Architectural Refactoring der Data Access Security

Stefan Kapferer

STUDIENARBEIT

Abteilung Informatik
Hochschule für Technik Rapperswil

Betreuer: Prof. Dr. Olaf Zimmermann

Industriepartner: Adcubum AG

Herbstsemester 2016

Inhaltsverzeichnis

Abstract

In adcubum SYRIUS®, einer geschichteten ERP-Lösung für Versicherungen, werden Zugriffsberechtigungen in der Datenbank gespeichert. Die Autorisierung der Daten wird bei jedem Datenzugriff in der Persistenzschicht durchgeführt. Solange nur Daten berechtigt werden müssen, welche in der eigenen Datenbank persistiert sind, ist diese Lösung genügend.

Strategisches Domain-Driven Design (DDD) ist eine Methodik, die zur Identifikation von Modulen oder Microservices dienen kann. In dieser Studienarbeit wurde ein Prototyp entwickelt, in welchem die Autorisierung einen eigenen Bounded Context nach DDD bildet. Dieser wird als eigener Microservice betrieben und von den Kerndomänen über eine Remote-Schnittstelle angesprochen. Dafür wurde im Rahmen dieser Studienarbeit eine Autorisierungsschnittstelle auf der Basis von RESTful HTTP entwickelt. Ausserdem wurde in SYRIUS ein Redesign in der Persistenzschicht durchgeführt, um die alte Berechtigungslösung zu extrahieren und die neue Autorisierungsschnittstelle aufzurufen. Desweiteren wurde das «Attribute-based Access Control» (ABAC) Paradigma analysiert, welches die Wartung und Konfiguration der heutigen RBAC-Lösung vereinfachen soll. Mittels einer Mock-Implementation des Autorisierungssystems und dessen Schnittstelle wurde schliesslich die Umsetzbarkeit des Konzepts einer externen Autorisierungslösung überprüft.

Für den Industriepartner Adcubum hat diese Studienarbeit das Wissen erarbeitet, an welchen Stellen in SYRIUS die Autorisierungsschnittstelle aufgerufen werden muss. Ausserdem dokumentiert die RESTful HTTP-Schnittstellendefinition, welche Daten dem Autorisierungssystem übergeben werden müssen. Aus den Überlegungen, wie das Zugriffskontrollmodell von RBAC auf ABAC umzustellen ist, wurde ein templatekonformes, wiederverwendbares Architectural Refactoring abgeleitet.

1. Management Summary

1.1. Ausgangslage

In adcubum SYRIUS® (nachfolgend SYRIUS genannt) sind die Zugriffsberechtigungen in der Datenbank abgelegt. Bei jedem Datenzugriff wird die WHERE-Klausel des SQL-Statements um die entsprechende Berechtigungsüberprüfung ergänzt. Sobald Datenteilbestände ausserhalb der eigenen Datenbank gehalten werden, wird dieser Ansatz jedoch kompliziert. Schon Heute indexiert SYRIUS gewisse Datenteilbestände in Elasticsearch. Um die indexierten Daten, welche aus Elasticsearch gelesen werden, autorisieren zu können, muss ein Service-Aufruf in SYRIUS durchgeführt werden. Dies ist nicht nur unpraktisch, sondern schlägt sich auch in der Performance nieder.

Im Rahmen der strategischen Wartung von SYRIUS und der Weiterentwicklungen im Bereich Frontoffice werden die Schnitte der einzelnen Applikationskomponenten in Zukunft angepasst. Aus diesem Grund werden in Zukunft weitere eigenständige Applikationskomponenten (Microservices) entstehen werden. Die Daten, welche von diesen Komponenten verwaltet werden, müssen ebenfalls autorisiert werden können.

Diese Überlegungen führen dazu, das aktuelle Berechtigungssystem durch ein externes Autorisierungssystem abzulösen, welches dann als eigener Microservice zur Verfügung steht. Die verschiedenen Applikationskomponenten können die Zugriffe auf die Business Objekte (BO's) und deren Attribute dann über diesen Microservice autorisieren.

1.2. Vorgehen

Im Rahmen dieser Studienarbeit wurde die bestehende Berechtigungslösung von SYRIUS analysiert und ermittelt, an welchen Stellen in der Persistenzschicht ein neues Autorisierungssystem aufgerufen werden muss. Es wurde untersucht wie eine neue Autorisierungsschnittstelle aussehen könnte und ein Prototyp entwickelt. Dabei wurde der Schwerpunkt auf das Schnittstellen-Design und den Aufruf des Prototypen aus SYRIUS gelegt. Bei der Entwicklung des Prototypen wurde das Autorisierungssystem als Blackbox betrachtet bzw. eine Mock-Implementation erstellt.

Beim Design der Schnittstelle und des Prototypen war zu berücksichtigen, dass das neue Berechtigungssystem nicht mehr Rollen-basiert (RBAC), sondern Attribut-basiert (ABAC) sein soll. Dies deshalb, weil das ABAC-Paradigma mehr Flexibilität bietet und das heutige Rollen-basierte System anspruchsvoll in der Parametrierung und Wartung ist.

Ein weiteres Ziel dieser Studienarbeit bestand in der Aufbereitung des gesammelten Architekturwissens in Form von Patterns und Architectural Refactorings.

1.3. Ergebnisse

In der Studienarbeit wurde ein Prototyp für drei ausgewählte Use Cases in SYRIUS entwickelt, welche den Lese- und Schreib-Schutz von BO's und deren Attribute berücksichtigen. Die Use Cases wurden mit dem Industriepartner ausgewählt und decken einen grossen Anteil der Funktionalität der heutigen Berechtigungslösung ab. Zu diesem Prototyp gehören die Schnittstellendefinition der RESTful HTTP Schnittstelle, die Mock-Implementation des Autorisierungssystems, sowie eine Java-Library (Client) welche in SYRIUS verwendet wird, um das Autorisierungssystem aufzurufen. Ein weiteres Ergebnis ist das Redesign innerhalb von SYRIUS, damit der Prototyp aus der Persistenzschicht aufgerufen werden kann.

1.4. Ausblick

In einem nächsten Schritt soll untersucht werden, wie das Autorisierungssystem implementiert werden kann. Dabei stellt sich vor allem die Frage, welche Daten das Autorisierungssystem benötigt und ob die Berechtigungskonfiguration in das neue ABAC-System überführt werden kann. Ausserdem soll die Konfiguration der Berechtigungen für den Kunden generell einfacher werden. Durch die Analyse dieser Problemstellungen soll die Management-Entscheidung «Make or Buy» vorbereitet werden, denn das Autorisierungssystem muss nicht zwingend von Adcubum selbst implementiert werden.

2. Kontext

2.1. Fachliche Domäne

Industriepartner dieser Studienarbeit ist die Adcubum AG. 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. Für den in dieser Studienarbeit entwickelten Prototypen eines neues Autorisierungssystems, wurde sowohl ein Redesign im bestehenden Source-Code von SYRIUS vorgenommen, als auch neue Komponenten entwickelt.

2.2. Aktuelle Architektur

Abbildung 2.1 gibt einen Überlick über die Architektur von SYRIUS. Das Backend ist eine geschichtete JEE-Applikation mit einer Oracle-Datenbank zur Persistenz.

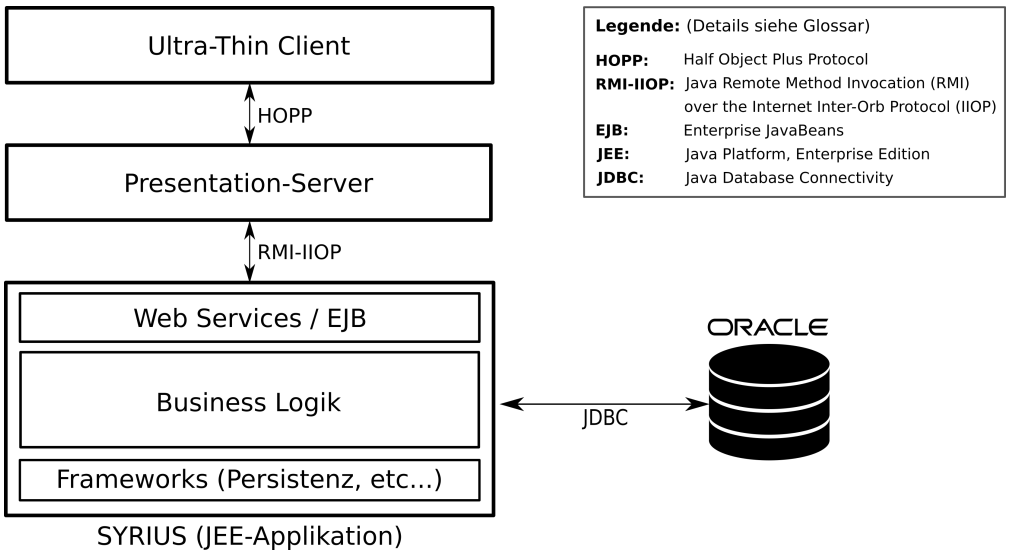

Abbildung 2.1.: SYRIUS Architektur

Der Presentation Server kommuniziert via *Remote Method Invocation (RMI) over Internet Inter-Orb Protocol (IIOP)* mit dem SYRIUS Application Server und bereitet das User Interface (UI) auf. Der Ultra-Thin Client (UTC) zeigt das UI lediglich an und implementiert keinerlei Logik. Die Kommunikation zwischen UTC und Presentation-Server funktioniert über das Half Object plus Protocol (HOPP) [17].

In der Persistenzschicht von SYRIUS werden die BO's, zum Beispiel Partner, Verträge oder Produkte, auf Datenbanktabellen abgebildet. Die Berechtigungslösung in der Persistenzschicht, welche den Schutz der BO's sicherstellt, wird Objektschutz genannt. Mit dem aktuellen Objektschutz lassen sich sämtliche BO's über *Schutzobjekte* schützen. Der Begriff Schutzobjekt stammt vom Industriepartner und bezeichnet dasjenige BO innerhalb eines *Schutzpfades*, welches die Berechtigungskonfiguration (Schutzdefinition) referenziert. BO's können also geschützt werden, indem sie selber das Schutzobjekt sind, oder dieses indirekt über den Schutzpfad referenzieren. Zum Beispiel kann ein *Vertrag* über das Schutzobjekt *Partner* geschützt werden. Da dies indirekt über mehrere Referenzen möglich ist, spricht man beim Industriepartner von einem Schutzpfad.

