Grad an Problemspezifität zu erreichen: die Sprache soll alle Probleme der Domäne darstellen können und nichts darstellen können, was außerhalb der Domäne liegt. Dadurch ist sie durch Domänenspezialisten ohne besonderes Zusatzwissen bedienbar. Das Gegenteil einer domänenspezifischen Sprache ist eine universell einsetzbare Programmiersprache, wie C und Java, oder eine universell einsetzbare Modellierungssprache, wie UML. 45, 46, 48, 81–83, 108, 148

EJB Wikipedia: Enterprise JavaBeans (EJB) sind standardisierte Komponenten innerhalb eines Java-EE-Servers (Java Enterprise Edition). Sie vereinfachen die Entwicklung komplexer mehrschichtiger verteilter Softwaresysteme mittels Java. Mit Enterprise JavaBeans können wichtige Konzepte für Unternehmensanwendungen, z. B. Transaktions-, Namens- oder Sicherheitsdienste, umgesetzt werden, die für die Geschäftslogik einer Anwendung nötig sind. 5, 155, 158

Entitlement Unter «Entitlement» fallen sämtliche Berechtigungsfeatures welche nicht

den Objektschutz betreffen und lediglich dazu dienen, das User Interface (UI) für den SYRIUS-Benutzer einfacher zu gestalten. Sie dienen dem Ziel, dem Benutzer nur diejenigen UI-Elemente anzuzeigen, welche er für seine Arbeit benötigt. Aus Sicht des Datenschutzes wäre es nicht problematisch wenn der Benutzer diese Daten sehen würde. Im Zusammenhang mit dieser Bachelorarbeit geht es hier um die Abgrenzung dieser Features, wie zum Beispiel den Taskschutz oder den Registerkartenschutz. 12

Fachbereich Ein Fachbereich ist bei einem Versicherer zuständig für die Abwicklung von Aufgaben aus einem Versicherungsbereich wie zum Beispiel Taggelder, Sachversicherungen, Unfallversicherung oder medizinische Behandlungen. Um die Aufgaben eines Fachbereiches zu erledigen, wird ein sehr spezifisches Fachwissen vorausgesetzt. Sachbearbeiter wechseln nur sehr selten den Fachbereich aufgrund des nötigen Fachwissens und der angeeigneten Fähigkeiten. 34, 153

Falldossier Ein Fall- oder Schadendossier bündelt verschiedene Leistungen, die aus dem gleichen Ereignis (Schaden oder Fall) entstanden sind oder die gleiche Ursache haben. Jedes Fall- oder Schadendossier beinhaltet die Leistungen zu einem Schadenfall. Im Bereich der Unfall- und Krankenversicherung wird das Falldossier verwendet. Im Bereich der Sachversicherung wird das Schadendossier verwendet. 29

Familienoberhaupt *(SYRIUS)* Das Familienoberhaupt ist in SYRIUS ein Partner. Meistens ist dies ein Elternteil einer versichterten Familie. Das Familienoberhaupt ist zum Beispiel für die Rechnungen der Familie zuständig. Über ein Familienoberhaupt können gewisse Konditionen an die restlichen Mitglieder der Familie weitergegeben werden. 24

Geschäftsrolle Eine Geschäftsrolle beschreibt die Aufgabe eines Mitarbeiters innerhalb der Firma. Üblicherweise ist die Geschäftsrolle abhängig von der Abteilung (Organisationseinheit (OE)) und den verschiedenen Stellen innerhalb der OE. Über die Geschäftsrolle wird zum Beispiel bestimmt, welche Postkörbe der Mitarbeiter bearbeiten kann. 30

GPL Auszug Wikipedia: Die GNU General Public License (kurz GNU GPL oder GPL) ist die am weitesten verbreitete Softwarelizenz, die einem gewährt, die Software auszuführen, zu studieren, zu ändern und zu verbreiten (kopieren). 49