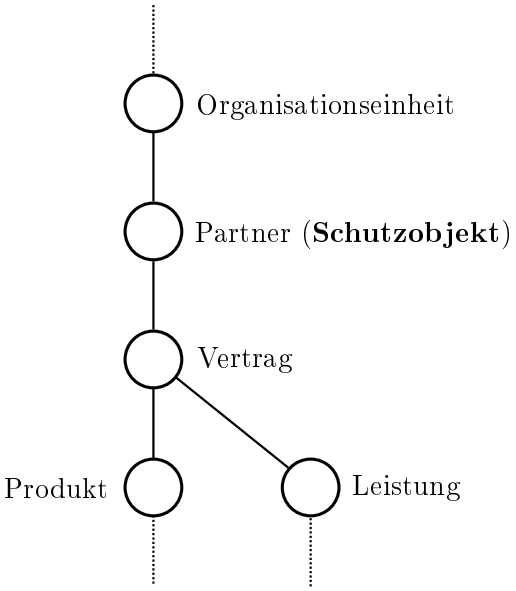

Abbildung 2.2.: Schutzpfad

In der Abbildung 2.2 ist ein solcher Schutzpfad beispielhaft dargestellt. Über das darin enthaltene Schutzobjekt (Partner) wird definiert, welche Benutzer die Objekte innerhalb dieses Schutzpfades sehen dürfen.

Implementiert ist dieser Objektschutz über eine Erweiterung der WHERE-Klausel, welche sicherstellt, dass keine Daten aus der Datenbank gelesen werden, auf welche der Benutzer keinen Zugriff hat. Da diese Lösung viele Sub-Queries generiert, hat sie auch

eine Auswirkung auf die Performance der einzelnen Datenbankabfragen.

```
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 2.1: Objektschutz: Erweiterung der WHERE-Klausel

Auflistung 2.1 zeigt ein vereinfachtes Beispiel einer solchen WHERE-Klausel. In diesem Beispiel wird ein Vertrag geladen und der Partner, welcher über die Fremdschlüsselspalte *itsPartner* referenziert wird, ist das Schutzobjekt. Der Partner referenziert über *itsPartnerSchutz* die Schutzdefinition (Berechtigungskonfiguration). Ist diese Schutzdefinition nicht eine derjenigen, auf welche der Benutzer Zugriff hat, wird der Vertrag nicht aus der Datenbank geladen.

2.3. Aufgabenstellung

Der folgende Abschnitt 2.4 fasst das Ziel dieser Studienarbeit zusammen. Die vollständige Aufgabenstellung der Studienarbeit befindet sich im Anhang E.

2.4. Ziel dieser Studienarbeit

Ziel ist es, die aktuelle Berechtigungslösung durch ein externes Autorisierungssystem abzulösen. Dabei soll der jetzige Objektschutz aus der Persistenzschicht extrahiert und stattdessen die Autorisierung in einem eigenen Microservice durchgeführt werden. Dieser kann von SYRIUS und anderen Systemen, wie Elasticsearch [7], aufgerufen werden. Elasticsearch [7] wird verwendet, um Daten aus SYRIUS für eine schnellere Durchsuchbarkeit zu indexieren. Um Suchresultate autorisieren zu können, soll in Zukunft kein Service-Aufruf in SYRIUS mehr notwendig sein.

Durch die Anstrengungen der Adcubum AG im Bereich Frontoffice, wird die Notwendigkeit eines solchen Autorisierungssystems ebenfalls immer grösser. Über diese Frontoffice

Applikationen werden in Zukunft nicht nur Sachbearbeiter auf Kundendaten zugreifen, sondern auch der Endkunde selbst. Auch diese Zugriffe erfordern eine Autorisierung. Wie aus Abbildung 2.3 hervorgeht, werden diese externen Benutzer jedoch in einem separaten *Lightweight Directory Access Protocol (LDAP)* -Verzeichnis verwaltet.

Das neue Autorisierungssystem wird also in eine Architektur-Landschaft eingebettet werden, welche aus verschiedensten Applikationen besteht. Entsprechend wird das System über Schnittstellen mit anderen Systemen kommunizieren. Vorerst wird sich dies aber auf eine Schnittstelle zu adcubum SYRIUS und einem LDAP-Verzeichnis beschränken.

Abbildung 2.3 zeigt die mittelfristige Systemlandschaft, in welches das neue Autorisierungssystem eingebettet werden soll.

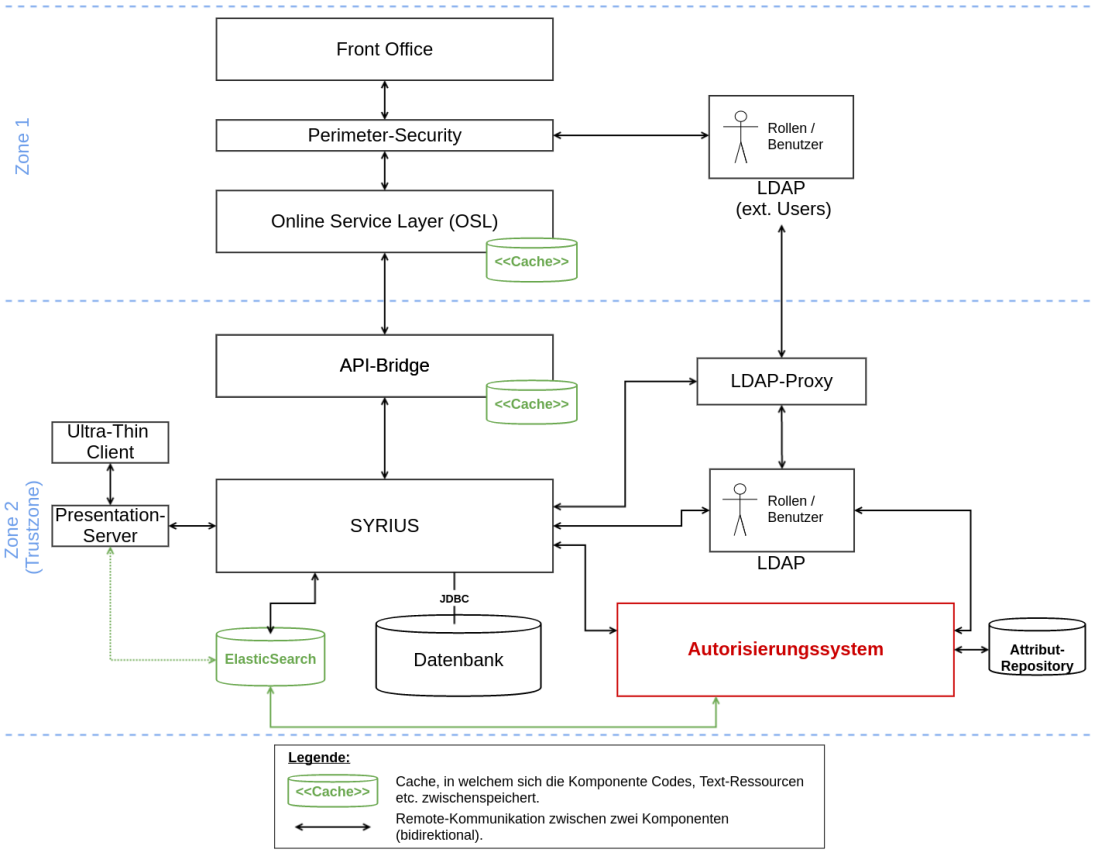

Abbildung 2.3.: Systemübersicht (Kontext)

Aus dem LDAP-Verzeichnis soll das Autorisierungssystem die Informationen zu einem Benutzer beziehen können. Da das System auf dem ABAC-Paradigma beruhen soll, wird

es nicht nur Attribute des Benutzers, sondern auch Attribute der Business-Objekte benötigen, um Autorisierungsentscheidungen treffen zu können. Diese Attribute muss sich das System entweder aus SYRIUS beziehen oder aus Performance-Gründen lokal persistieren, was in Abbildung 2.3 mit dem *Attribut-Repository* angedeutet wird. Das ABAC-Paradigma wird im Kapitel 3 im Detail erläutert.

2.5. Domain-Driven Design Kontext

Strategisches Domain-Driven Design DDD ist eine Methodik zur Modellierung und Organisation komplexer Businesslogik, welche von Eric Evans [9] geprägt wurde. Das Erarbeiten sogenannter *Bounded Context* legt die Grenzen der einzelnen Kontexte fest und sorgt dafür, dass die Kommunikation zwischen den Kontexten über wohldefinierte Schnittstellen läuft. Die Methodik eignet sich daher auch gut zur Identifikation von einzelnen Modulen oder Microservices [8]. Auch Adcubum arbeitet im Rahmen der strategischen Wartung mit DDD, um die einzelnen Module und deren Grenzen festzulegen. Das neue Autorisierungssystem soll in diesem DDD-Umfeld einen eigenen *Bounded Context* bilden.

Das Domänenmodell von Adcubum SYRIUS ist in die drei Hauptbereiche *Kerndomänen*, *Querschnittsdomänen* und *IT-Unterstützungsdomänen* eingeteilt.

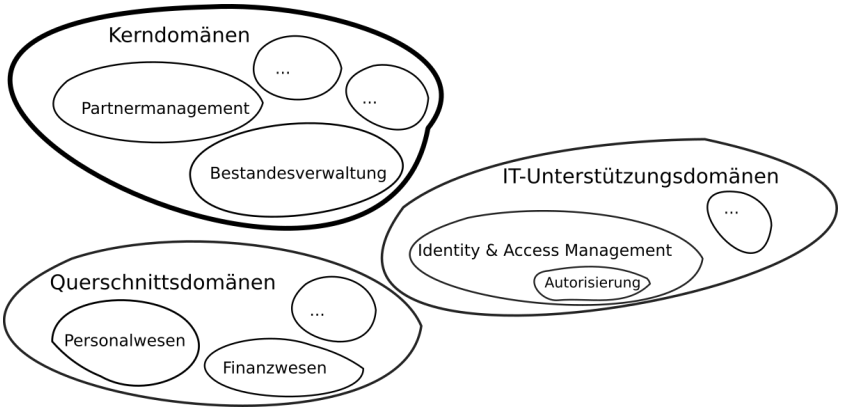

Abbildung 2.4.: Syrius Domänenmodell

In den Kerndomänen ist die fachliche Business-Logik der Versicherungswelt abgebildet. Die Querschnitts- und IT-Unterstützungsdomänen bieten unterstützende Funktionalitäten und sind sogenannte *Supporting Domains*. Abbildung 2.4 zeigt diese Unterteilung in einer vereinfachten Art, ohne alle Domänen explizit zu beinhalten.

Ein essentielles Konzept in DDD sind die *Context Maps*. Werden die Linien zwischen den einzelnen «Bounded Contexts» nicht klar definiert und die Grenzen von den Entwick-

lungsteams nicht verstanden, entsteht mit der Zeit das, was in der Literatur «Big Ball of Mud» genannt wird. Eine Context Map ist ein effektives Werkzeug um die Grenzen der einzelnen Domänen zu kommunizieren. Ausserdem illustriert sie die technischen und organisatorischen Abhängigkeiten zwischen den einzelnen «Bounded Context's».

Die «Context Map», welche sich stetig mit den Veränderungen in der Software entwickeln soll, bietet einem Entwicklungsteam eine ganzheitliche Sicht auf das System. Diese Studienarbeit befasst sich mit der Domäne der *Autorisierung*. Die Autorisierung ist den IT-Unterstützungsdomänen, konkret der Domäne *Identity & Access Management (IAM)*, zuzuordnen. Diese ist in Abbildung 2.4 bereits ersichtlich.

Der «Bounded Context» der Autorisierung soll nun mittels einer Context Map genauer beleuchtet werden. Die nachfolgende Context Map und deren Beschreibung hält sich an einige Fachbegriffe aus der Domain-Driven Design Literatur [9, 45, 46], welche an dieser Stelle nicht erklärt werden.

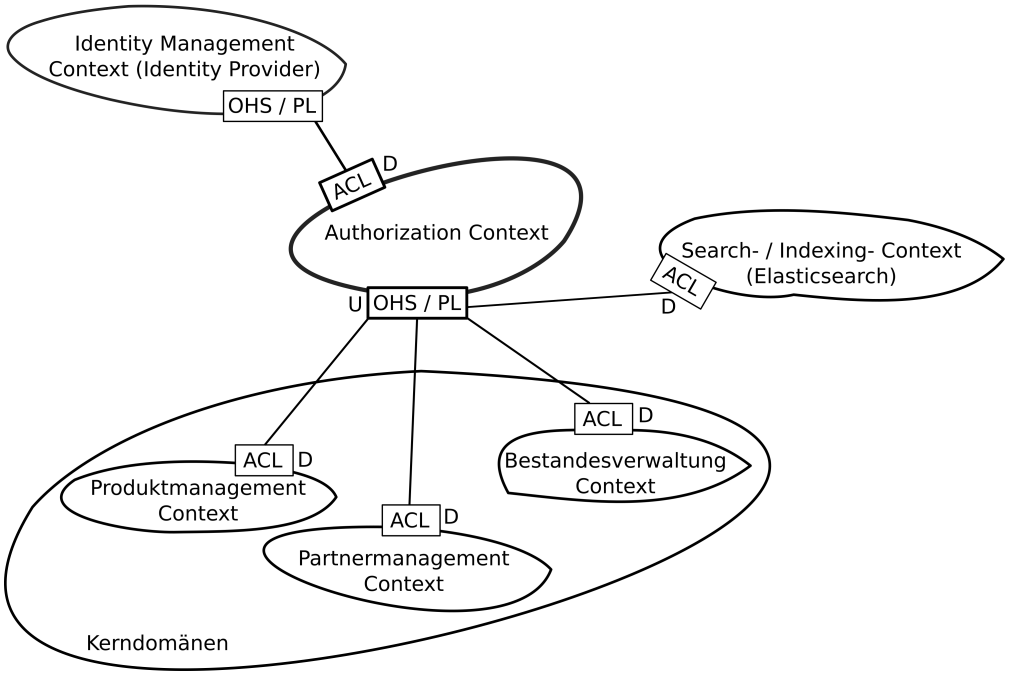

Abbildung 2.5.: DDD Context Map

Der *Autorisierungscontext* hat eine Beziehung zum *Identity Management Context*, da zur Autorisierung die Identität des Benutzers entscheidend ist. Der *Identity Management Context* wird durch ein externes LDAP-Directory implementiert und ist daher der «Upstream» in der «Upstream/Downstream»-Beziehung.

Wie Abbildung 2.5 ausserdem zeigt, exponiert der *Autorisierungskontext* einen «Open Host Service» (OHS) und die dazugehörige «Published Language». Dieses Pattern wurde in dieser Studienarbeit durch eine RESTful HTTP Schnittstelle realisiert. Dabei repräsentiert das API-Modell, welches in JSON dargestellt wird, die «Published Language». Zu dieser «Published Language» gehören die Objekte wie der *UserIdentifier* oder *BOIdentifier*, welche nötig sind um eine Autorisierung durchzuführen. Diese Objekte werden später im Kapitel 5 bei der Dokumentation der Schnittstellendefinition genauer erläutert.

Dieser «Open Host Service» wird von verschiedenen Kerndomänen in SYRIUS verwendet, welche die Schnittstelle ihrerseits direkt oder über einen «Anti-Corruption Layer» (ACL) aufrufen. Ein ACL wird von einem *Downstream*-Context häufig eingesetzt, wenn die *Published Language* des *Upstream* (OHS) nicht zum eigenen Modell passt und eine Konvertierung nötig ist. Evans beschreibt in seinem Buch [9], wie ein ACL mit den Patterns FACADE und ADAPTER [12] implementiert werden kann. Ob die verschiedenen SYRIUS-Kerndomänen alle einen ACL implementieren, um die Autorisierungsschnittstelle aufzurufen, bleibt letztendlich den einzelnen Teams überlassen.

Desweiteren wird der *Autorisierungskontext*, wie in Abschnitt 2.4 bereits erwähnt, von der Such- und Indexierungs-Lösung Elasticsearch [7] verwendet.

Nachdem dieses Kapitel den Kontext der Studienarbeit verdeutlicht hat, geht das nächste Kapital auf die Analyse des Themenbereichs *Autorisierung* und der bestehenden Berechtigungslösung ein.

3. Analyse

Dieses Kapitel führt in die Autorisierungs- und Zugriffskontrollmodelle ein und wirft dabei einen Blick auf verwandte Forschungsarbeiten dieses Themenbereichs. Ausserdem wird das bestehende Modell aus SYRIUS vorgestellt.

3.1. Zugriffskontrollmodelle

Das *Confidentiality, Integrity, Availability (CIA)* Prinzip (Vertraulichkeit, Integrität, Verfügbarkeit) beschreibt in der Informationssicherheit die drei wichtigsten Sicherheitsanforderungen an ein Computersystem. Im Kontext dieser Arbeit ist das Prinzip der Vertraulichkeit entscheidend, da es für eine Unternehmung von grosser Bedeutung ist, dass nur autorisierte Benutzer Zugriff auf die Daten erhalten. Ein Unternehmen ist rechtlich verpflichtet, sensible Kundendaten vor nicht autorisierten Zugriffen zu schützen. Es existieren verschiedene Zugriffskontrollmodelle, welche auf einem hohen Abstraktionslevel Lösungen für diese Sicherheitsanforderung vorschlagen [41]. Diese Modelle versuchen die Frage zu beantworten, wie der Zugriff auf die schützenswerten Objekte modelliert und implementiert werden sollen. Speziell in grossen verteilten Systemen ist dies eine Herausforderung.

Eines der bekanntesten Modelle ist das *Role-Based Access Control (RBAC)* Modell. Dieses Modell schlägt vor, dem Benutzer eine beliebige Anzahl von *Rollen* zuzuweisen.

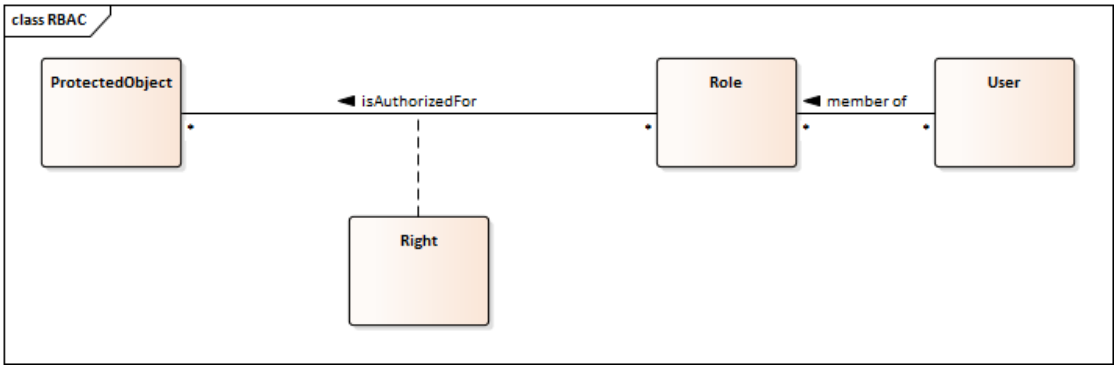

Abbildung 3.1.: Role-based Access Control [41] (UML Klassendiagramm)

Den einzelnen Rollen werden dann Rechte zugewiesen, welche definieren, auf welche *geschützen Objekte* der Zugriff erlaubt wird.

RBAC hat sich in den 1990er Jahren verbreitet und ist bis heute eines der meist verwendeten Modelle in kommerziellen Systemen [40]. Die grösste Herausforderung dieses Modells ist das sorgfältige Entwickeln der Rollen. Ist die Granularität der Rollen nicht adäquat, endet man schnell in einer *Role Explosion* [21, 33, 40], bei welcher fast jedem Anwender explizit Rollen zugewiesen werden müssen. In diesem Modell entsteht tendenziell eine sehr *statische* Verbindung zwischen dem Benutzer und den Objekten.