IIOP Wikipedia: General Inter-ORB Protocol (GIOP) bezeichnet ein abstraktes Protokoll zur Kommunikation von Object Request Brokern (ORBs) im Bereich des Verteilten Rechnens. Bei GIOP handelt es sich um ein in CORBA 2.0 definiertes abstraktes Protokoll zur Kommunikation zwischen ORBs. Das GIOP ist unabhängig vom verwendeten Transportprotokoll. Das Internet Inter-ORB Protocol (IIOP) ist eine Spezialisierung des GIOP auf TCP/IP als Transportprotokoll und muss von jeder CORBA-Implementierung unterstützt werden. 5, 155

Java EE Wikipedia: Java Platform, Enterprise Edition, abgekürzt Java EE oder früher J2EE, ist die Spezifikation einer Softwarearchitektur für die transaktionsbasierte Ausführung von in Java programmierten Anwendungen und insbesondere Web-Anwendungen. Sie ist eine der großen Plattformen, die um den Middleware-Markt kämpfen. Größter Konkurrent ist dabei die .NET-Plattform von Microsoft. 5

JDBC Wikipedia: Java Database Connectivity (JDBC, englisch für Java Datenbankverbindungsfähigkeit) ist eine Datenbankschnittstelle der Java-Plattform, die eine einheitliche Schnittstelle zu Datenbanken verschiedener Hersteller bietet und speziell auf relationale Datenbanken ausgerichtet ist. JDBC ist in seiner Funktion als universelle Datenbankschnittstelle vergleichbar mit z.B. ODBC unter Windows oder DBI unter Perl. Zu den Aufgaben von JDBC gehört es, Datenbankverbindungen aufzubauen und zu verwalten, SQL-Anfragen an die Datenbank weiterzuleiten und die Ergebnisse in eine für Java nutzbare Form umzuwandeln und dem Programm zur Verfügung zu stellen.. 5, 155, 158

Konzernmitglied Ein Konzernmitglied im Kontext der User Stories ist eine Firma oder eine juristische Person, welche in SYRIUS als Partner-BO abgelegt wird und in einem Konzern angegliedert ist. Durch diese Angliederung gelten spezielle Berechtigungen, zum Beispiel für den Konzernspezialist. 26, 27, 41, 42, 87, 88

Konzernspezialist Ein Konzernspezialist ist ein Mitarbeiter eines Versicherers, welcher sich mit den speziellen Gegebenheiten und Aufgaben in einem Konzern auskennt. Jeder Konzern wird durch einen Konzernspezialisten betreut. Der Konzernspezialist kann alle Firmen innerhalb des Konzerns verwalten. 22, 25–28, 41–44, 87, 88, 124, 151

Leistung Eine Leistung bezieht sich auf eine Versicherungsleistung. Der Kunde versichert bei der Versicherung verschiedene Leistungen, zum Beispiel Arbeitsunfall, Brandversicherung, etc. Tritt nun der versicherte Fall ein, kann der Kunde die Leistung bei der Versicherung beantragen. 16, 17, 22, 24, 29, 31, 33, 34, 41, 149–151, 154

Leistungserbringer Ein Leistungserbringer in SYRIUS ist eine Ausprägung des Partners. Sie erbringen Leistungen für welche die Versicherung eventuell aufkommen muss. Im Gesundheitswesen könnten dies zu Beispiel Ärzte, Spitäler, oder Therapeuten sein. Santésuisse stellt die Daten für Leistungserbringer allen Versicherungen zur Verfügung. 31, 41, 44, 152, 153

MetaBO *(SYRIUS)* Basisklasse für die Meta information in Syrius. Ein MetaBO bezeichnet den Typ eines BO, zum Beispiel ob das BO ein Partner oder ein Vertrag ist. 63, 82, 84, 109, 132, 135, 149

OASIS Organization for the Advancement of Structured Information Standards (OASIS) [39] ist ein Non-Profit-Konsortium, welches sich mit offenen Standards der

IT auseinandersetzt, solche erarbeitet und unterhält. Das Konsortium besteht aus über 5000 Mitgliedern aus 600 Organisationen. 46, 62, 81, 156

Objektschutz *(SYRIUS) Objektschutz* ist die Bezeichnung der aktuellen Berechtigungslösung in der Persistenzschicht von SYRIUS, welche den Schutz der BOs sicherstellt. 4–6, 12, 13, 106, 150

Objektschutzfacade *(SYRIUS)* Die *Objektschutzfacade* ist eine Klasse in der SYRIUS-Persistenzschicht. Sie ist die zentrale Stelle, welche von allen Komponenten die eine Abfrage mit der Objektschutz WHERE-Klausel erweitern wollen, aufgerufen wird. In der Implementation der *Objektschutzfacade* lässt sich der Objektschutz in SYRIUS ausschalten. In der Studienarbeit [28] wird die Rolle der *Objektschutzfacade* im Rahmen des Refactoring der Persistenzschicht detaillierter erläutert. 116

OE *(SYRIUS)* Organisationseinheiten (OEs) sind in SYRIUS eine Ausprägung des Partner-Objekts. Über den *PartnerTyp* kann ein Partner als OE deklariert werden. OEs werden meistens verwendet um die Struktur des Versicherers abzubilden. Aufgaben oder Postkörbe können einer OE zugewiesen werden. 16, 22, 24–26, 29, 41, 42, 86, 88, 101, 150, 152, 153, 156

Oracle SQL Developer Auszug Wikipedia: Der SQL Developer [44] ist eine Entwicklungsumgebung (Integrated Development Environment (IDE)) des Unternehmens Oracle für SQL, PL/SQL sowie ein Verwaltungswerkzeug für Datenbanken. Der SQL-Developer basiert auf Java und benötigt deshalb eine Java-Laufzeitumgebung, ist deshalb aber auch auf mehreren Plattformen lauffähig. Ursprünglich hieß der SQL Developer "Project Raptor". Die offizielle Unterstützung erstreckt sich dabei zwar nur auf Oracle-Datenbanken (ab Version 10g) und Microsoft Access, durch die auf Java basierende Architektur lassen sich aber auch JDBC-Treiber anderer Datenbanken wie beispielsweise MySQL einbinden, allerdings stehen dort diverse Oracle-spezifische Funktionen nicht zur Verfügung. 115

Partner *(SYRIUS)* Der Begriff Partner weist im Kontext von SYRIUS auf das BO *Partner* hin. Verschiedene Ausprägungen des Partners sind natürliche Personen, juristische Personen, Leistungserbringer oder OEs. Natürliche oder juristische Personen sind aus Sicht des Versicherers häufig Kunden, mit welchen Verträge abgeschlossen wurden. Leistungserbringer sind häufig Ärzte, Spitäler oder sonstige Rechnungssteller. OEs dienen meist zur Abbildung von Geschäftsstrukturen. 6, 16, 17, 19, 22–26, 29, 41, 42, 58, 63, 64, 78, 86, 92, 111, 149–152, 154

PL/SQL Auszug Wikipedia: Procedural Language (PL)/SQL (Procedural Language / Structured Query Language) ist eine proprietäre Programmiersprache der Firma Oracle. PL/SQL verbindet die Abfragesprache SQL mit einer prozeduralen Programmiersprache. Die Syntax ist stark an die Programmiersprache Ada angelehnt. Unterstützt werden Variablen, Bedingungen, Schleifen und Ausnahmebehandlungen. Ab Version acht des Oracle-Datenbanksystems halten auch objektorientierte

Merkmale Einzug. 152

Policy Policy ist ein Begriff aus dem ABAC-Konzept. Mehrere Policies können unter XACML zu einem *Policyset* zusammengefasst werden. Eine Policy bildet eine Gruppe von Zugriffsregeln ab. Als Beispiel kann in einer Policy definiert sein, welche Leute auf Mitarbeiterdaten zugreifen dürfen. 2, 3, 9, 10, 12, 15, 17, 19, 20, 24, 35–40, 42–49, 54, 55, 57–84, 86–89, 100–102, 106–110, 123, 124, 128–131, 134–137, 139, 140, 148, 154, 158, 160, 161

Postkorb *(SYRIUS)* Ein Postkorb ist ein Gefäss in welchem Aufgaben abgelegt sind. Jede Aufgabe ist zwingend einem Postkorb zugewiesen. Oft sind Postkörbe einer OE zugewiesen, damit die darin enthaltenen Aufgaben den richtigen Personen zugewiesen werden. Postkörbe können auch für alle Mitarbeiter einsehbar sein (öffentliche Postkörbe). 30, 41, 148, 150, 152