In den vergangenen Jahren hat sich das Modell *Attribute-based Access Control (ABAC)* als Alternative zu RBAC entwickelt. Wie der Name ABAC bereits impliziert, werden in diesem Modell die Berechtigungsregeln mit *Attributen*, anstatt mit *Rollen* definiert. Ein Attribut ist dabei immer ein *Name-/Value-Pair*. Dieses Modell bringt gegenüber RBAC eine grössere Flexibilität mit sich. Es können Attribute von beliebigen Objekten, wie zum Beispiel dem *zu schützenden Objekt*, dem *Benutzer* oder auch des *Systems*, verwendet werden.

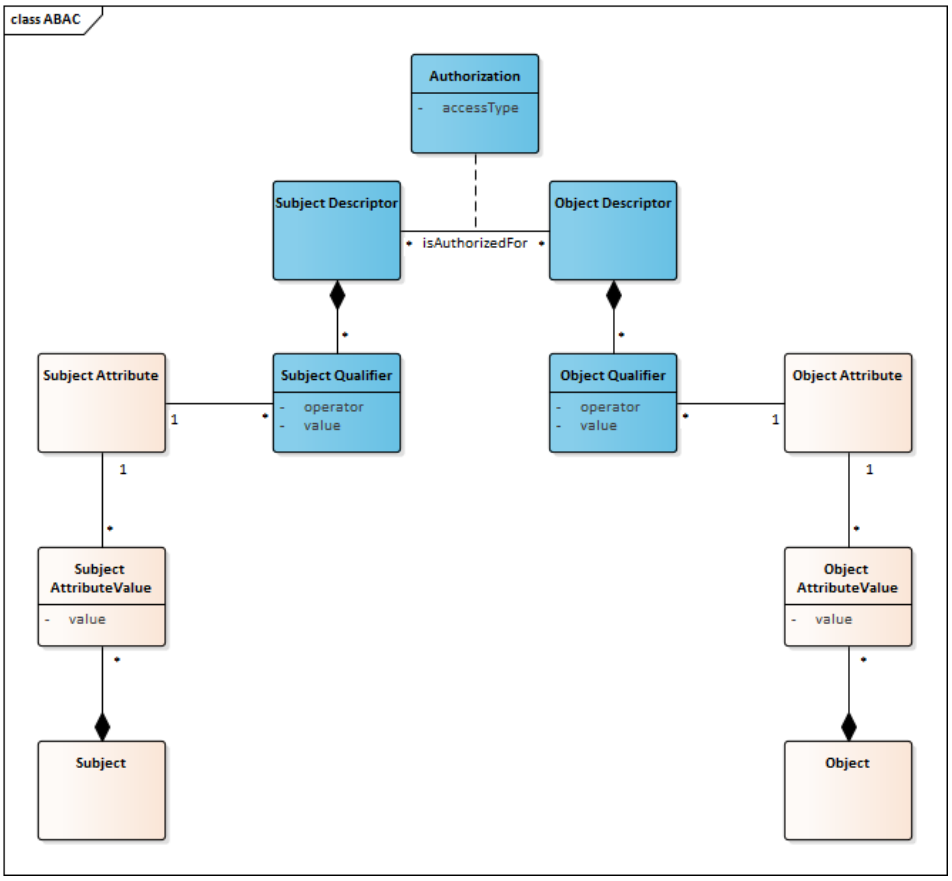

Abbildung 3.2.: Attribute-based Access Control [38] (UML Klassendiagramm)

In einem verteilten System mit verschiedenen «Bounded Context» oder «Microservices», können die Attribute ausserdem von verschiedensten Applikationkomponenten stammen. Da ABAC die Autorisierungsentscheidung aufgrund von Attributen der vorhandenen Objekte trifft, lässt dieses Modell im Vergleich zu RBAC eine *lose Kopplung* zwischen den Benutzern und den Objekten zu. Es muss keine *statische* Verbindung zwischen den Benutzern, deren Rollen und den Objekten hergestellt werden.

Abbildung 3.2 zeigt, dass die Autorisierung mit dem ABAC-Modell, über Subjekt- und Objektdeskriptoren definiert werden. Diese enthalten eine oder mehrere Attributbedingungen (Subject- und ObjectQualifier), wie zum Beispiel «Alter $>$ 18» oder «PLZ ist gleich 9000» [38]. Auf diese Weise lassen sich flexibel Autorisierungsregeln entwickeln, ohne das Subjekt oder das Objekt statisch zu referenzieren.

Unter dem *Subjekt* wird der Benutzer oder das zugreifende System verstanden. Das *Objekt* ist das zu schützende Objekt oder die zu schützende Ressource.

3.2. Heutige Lösung in SYRIUS

Die aktuelle Lösung in SYRIUS kann im Wesentlichen auf ein RBAC-Modell reduziert werden, auch wenn die Implementation etwas komplexer ist.

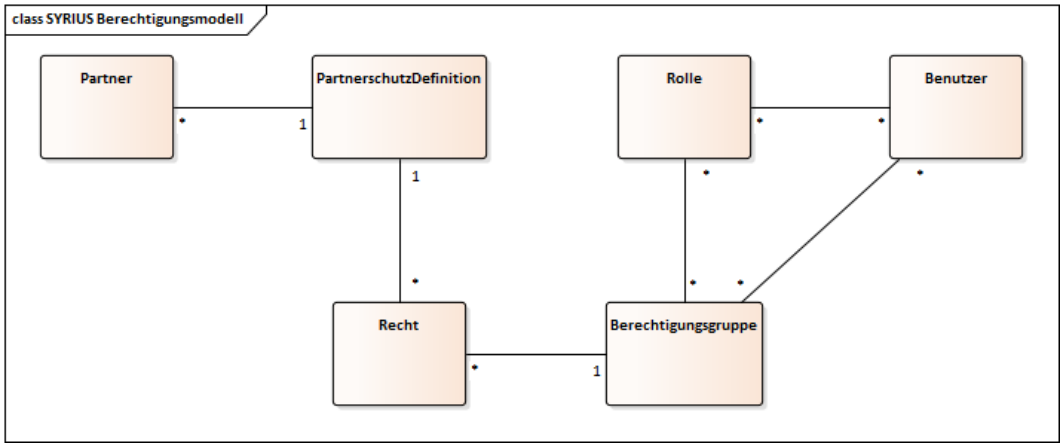

Abbildung 3.3.: Vereinfachtes Modell für Partnerschutz (UML Klassendiagramm)

SYRIUS lässt durch die vielen Konfigurationsmöglichkeiten verschiedene einfachere oder komplexere Modelle zu. Wie bei RBAC gibt es aber immer Benutzer, Rollen und die statische Verbindung zu dem zu schützenden Objekt. Durch die verschiedenen Konfigurationsmöglichkeiten lässt sich steuern, wie komplex die Verbindung zwischen diesen Objekten ist.

In Abbildung 3.3 wird eine mögliche Variante dargestellt. In diesem Fall ist der Partner das Schutzobjekt. Das Schutzobjekt hat immer eine Referenz auf eine *Schutzdefinition*, in diesem Fall die *Partnerschutzdefinition*. Über *Rechte* wird die Verbindung zwischen den Schutzdefinitionen und den Berechtigungsgruppen hergestellt, welchen dann die Rollen und Benutzer zugeordnet werden.

Das Modell aus Abbildung 3.3 wird in der relationalen Datenbank von SYRIUS gespeichert. Im Datenmodell entsteht auf diese Weise wieder eine *statische* Verbindung zwischen den BO's und den Benutzern, was auch einer Verbindung zwischen den Laufdaten (Partner, Verträge, etc.) und Konfigurationsobjekten entspricht. Die Autorisierung der Daten wird, wie in Kapitel 2 bereits erklärt, in der Persistenzschicht über die Erweiterung der SQL WHERE-Klausel gelöst.

Diese Lösung bringt einige Herausforderungen mit sich. Einerseits ist die Konfiguration der Berechtigungen sehr komplex und damit aufwendig. Andererseits ist die Kopplung der Berechtigungen und der Fachobjekte in der SYRIUS-Persistenz gross, was eine Autorisierung von Daten, welche nicht in SYRIUS gespeichert sind, umständlich macht. Die in Elasticsearch [7] indexierten Daten können zum Beispiel nicht direkt autorisiert werden. Es ist in jedem Fall ein Aufruf eines SYRIUS-Service notwendig. Ausserdem erhöht diese Lösung die Komplexität der Persistenzschicht von SYRIUS und verschlechtert damit deren Wartbarkeit. Ein weiteres Problem ist, dass die Erweiterung der WHERE-Klausel die Performance von Datenabfragen wesentlich verschlechtert.

3.3. Umfang dieser Studienarbeit

Aus den in den vorigen Abschnitten erwähnten Gründen ist ein Architekturrefactoring, welches ein flexibleres Zugriffskontrollmodell einführt, ein wichtiges Ziel für den Industriepartner Adcubum. Das neue Autorisierungssystem soll auf dem ABAC-Paradigma basieren.

Diese Studienarbeit konzentriert sich auf die Entwicklung einer Schnittstellendefinition, über welche SYRIUS in Zukunft ein ABAC-basiertes Autorisierungssystem aufrufen soll. Es soll ein Prototyp erstellt werden, in welchem aus SYRIUS diese Schnittstelle zur Autorisierung aufgerufen wird. Das Ziel dieser Arbeit ist nicht die Implementation eines Autorisierungssystems. Dass das neue System auf dem ABAC-Paradigma basieren wird, ist aber entscheidend für die Schnittstelle, da es einen Einfluss darauf hat, welche Daten dem Autorisierungssystem übergeben werden müssen.

3.4. Standards und verwandte Forschungsarbeiten

3.4.1. XACML

XACML ist ein primär auf dem ABAC-Paradigma basierender Standard [32]. Es handelt sich um eine XML-basierte, deklarative Sprache um Attribut-basierte Autorisierungsregeln zu definieren. Ausserdem enthält der Standard ein Verarbeitungsmodell welches beschreibt, wie Autorisierungsanfragen anhand der definierten Regeln evaluiert werden sollen. Diverse Open-Source und kommerzielle Produkte, welche auf dem ABAC-Paradigma basieren, wie zum Beispiel *Axiomatics Policy Server* [3], *SunXACML* [42] oder *AuthZForce* [2], implementieren diesen Standard.

Der XACML Standard [32] hat im Rahmen dieser Studienarbeit wertvolle Hinweise geliefert wie, ABAC funktioniert und wie eine entsprechende Schnittstelle entworfen werden kann.

3.4.2. Verwandte Arbeiten

Einige verwandte Forschungsarbeiten haben bei der Analyse der Problemstellung geholfen und die Resultate dieser Arbeit damit mitbeeinflusst.

Die Arbeiten von *Hüffmeyer und Schreier* [19, 20, 21] der Universität Furtwangen beschreiben eine mögliche Lösung für eine REST-basierte ABAC-Schnittstelle als Alternative zu XACML.

Yuan und Tong, [48] sowie *Shen und Hong* [4] gehen auf die ABAC-Konzepte im Web Service Umfeld ein. *Dan, Hua-Ji, Yuan und Jia-Hu* [6] diskutieren in ihrer Arbeit ABAC und «Cross-Domain Access Control» im SOA Umfeld. Die Arbeit von *Oh und Kim* [34] bietet eine gute Einführung in die Thematik der dezentralen Zugriffskontrolle in Resourcenorientierten Architekturen (REST) für das Web. *Priebe, Dobmeier, Muschall und Pernul* [38] der Universität Regensburg stellen ein «Referenzmodell für attributbasierte Zugriffskontrolle» vor und damit einen guten Einstieg in das ABAC-Paradigma. Ausserdem stellt diese Arbeit eine interessante UML-Notation zur Dokumentation Attribut-basierter Zugriffsregeln vor. Das Dokument [33] des *National Institute of Standards and Technology NIST* ist eine gute Einführung in die Definitionen und Überlegungen der Attributbasierten Zugriffskontrolle. *Sandhu* [40] vergleicht RBAC und ABAC mit deren Vor- und Nachteilen und hat damit wertvolle Hinweise zum Architectural Refactoring «From RBAC to ABAC» (siehe Kapitel 5) geliefert.

4. Anforderungen

Mit dem aktuellen Objektschutz in SYRIUS lassen sich sämtliche BO's und deren Attribute schützen. Um im Rahmen dieser Studienarbeit einen Prototypen entwickeln zu können, wurden mit dem Industriepartner drei spezifische *Use Cases* ausgewählt. Die Auswahl der Use Cases sollte einen lesenden und einen schreibenden Zugriff auf ein BO, als auch den Schutz einzelner Attribute eines BO's enthalten. Der Prototyp soll diese Use Cases mit einem externen Autorisierungssystem (Mock-Implementation) lösen. Alle Anforderungen und Use Cases zu betrachten, welche SYRIUS aktuell anbietet, würde den Rahmen einer Studienarbeit sprengen.

In diesem Kapitel werden sowohl die funktionalen Anforderungen in Form von *Use Cases*, als auch die nichtfunktionalen Anforderungen an den Prototypen erläutert.

Ziel dieser Studienarbeit ist es, ein *Redesign der Persistenzschicht* in SYRIUS durchzuführen, welches für definierte Use Cases, die bestehende Objektschutz-Lösung durch einen Aufruf eines externen Autorisierungssystems ablöst. Die Implementation dieses Autorisierungssystems ist nicht im Fokus dieses Projekts, und wird daher als «Blackbox» betrachtet.

4.1. Aktoren und Stakeholder

Bevor auf die konkreten Use Cases eingegangen wird, stellen die nächsten Abschnitte alle beteiligten *Aktoren* und *Stakeholder* vor. Die Aktoren sind Personen oder Systeme, welche in den folgenden Use Cases direkt beteiligt sind und mit dem System interagieren. Die Stakeholder hingegen stellen bestimmte Anforderungen an das System, arbeiten jedoch nicht direkt mit diesem.

4.1.1. Aktoren

In der folgenden Tabelle 4.1 werden die Aktoren vorgestellt, welche im nächsten Kapitel in den Use Cases vorkommen. SYRIUS ist das zentrale System, in welchem sich die Use Cases abspielen und wird daher nicht als Aktor aufgelistet.

Tabelle 4.1.: Aktoren

Aktor	**Beschreibung**
Sachbearbeiter	Ein Sachbearbeiter ist ein Benutzer, welcher mit SYRIUS arbeitet. Er greift über den UTC-Client auf das System zu. In der realen Welt gibt es selbstverständlich verschiedene Ausprägungen der Rolle *Sachbearbeiter* mit unterschiedlichen Berechtigungen. Auch der Vorgesetzte eines Sachbearbeiters oder andere Mitarbeiter der Versicherung, haben die Möglichkeit auf das System zuzugreifen. Die Unterscheidung dieser Ausprägungen ist aber für die Betrachtung der Use Cases innerhalb dieser Studienarbeit unerheblich und es wird deshalb darauf verzichtet. Diese Entscheidung wurde in der Diskussion zusammen mit dem Industriepartner getroffen. Der Sachbearbeiter hat Lese- und/oder Schreib-Zugriff auf bestimmte Partner (BO) in SYRIUS, abhängig von seinen Berechtigungen. Dabei kann der Zugriff auf den gesamten Partner oder nur auf eine Teilmenge der Attribute gewährt werden.
Autorisierungssystem (Mock-Implementation)	Das Autorisierungssystem wird von SYRIUS aufgerufen und nimmt Berechtigungsanfragen entgegen. Diese Berechtigungsanfragen muss das Autorisierungssystem mit «PERMIT» oder «DENY» beantworten. Unter Umständen muss es dem Aufrufer auch noch mitteilen, auf welche Attribute des BO's der Benutzer zugreifen darf.

Neben den Aktoren gibt es weitere Anspruchsgruppen, welche ein Interesse daran haben, dass die Daten in SYRIUS durch ein Autorisierungssystem geschützt werden. Sie interagieren nicht direkt mit dem System, möchten aber, dass bestimmte funktionale sowie nichtfunktionale Anforderungen erfüllt werden. In der folgenden Tabelle 4.2 werden diese Stakeholder vorgestellt.

Tabelle 4.2.: Stakeholder

Stakeholder	**Beschreibung**
Kunde	Der Kunde möchte, dass seine persönlichen Daten vor Zugriffen und Manipulationen nicht autorisierter Personen geschützt werden. Er erwartet von seinem Versicherer daher, dass dieser entsprechende Sicherheitsmassnahmen umsetzt.

Tabelle 4.2.: Stakeholder

Stakeholder	**Beschreibung**
Versicherer	Der Versicherer ist seinerseits ebenfalls daran interessiert, die Daten seiner Kunden schützen zu können. Er möchte den Sachbearbeitern unterschiedliche Berechtigungen erteilen können, um zu steuern, welche Personen auf welche Daten zugreifen können. Für ihn ist es wichtig, dass diese Lösung sicher, aber auch performant funktioniert. Die Pflege der Berechtigungen soll dabei mit einem vernünftigen Aufwand möglich sein.
Architekt	Seitens der Produktentwicklung werden vor allem Anforderungen an die Architektur der Autorisierungslösung herangetragen. Berechtigungsanfragen sollen aus verschiedenen Systemen ausgeführt werden können. Dies ist zur Zeit nicht möglich, ohne dass die Persistenzschicht von SYRIUS aufgerufen wird.
Adcubum	Für Adcubum als Software-Hersteller ist es wichtig, dass die Wartbarkeit der Lösung wirtschaftlich ist. Diese Anforderung ist aktuell aufgrund der komplexen Lösung in der Persistenzschicht nicht mehr erfüllt. Ein entsprechendes Architektur-Refactoring muss diesem Problem entgegenwirken.

4.2. Use Cases (funktionale Anforderungen)

Für diese Studienarbeit wurden mit dem Industriepartner zusammen drei Use Cases ausgewählt, welche die wichtigsten Anforderungen an das Autorisierungssystem abdecken.

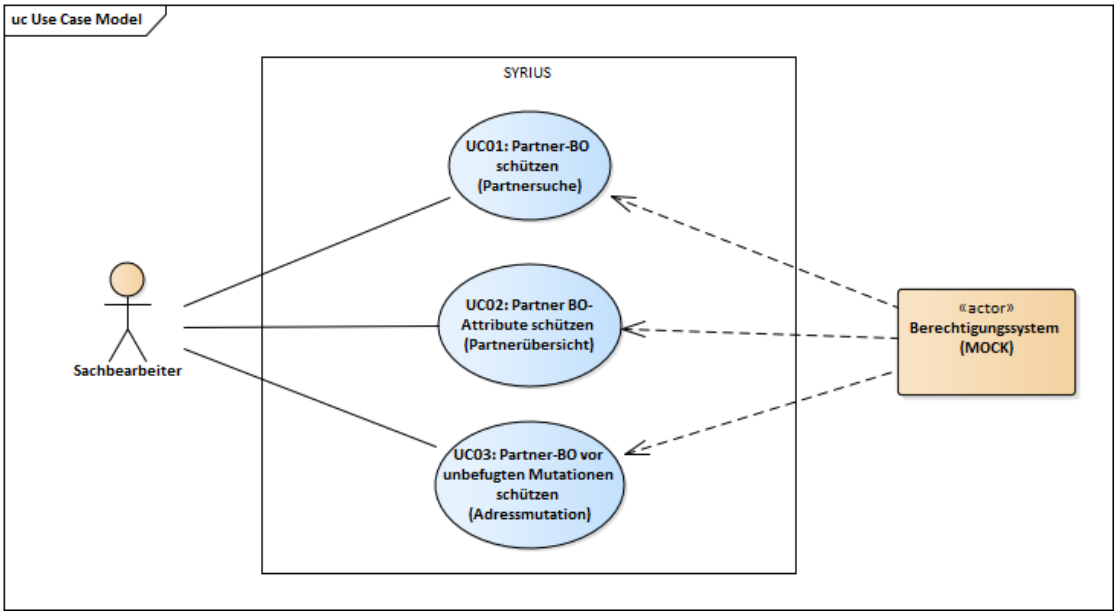

Abbildung 4.1.: Use Case Diagram

Die Use Cases decken im wesentlichen den Lese-, Mutations- und Attribut-Schutz, im spezifischen Fall des Partner-BO ab. Die Funktionalität der Schutzpfade muss bei der Auswahl der Use Cases nicht berücksichtigt werden, da das zukünftige Autorisierungssystem wissen muss, welche Berechtigungsregeln bei einem BO zur Anwendung kommen. Die dafür benötigten Informationen muss sich das ABAC-System im Attribut-Repository halten.

4.2.1. UC01: Partner-BO schützen

Das Partner-BO muss vor unbefugten Zugriffen geschützt werden. Bei diesem Use Case geht es um den Schutz des gesamten BO's. Hat ein Benutzer keine Lese-Berechtigung auf ein bestimmtes BO, darf er es in der Partner-Suchmaske von SYRIUS gar nicht erst finden.

4.2.2. UC02: Partner BO-Attribute schützen (Attributschutz)

Im Fall dieses Use Cases hat der Benutzer eine eingeschränkte Lese-Berechtigung auf das Partner-BO. Er kann das BO in der Partner-Suchmaske finden und auch die Partnerübericht öffnen. Allerdings kann er überall nur die Attribute sehen, auf welche er auch den Zugriff erhalten hat. Gesperrte Attribut-Werte werden auf den Benutzeroberflächen in SYRIUS mit «#####» angezeigt.

4.2.3. UC03: Partner-BO vor unbefugten Mutationen schützen (Adressmutation)

Einen weiteren Use Case stellt die «Mutation» dar. Ein Benutzer kann einen Lese-Zugriff auf ein BO besitzen, jedoch unter Umständen keinen Schreib-Zugriff. Die Mutationstasks sind auf den Benutzeroberflächen von SYRIUS deaktiviert. Eine Mutation darf allerdings nicht nur auf dem GUI verhindert werden, sondern muss auch von der Persistenz-Schicht von SYRIUS blockiert werden. Aus diesem Grund umfasst dieser Use Case nicht nur eine Mutation per UTC-Client, sondern auch über einen Webservice-Aufruf (SOAP [51]).

4.2.4. Use Cases im Detail

Die Use Cases sind im Anhang B detailliert im «Fully Dressed» Format und einem Szenario beschrieben.

4.3. Nichtfunktionale Anforderungen (NFA)

4.3.1. Qualitätsmerkmale

Sicherheit

Die Kommunikation zwischen dem neuen Autorisierungssystem und SYRIUS soll eine sichere Verbindung mittels HTTPS unterstützen, welche starke Verschlüsselungsalgorithmen einsetzt. Es gibt bei Adcubum keinen internen Standard oder eine Richtlinie diesbezüglich, womit diese Nichtfunktionale Anforderung (NFA) messbar spezifiziert werden könnte. Aus diesem Grund wird an dieser Stelle an den *Guide to Cryptography* [37] der OWASP verwiesen, welchem auch die kryptographischen Algorithmen entnommen werden können, welche zur Zeit als *sicher* eingestuft werden. Infrastruktur-Security wird bei dieser Studienarbeit jedoch bewusst ausgeblendet, da es nicht zum Schwerpunkt der Arbeit gehört.

Performance

Gemäss Absprache mit dem Industriepartner muss die Performance in dieser Studienarbeit nicht berücksichtig werden, da die Implementation des Autorisierungssystems auch noch nicht im Fokus steht. Es müssen auch noch keine Massnahmen getroffen werden, um die Performance zu verbessern (z.B. Implementation eines Caches). Die Performance wird im Betrieb des neuen Berechtigungssystems trotzdem ein entscheidender Faktor und damit eine wichtige NFA sein. Innerhalb der Studienarbeit soll bei Design-Entscheidungen und Auswahl von Schnittstellen-Technologien darauf geachtet werden dass später eine gute Performance erreicht werden kann. Zum Beispiel können Caching-Mechanismen vorgesehen werden, müssen aber nicht innerhalb der Studienarbeit umgesetzt werden. Hierfür ist eine interoperable Schnittstellentechnologie einzusetzen, welche z.B. auch für eine Integration von Elasticsearch [7] geeignet ist. Binäre Protokolle sind auf expliziten Wunsch des Industriepartners zu vermeiden, da mit deren Verwendung auch Binäre Abhängigkeiten und Plattformabhängigkeiten einhergehen. Ausserdem soll es möglich sein, den Inhalt der Autorisierungsantworten an gewissen Stellen im Netzwerk überwachen und loggen zu können.

Lizenzen

In Bezug auf Thirdparty Libraries dürfen bei Adcubum nicht beliebige Lizenzen verwendet werden. Offene Lizenzen wie zum Beispiel die 'Apache Licence 2.0' [35] sind erlaubt, GPL3 [18] hingegen nicht. Es wird an dieser Stelle auf das interne Dokument *Anbindung Fremdbibliotheken* [1] von Adcubum verwiesen, welches die erlaubten Lizenzen auflistet.

Dieses Dokument gilt auch für diese Studienarbeit und ist zwingend einzuhalten.

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:

- Zeitstempel (Eindeutiger Zeitpunkt des Zugriffs)
- Identität des Benutzers
- Parameter und deren Werte, welche an die Schnittstelle übergeben wurden
- Entscheidung des Berechtigungssystems: Zugriff gewährt oder nicht

Interoperabilität der Schnittstelle

Die Schnittstelle des Autorisierungssystems soll so designed werden, dass sie fähig ist mit verschiedenen Umsystemen unabhängig derer Technologie, zu kommunizieren. Es ist sicherzustellen, dass die Schnittstelle technologieunabhängig implementiert wird. Auf ihrer Webseite [22] beschreibt das *Europäische Institut für Telekommunikationsnormen ETSI* was unter Interoperabilität zu verstehen ist und was deren Vorteile sind.

Einfachheit der Schnittstelle

Die Schnittstelle soll für deren Benutzer einfach zu nutzen sein. Das Design ist so simpel wie möglich zu halten. Ausserdem ist eine Dokumentation der Schnittstelle zu erstellen, welche jede Methode, deren Parameter und funktionsweise verständlich dokumentiert. Mit der Dokumentation soll ein Entwickler in der Lage sein, die Schnittstelle nach einer Stunde Einarbeitungszeit, zu verstehen.

Schnittstelletechnologie

Der Industriepartner schreibt keine Schnittstellen-Technologie vor. Das Hauptziel dieser Arbeit ist es, zu ermitteln, welche Daten dem Autorisierungssystem übergeben werden müssen. Die ausgewählte Technologie soll aber interoperabel sein, damit Umsysteme verschiedenster Technologien die API benutzen können. Wie im obigen Abschnitt *Performance* bereits erwähnt, soll bei der Umsetzung der API darauf geachtet werden, dass sich Performance-Optimierende Massnahmen, wie z.B. Caching, gut umsetzen lassen.

4.3.2. Randbedingungen

Externes System

Dass das neue Berechtigungssystem ein *externes* System sein soll, ist die entscheidenste Randbedingung in diesem Projekt. Es ist stets darauf zu achten, dass die Unabhängigkeit gegenüber SYRIUS gewahrt wird, sodass wirklich von einem *externen* System gesprochen werden kann.

Dadurch werden verschiedene Deployment-Modelle möglich, in welchen das Autorisierungssystem auch auf anderen Servern wie SYRIUS betrieben werden kann.

4.4. Abgrenzungen

4.4.1. Alternative Szenarien

In den in diesem Kapitel spezifizierten Use Cases wird jeweils auf eine spezifische Art und Weise auf die BO's zugegriffen. Natürlich gibt es in SYRIUS diverse alternative Szenarien, wie auf die Daten dieser BO's zugegriffen werden kann. Als Beispiel könnte ein Partner nicht über die Partnersuche, sondern über einen anderen Service gesucht werden. Häufig greifen Angreifer eines Systems auch auf Daten zu, indem ID's von Objekten erraten werden.

Der Vorteil der aktuellen Lösung ist, dass die Berechtigungsprüfung in jedem Fall durchgeführt wird, da sie tief in der Persistenzschicht eingebaut ist. Ein «Bypassing» wäre nur möglich, wenn der Benutzer direkt mittels JDBC auf die Datenbank zugreifen würde. Auch in der neuen Lösung ist zu berücksichtigen, dass das Autorisierungssystem an einer zentralen Stelle des Persistenz-Frameworks von SYRIUS aufgerufen wird. Damit soll sicherergestellt werden, dass auch in Zukunft alle Szenarien abgedeckt werden können und kein «Bypassing» des Autorisierungssystems durch den Fachentwickler oder einen Angreifer möglich ist.