RBAC Role-Based Access Control (RBAC) ist ein Zugriffskontroll-Paradigma bei welchem Zugriffsrechte über Rollen, welche dem Benutzer und dem zu schützenden Objekt *statisch* zugewiesen werden, kontrolliert werden. 1, 9, 100, 156

RMI Wikipedia: Remote Method Invocation (RMI, deutsch etwa „Aufruf entfernter Methoden“), gelegentlich auch als Methodenfernaufruf bezeichnet, ist der Aufruf einer Methode eines entfernten Java-Objekts und realisiert die Java-eigene Art des Remote Procedure Call. 5, 156

Sachbearbeiter Ein Sachbearbeiter ist ein Angestellter des Versicherers. Ein Sachbearbeiter arbeitet meistens in einem Fachteam, welches für einen speziellen Fachbereich zuständig ist. Da die Fachbereiche sehr unterschiedlich sein können, erarbeitet sich ein Sachbearbeiter spezifisches Wissen zu seinem Fachbereich, welches ihn zu einem Spezialisten in diesem Bereich macht. 17, 19, 22–25, 28, 33, 150

Santésuisse Santésuisse ist die Branchenorganisation der Schweizer Krankenversicherungen. Über eine Tochterfirma *SASIS AG* stellt santésuisse allen Krankenversicherungen die Stammdaten der Leistungserbringer zur Verfügung. Mehr Informationen sind der Website von Santésuisse [53] zu entnehmen. 151

Schutzobjekt *(SYRIUS)* Der Begriff Schutzobjekt stammt von Adcubum und ist ein Teil der aktuellen Berechtigungslösung. Das Schutzobjekt ist das BO in einem Schutzpfad, welches die Berechtigungskonfiguration (Schutzdefinition) referenziert. Es definiert damit die Berechtigungen für die BOs innerhalb des aktuellen Schutzpfades. Die anderen BOs im Schutzpfad beziehen die Berechtigungskonfiguration über dieses Schutzobjekt. 6, 22, 23, 154

Schutzpfad *(SYRIUS)* Der Begriff *Schutzpfad* stammt aus der aktuellen Berechtigungslösung von Adcubum. Nicht jedes BO wird direkt durch eine Konfiguration von Rechten (Schutzdefinition) geschützt, sondern indirekt über ein sogenanntes Schut-

zobjekt. Das Schutzobjekt referenziert die Berechtigungskonfiguration (Schutzdefinition) und legt damit die Berechtigungen für alle Objekte im Schutzpfad fest. Zum Beispiel wird ein Vertrag über die Schutzdefinition des Partners geschützt. In diesem Fall ist der Vertrag Teil des Schutzpfades und der Partner das Schutzobjekt. Da die BOs das Schutzobjekt auch indirekt über diverse andere BOs referenzieren können, wird bei Adcubum von einem Schutzpfad gesprochen. Dabei muss sichergestellt werden, dass das Schutzobjekt für ein BO eindeutig ermittelt werden kann. 6, 22, 23, 42, 44, 64, 113, 153, 158

UTC *(SYRIUS)*UTC steht für 'Ultra-Thin Client' und bezeichnet eine Präsentationskomponente in Adcubum SYRIUS. 5, 28, 113, 157

Vertriebspartner Vertriebspartner arbeiten für einen Versicherer. Sie verkaufen Verträge und Produkte des Versicherers an die Kunden. Allerdings sollen Vertriebspartner in SYRIUS nicht mehr sehen, als sie für ihre Aufgabe benötigen, da sie oft nicht bei der Versicherung arbeiten. 28, 29, 31, 32, 41

Vorbehalt Vorbehalte werden von Versicherern beim Vertragsabschluss evaluiert. Aufgrund eines Vorbehalts können zum Beispiel gewisse Leistungen bei der Krankenversicherung entfallen, basierend auf der Krankengeschichte des Kunden. Vorbehalte zählen zu den SPIs, und sind besonders sensibel, da daraus auf den Gesundheitszustand einer Person geschlossen werden kann. 23, 33, 41, 90