Allerdings können in dieser Studienarbeit nur die bereits erwähnten Use Cases betrachtet und getestet werden. Auch wenn das Lösungsdesign grundsätzlich dafür ausgelegt sein soll alle Szenarien abzudecken, so ist es innerhalb dieser Arbeit trotzdem nicht möglich, alle Szenarien zu identifizieren und zu testen.

4.4.2. Spezialberechtigungen

Es gibt in SYRIUS sogenannte *Spezialberechtigungen*. Zum Beispiel kann für eine *Ferienvertretung* eine Spezialberechtigung definiert werden, welche es einer Vertretung eines Sachbearbeiters über eine spezifische Zeitdauer erlaubt, dessen Berechtigungen zu erhalten. So kann der oder die vertretende Sachbearbeiter(in) auf die Daten der Kunden des abwesenden Sachbearbeiters zugreifen.

In Absprache mit dem Industriepartner werden die Spezialberechtigungen innerhalb dieser Studienarbeit nicht betrachtet, da es sich um ein Implementationsdetail des zukünftigen Autorisierungssystems handelt. Ob und wie dies mit einem ABAC-basierten System möglich ist, wurde im Rahmen dieser Arbeit nicht überprüft. Diese Fragestellung wird vermutlich in der Folgearbeit (Bachelorarbeit) genauer betrachtet werden.

4.4.3. Implementation des Autorisierungssystems

Die Implementation des Autorisierungssystems ist nicht Teil dieser Studienarbeit. Für den Prototyp soll eine Mock-Implementation erstellt werden, welche für die in den detaillierten Szenarien der Use Cases in Anhang B statische Antworten zurückliefert.

Nachdem die Anforderungen an den zu entwickelnden Prototypen definiert wurden, geht das nächste Kapitel auf das Design und die Implementation ein.

5. Design und Implementation

In Abbildung 2.3 des Kapitels 2 wurde bereits klar, mit welchen Umsystemen das Autorisierungssystem kommunizieren wird. Abbildung 5.1 konkretisiert dieses Bild aus der Perspektive des Autorisierungssystems in Form eines UML Komponentendiagramms.

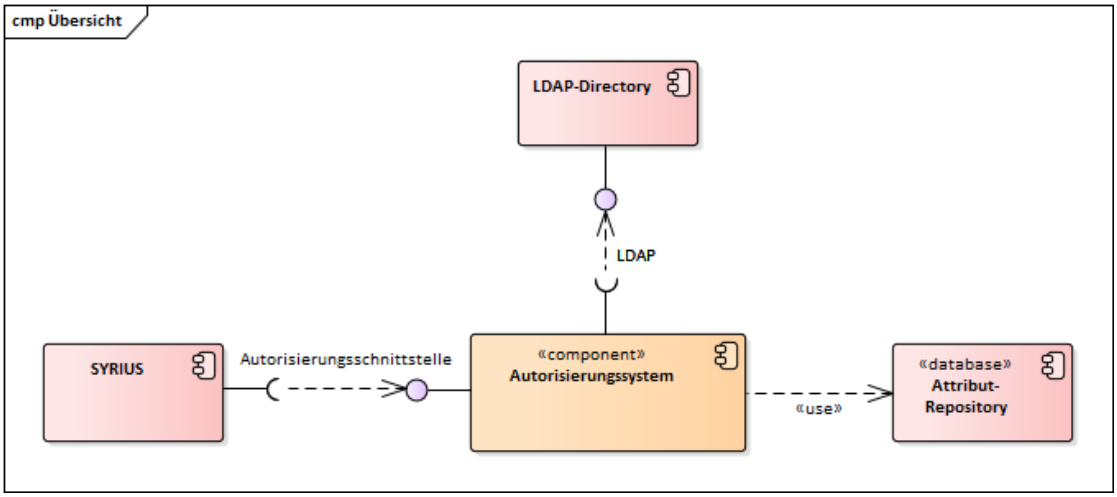

Abbildung 5.1.: Autorisierungssystem (UML Komponentendiagramm)

Der während dieser Studienarbeit entwickelte Prototyp beinhaltet sowohl eine Mock-Implementation des in Abbildung 5.1 gezeigten Autorisierungssystems, als auch die Autorisierungsschnittstelle und ein Redesign der Persistenzschicht in SYRIUS. Das Autorisierungssystem wird zukünftig ein LDAP-Verzeichnis anbinden und ein Attribut-Repository verwenden, um alle Attribute der Benutzer und Objekte zu beziehen, welche für die Autorisierung notwendig sind. Für Mock-Implementation ist dies jedoch nicht notwendig, da die Autorisierungsregeln statisch hinterlegt sind.

Dieses Kapitel dokumentiert sowohl die Design- und Architekurentscheidungen welche getroffen wurden, um diesen Prototypen zu entwickeln und die Anforderungen aus Kapitel 4 zu erfüllen, als auch die wichtigsten Implementationsdetails.

5.1. Design- und Architekturentscheidungen

5.1.1. «From RBAC to ABAC»

In Kapitel 3 wurde bereits erläutert, dass die Ablösung von RBAC durch ein flexibleres Zugriffskontrollmodell angestrebt werden soll. Das Ziel ist also nicht nur die Autorisierungslösung aus SYRIUS zu extrahieren und einen eigenen «Bounded Context» bzw. «Microservice» zu erstellen, sondern auch ein architekturelles Refactoring welches eine Migration der Zugriffskontrolldaten erfordert. Konkret müssen die Rollen des RBAC-Systems in Attribut-basierte Regeln (ABAC) überführt werden.

Der Begriff des «Architectural Refactoring» stammt von Michael Stal [30] und wurde von Olaf Zimmermann weiterentwickelt [49, 50]. Ein «Architectural Refactoring» schlägt eine *Architekturumbaumassnahme* vor, um bestimmten «Smells» welche im System vorhanden sind, entgegenzuwirken. Olaf Zimmermann [49] hat in seiner Arbeit ein Template entworfen, um solche «Architectural Refactorings» in einer wiederverwendbaren Form zu dokumentieren. Christian Bisig hat in seiner Bachelorarbeit [5] das Tool «ART», ein werkzeugunterstütztes Knowledge Repository für «Architectural Refactoring's», entwickelt.

Das Vorhaben des Industriepartners, das Zugriffskontrollmodell von RBAC nach ABAC umzustellen, wird nachfolgend mit dem von Olaf Zimmermann vorgeschlagenen Template dokumentiert.

Tabelle 5.1.: Architectural Refactoring: «From RBAC to ABAC»

Architectural Refactoring	«Change Access Control Model From RBAC To ABAC»
Context	• Logische Sicht
Stakeholder concerns and quality attributes (design forces)	• Flexibilität in der Gestaltung der Zugriffs-Regeln. • Geringe Kopplung zwischen Benutzer und zu schützendem Objekt.
Architectural smell	• Um die nötige Flexibilität zu erreichen, müssen häufig künstliche *Pseudo-Rollen* erstellt werden, was oft zu einer *Role Explosion* [40, 33, 21] führt. • Die starke Kopplung zwischen den Objekten und den Rollen bzw. Benutzern, führen dazu, dass die Implementation der Autorisierung und die Businesslogik ebenfalls eine hohe Kopplung aufweisen.

Architectural decision(s)	• Wahl eines Zugriffskontrollparadigma's (in diesem Fall ABAC-Paradigma). • Wahl oder Implementation einer Autorisierungsschnittstelle welche das Paradigma unterstützt. • Wahl oder Implementation eines Autorisierungssystems.
Description	• **Initial position:** In einem Softwaresystem, welches mit Rollen zur Autorisierung arbeitet (RBAC), ist die Granularität der Rollen aufgrund der fehlenden Flexibilität des Zugriffskontrollmodels nicht mehr adäquat, um eine genügend feingranulare Autorisierung umzusetzen. Ausserdem referenzieren die Rollen und Benutzer *statisch* die zu schützenden Objekte innerhalb der Persistenz der Applikation. • **Revised design:** Ziel dieses Architectural Refactoring ist es, die Gestaltung der Autorisierungsregeln flexibler zu gestalten und dabei die Kopplung zwischen den Benutzern und deren Rollen und zu schützenden Objekten zu verringern. Die Regeln werden über Attribute definiert. Dabei ist ein Attribut ein beliebiges *Key-Value-Pair* und kann vom Benutzer, dem zu schützenden Objekt oder auch vom System stammen. Die Autorisierungsregeln definieren, mit welchen Werten die Attribute der Benutzer, Objekte und Systeme belegt sein müssen, damit ein Zugriff gewährt wird oder nicht. Dadurch können beliebig flexible Regeln definiert werden und es besteht keine *statische* Verbindung zwischen dem Benutzer und dem zu schützenden Objekt. Für den Vergleich von RBAC und ABAC sei auf das Kapitel 3 verwiesen. • **Pitfalls to avoid:** Die Flexibilität, welche ABAC mit sich bringt, kann ebenfalls zu einer unübersichtlichen Anzahl und Komplexität von Regeln führen. Das Attribute Engineering [40] muss wohlüberlegt durchgeführt werden, damit die neu gewonnene Flexibilität nicht zu einem potentiellen «Chaos» von komplexen Regeln führt.

Affected architectural elements	• Autorisierungskomponente • Applikations-Schicht (Datenzugriffsschicht) welche Autorisierungsaufruf durchführt. • Weitere Systeme aus welchen das Autorisierungssystem Daten bezieht, wie zum Beispiel das «Identity Management» System.
Execution tasks	• Implementiere ABAC-basiertes Autorisierungssystem oder verwende ein Standard-Produkt. • Entwerfe eine Remote-Schnittstelle, um Autorisierungsrequests durchführen zu können. • Refaktoriere die Datenzugriffsschicht, in welcher die Autorisierung der Daten durchgeführt wird. Rufe dort die Remote-Schnittstelle des neuen ABAC-basierten Autorisierungssystems auf und filtere anschliessend die Daten, auf welche der Benutzer keinen Zugriff erhalten hat. Ein Beispiel dieses Ablaufs in Form einer Prozess-Sicht wird später in diesem Kapitel in der Abbildung 5.5 gezeigt. • Migriere die bestehenden Rollen zu Attribut-basierten Regeln. • Prüfe ob Design-Ziele und Requirements mit dem Architectural Refactoring erreicht wurden. • Prüfe ob Performance- und Audit-Anforderungen erfüllt werden.

Die vollständige Umsetzung dieses «Architectural Refactoring's» ist ein dieser Studienarbeit übergeordnetes Ziel. Einzelne Schritte dieses «Architectural Refactorings» wie die Definition der Schnittstelle zum Autorisierungssystem und das Redesign der Persistenzschicht zur Nutzung des neuen Autorisierungssystems, sind aber Bestandteile des in dieser Arbeit entwickelten Prototypen.

5.1.2. Eigenständige Komponente («Bounded Context»)

Dass die Autorisierung in Zukunft eine eigenständige Komponente bilden soll, ist nicht nur eine wichtige Architekturentscheidung sondern auch eine der Anforderungen, welche von Beginn weg an dieses Projekt gestellt wurde.

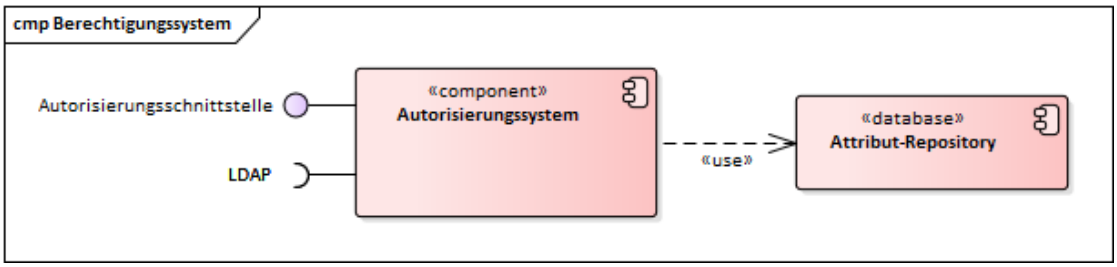

Abbildung 5.2.: Eigenständige Komponente (UML Komponentendiagramm)

Im Kontext der aktuellen Autorisierungslösung wurde, in Anbetracht der zu starken Kopplung und der umständlichen Autorisierung verteilter Datenteilbestände, die Entscheidung getroffen, eine eigenständig «deploybare» Komponente zu erstellen. Dadurch wird eine bessere Wartbarkeit der Autorisierungslösung und der Persistenzschicht in SYRIUS erreicht. Ausserdem wird die Autorisierung extern gehaltener Daten dadurch einfacher.

Mit dieser Entscheidung werden die zusätzlichen Herausforderungen akzeptiert, welche die Autorisierung über eine Remote-Schnittstelle in Bezug auf die *Performance* mit sich bringen wird.

In [5] wurde das dazu passende «Architectural Refactoring» «Split components to reduce overly tight coupling» vorgestellt, in welchem ebenfalls die Wartbarkeit (*Maintainability*) als Qualitätsattribut hervorgehoben wird.

5.1.3. Fachlich orientierte Schnittstelle

Die Schnittstellen standardisierter Autorisierungssysteme sind sehr «generisch» aufgebaut, da sie in den verschiedensten fachlichen Domänen zum Einsatz kommen. Der XACML-Standard [32], welcher das ABAC-Paradigma implementiert, ist ein solch generisches Beispiel. Der Standard spezifiziert einen Request als eine Liste beliebiger Attribute, wie Auflisting 5.1 zeigt.