XACML eXtensible Access Control Markup Language (XACML) ist ein XML-Schema, das die Darstellung und Verarbeitung von Autorisierungs-Policies standardisiert. XACML ist die bekannteste Standard-Implementation von ABAC. 2, 3, 40, 45–51, 54, 55, 57–63, 67–70, 74–77, 80–82, 100, 106–110, 123, 124, 129–132, 134, 137, 139, 140, 148, 153, 158, 160, 161

XML Auszug Wikipedia: Die Erweiterbare Auszeichnungssprache (englisch Extensible Markup Language), abgekürzt XML, ist eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien. XML wird auch für den plattform- und implementationsunabhängigen Austausch von Daten zwischen Computersystemen eingesetzt, insbesondere über das Internet. XML wurde vom World Wide Web Consortium (W3C) am 10. Februar 1998 veröffentlicht. 45, 46, 48, 61, 62, 73, 75, 132

Abkürzungsverzeichnis

ABAC Attribute-Based Access Control, siehe Glossar: ABAC. 1–3, 9–12, 15, 20, 24, 35, 36, 38, 40, 44–46, 48, 50, 51, 54, 55, 61, 84, 86, 94, 100–102, 106–108, 110, 123, 140, 148, 153, 154, 158

ALFA Abbreviated Language For Authorization, siehe Glossar: ALFA. 46–48, 81–83, 108, 139, 148, 160

AOP Aspektorientierte Programmierung. 102

BO Business Object, siehe Glossar: BO. 5, 6, 12, 17, 18, 23, 29, 41–43, 46, 52–55, 57, 63, 64, 68, 69, 71, 77, 84–90, 92, 101, 102, 109, 112, 113, 122, 130, 143, 149, 151–154, 160

CPU Central Processing Unit. 91

DDD Domain-Driven Design. 7

DSG Datenschutzgesetz. 17, 28, 32, 33, 149

EJB Enterprise JavaBean. Siehe Glossar: EJB. 5, 158

GPL3 GNU General Public License v3.0. 37

GUI Graphical User Interface. 12, 15, 55

HOPP Half Object plus Protocol. 5

HR Human Resources. 24, 25

IDE Integrated Development Environment. 152

IIOP Internet Inter-Orb Protocol. Siehe Glossar: IIOP. 5

JDBC Java Database Connectivity. Siehe Glossar: JDBC. 5, 158

JSON JavaScript Object Notation. 45, 46, 53

LDAP Lightweight Directory Access Protocol. 8, 16, 17, 54, 81, 84, 85, 89, 95–97

NFA nichtfunktionale Anforderung. 14, 40, 47–49, 51, 81, 90, 104–111, 123, 124, 127, 137, 140

OASIS Organization for the Advancement of Structured Information Standards, siehe Glossar: OASIS. 46, 62, 81

OE Organisationseinheit, siehe Glossar: OE. 16, 22, 24–26, 29, 41, 42, 86, 88, 101, 150, 152, 153

PAP Policy Administration Point. 10, 50, 51, 55

PDP Policy Decision Point. 2, 10, 24, 40, 45–51, 54–57, 59, 60, 62, 68, 69, 72–76, 79, 80, 94, 100, 109, 148, 161

PEP Policy Enforcement Point. 10, 50, 51, 60, 62, 68, 94

PIP Policy Information Point. 10–13, 15, 42, 45, 54–60, 64, 81, 84–88, 90, 94, 100, 101, 110

PL Procedural Language. 152

RAM Random-access Memory. 91

RBAC Role-Based Access Control, siehe Glossar: RBAC. 1, 9, 100

REST Representational State Transfer. 45, 46, 91

RMI Remote Method Invocation. Siehe Glossar: RMI. 5

RTT Round Trip Time. 92

SLA Service Level Agreement. 38

SPI Sensitive Personal Information. 28, 149, 154

SQL Structured Query Language. 5, 6, 111, 113–117, 119–121, 152

UC Use Case. 21

UI User Interface. 5, 150

UML Unified Modelling Language. 14, 124, 149, 158

US User Story. 21

UTC Ultra-Thin Client. Siehe Glossar: UTC. 5, 28, 113

VIP Very important Person. 19, 24, 41–44, 58, 59, 63–65, 67, 71, 74, 76–78, 82, 128, 158, 160

Abbildungsverzeichnis

Auflistungsverzeichnis

Tabellenverzeichnis