```
<Request CombinedDecision="false" ReturnPolicyIdList="false">
    <Attributes Category="action">
        <Attribute AttributeId="action-id" IncludeInResult="false">
            <AttributeValue DataType="string">READ</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes Category="access-subject">
        <Attribute AttributeId="subject-id" IncludeInResult="false">
            <AttributeValue DataType="string">TestUser</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes Category="resource">
        <Attribute AttributeId="resource-id" IncludeInResult="false">
            <AttributeValue DataType="string">/partner/5</AttributeValue>
        </Attribute>
    </Attributes>
</Request>
```

Auflistung 5.1: XACML Beispiel-Request

In der Arbeit von Hüffmeyer und Schreier [21] der Universität Furtwangen, wird eine REST-basierte Zugriffskontroll-Schnittstelle (RestACL) als Alternative zu XACML im RESTful HTTP Umfeld vorgeschlagen.

Auch diese Schnittstelle ist wie XACML eine generische Schnittstelle. Ein entsprechender Request ist, wie Abbildung 5.2 zeigt, einem XACML-Request aus der Sicht des Datenmodells sehr ähnlich. Der Unterschied ist hier die starke Orientierung an den «Resourcen» des RESTful HTTP.

```
{
    "uri": "http://example.org/partner/5",
    "method": "GET",
    "attributes": [{
        "category": "subject",
        "designator": "subject-id",
        "value": "TestUser"
    },{
        "category": "resource",
        "designator": "resource-id",
        "value": "5234"
    }]
}
```

Auflistung 5.2: RestACL Beispiel-Request [21] (JSON)

Eine entsprechende Response ist in beiden Lösungen im Wesentlichen ein einfaches «PERMIT» oder «DENY».

```
{
    "decision": "PERMIT"
}
```

Auflistung 5.3: RestACL Beispiel-Response [21] (JSON)

Der Vorteil einer solch generischen Schnittstelle ist die Flexibilität. Sie kann unabhängig von der Fachlichkeit der Applikation, welche die Autorisierungsrequests durchführt, eingesetzt werden.

Wird aus der Applikation direkt eine solche Autorisierungsschnittstelle aufgerufen, bringt dies allerdings auch einen Nachteil mit sich. Einerseits muss die Schnittstelle vom Fachentwickler, welcher die Applikation entwickelt, zuerst verstanden werden. Er kann das Modell, welches er aus der Applikation kennt, in der Schnittstellendefinition nicht wiederfinden. Andererseits muss die Fachapplikation das eigene Modell in das generische Modell der Autorisierungsschnittstelle konvertieren, damit sie einen Request schicken kann.

Dies benötigt zusätzliche Programmlogik in der Applikation, um die Konvertierung des Modells vorzunehmen. In der Diskussion mit dem Industriepartner wurde die Designentscheidung getroffen, dass eine solche Modellkonvertierung in SYRIUS nicht erwünscht ist. Die Schnittstelle, welche für den Prototypen innerhalb dieser Studienarbeit erar-

beitet wurde, sollte fachlich orientiert sein und das Modell aus SYRIUS abbilden. Dies vereinfacht den Aufruf der Schnittstelle aus SYRIUS, aber auch aus anderen Systemen wie Elasticsearch [7], weil das Modell mit diesen Systemen übereinstimmt. Ein weiterer Vorteil davon ist, dass die Schnittstelle nicht beliebig generische Attribute annimmt und dadurch das Risiko semantisch nicht korrekter Aufrufe seitens der Applikationsentwickler verringert werden. Desweiteren führt eine einfachere Schnittstelle auch zu einer Reduktion von Angriffsvektoren.

Diese Architekturentscheidung unterstützt ausserdem die Erfüllung der NFA «Einfachheit der Schnittstelle» (siehe Abschnitt 4.3). Ein Fachentwickler kann die Schnittstelle in kurzer Zeit verstehen, da er die Begriffe des API-Modells aus der SYRIUS-Domäne bereits kennt.

Trotzdem kann die Schnittstelle, welche das Autorisierungssystem anbietet, eine *generische* sein. Vor allem dann wenn das System nicht selbst implementiert, sondern ein Standardprodukt eingesetzt wird, bei welchem man keinen Einfluss auf die Schnittstelle hat. In diesem Fall wird ein «Gateway» [10] eingesetzt, welcher unsere fachliche Schnittstelle zur Verfügung stellt und die generische Schnittstelle des Autorisierungssystems aufruft.

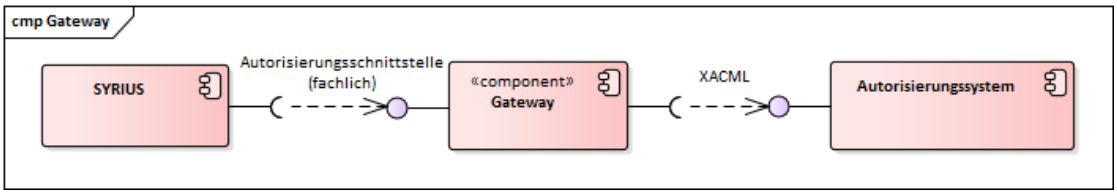

Abbildung 5.3.: Gateway (UML Komponentendiagramm)

Ein kurzer Ausschnitt aus dem Buch von *Martin Fowler* [10] beschreibt den Gateway folgendermassen:

> «In reality this is a simple wrapper pattern. Take the external resource. What does the application need to do with it? Create a simple API for your usage and use the *Gateway* to translate to the external source.»

Damit beschreibt er exakt, was in diesem Falle erreicht werden soll. Nämlich SYRIUS eine «simple» API zur Verfügung zu stellen, welche der Fachlichkeit entspricht und darum einfach aufzurufen ist. Der Gateway transformiert dann den Aufruf auf die externe Schnittstelle, wie zum Beispiel XACML. Worauf Fowler in diesem Pattern nicht eingeht, ist die Frage, ob der Gateway direkt in der Applikation oder in einer eigenen Komponente (Remote) implementiert wird. Da eine lose Kopplung zwischen dem Autorisierungskontext und SYRIUS geschaffen werden soll, kann der Gateway eine eigene Komponente bilden, wie in Abbildung 5.3 dargestellt.

Im Kontext der zu erstellenden Schnittstellendefinition wurde aufgrund der Komplexität der existierenden Standardschnittstellen entschieden, eine einfachere und fachlich orientierte Schnittstelle zu implementieren, um die Komplexität für die Schnittstellenbenutzer zu verringern.

5.1.4. Schnittstellentechnologie

Dieser Abschnitt widmet sich nun der Wahl einer geeigneten Schnittstellentechnologie für die Autorisierungsschnittstelle.

RPC vs. Asynchrones Messaging

Bei der Auswahl der Schnittstellentechnologie stellt sich als erstes die Frage, ob es ein «Remote Procedure Call (RPC)» oder «Messaging» sein soll. Die Wahl ist in dieser Studienarbeit auf RPC gefallen. Der Grund dafür ist die Tatsache, dass die fachliche Problemstellung grunsätzlich *synchron* ist. Auf eine Autorisierungsanfrage benötigen wir sofort eine Antwort, da ansonsten die Datenabfrage nicht weiterverarbeitet werden kann.

Eine asynchrone Lösung bringt aus diesem Grund keinerlei Vorteile für diese Problemstellung.

SOAP vs. RESTful HTTP

Desweiteren ist die Interoperabilität eine entscheidende NFA. Die Schnittstelle soll von verschiedenen Programmiersprachen und Technologien unterstützt werden können. Aus dieser Ausgangslage bieten sich folgende beiden Technologien an:

- SOAP
- RESTful HTTP

Grundsätzlich wären für die Implementation des Prototyps beide Schnittstellentechnologien möglich gewesen. Wir haben uns vor allem aus einem Grund für eine RESTful HTTP Schnittstelle entschieden. Und zwar gibt es dabei einen signifikanten Vorteil bei der Integration von Elasticsearch [7], da dieses System ihre Schnittstellen ebenfalls über RESTful HTTP anbietet.

Wie bereits erwähnt wurde, sind binäre Protokolle auf Wunsch des Industriepartners zu vermeiden. Die Argumentation dazu befindet sich in Kapitel 4, Absatz 4.3.

5.1.5. Schnittstellendokumentation

Zur Modellierung und Dokumentation der Schnittstellendefinition wurde Swagger [43] eingesetzt, da es aktuell wohl eines der populärsten Tools im RESTful HTTP Umfeld ist. Der Industriepartner hat in diesem Zusammenhang kein Tool vorgeschrieben, Swagger wird aber auch bei Adcubum in Projekten eingesetzt. Aus diesen Gründen wurde auf eine ausführliche Evaluation verzichtet. RAML [16] wäre ebenfalls eine mögliche Option gewesen. Die Wahl des Tools wurde trotzdem mit dem Industriepartner abgesprochen und gutgeheissen.

5.1.6. Zusammenfassung Architekturentscheidungen

Die folgende Tabelle fasst die getroffenen Architekturentscheidungen noch einmal zusammen.

Tabelle 5.2.: Architekturentscheidungen

Problemstellung	**Gewählte Option**	**Verworfene Option**	**Begründung**
Wahl des Zugriffkontrollmodells	Attribute-based Access Control (ABAC)	Role-based Access Control (RBAC)	• Grössere Flexibilität • Geringere Kopplung zwischen Subjekt und Objekt
Implementation der Autorisierungslösung	Eigenständige Komponente	Lösung innerhalb von SYRIUS	• Einfachere Autorisierung externer Datenteilbestände • Wartbarkeit • Lose Kopplung
Schnittstellen-Modellierung	Fachlich, am SYRIUS-Modell orientierte Schnittstelle	Verwendung einer Standard ABAC-Schnittstelle	• Einfachheit der Schnittstelle • Geringere Komplixität für Aufrufer
Schnittstellen-Technologie	RESTful HTTP	Asynchrones Messaging, SOAP	• Problematik der Autorisierung ist synchron • Vorteile von RESTful HTTP gegenüber SOAP aufgrund der Anbindung zu Elasticsearch [7]

Nachdem die wichtigen Design- und Architekturentscheidungen erläutert wurden, wird nun auf die einzelnen Umsetzungbestandteile des Prototypen, dem *Redesign in SYRIUS*, der *Autorisierungsschnittstelle* und der *Implementation des Autorisierungssystems* (Mock-Implementation), eingegangen.

5.2. Redesign der Persistenzschicht

In diesem Abschnitt wird auf das in SYRIUS durchgeführte Redesign der Persistenzschicht zur Nutzung der neuen Autorisierungsschnittstelle eingegangen.

5.2.1. Ausgangslage

Das SYRIUS Persistenz-Framework bietet verschiedene Möglichkeiten wie die BO's aus der Datenbank gelesen werden können. Betrachtet wurden während diesem Redesign vor allem diejenigen Stellen welche für die Use Cases (siehe Kapitel 4) relevant sind.

Während der Analyse wurde festgestellt, dass aufgrund der Tatsache, dass die heutige Berechtigungslösung auf einer Erweiterung der WHERE-Klausel basiert, die Logik für den Objektschutz in diversen Query-Methoden eingebaut wurde. Darin hat sich die Aussage des Industriepartners bestätigt, dass die Kopplung der aktuellen Berechtigungslösung und der Persistenzlogik gross ist und dies eine Herausforderung bezüglich der Wartbarkeit der Persistenzschicht ist.

Die Analyse hat gezeigt, dass die Datenzugriffe für die definierten Use Cases im Wesentlichen in zwei zentralen Klassen des Persistenzframeworks durchgeführt werden. Einerseits im sogenannten *GenericStatementGenerator*, mit welchem Suchabfragen (Queries) durchgeführt werden, und andererseits in der *DBSession*, aus welcher einzelne BO's über bereits bekannte BO-ID's geladen werden.

5.2.2. «Security is an Aspect»

Die Aspektorientierte Programmierung (AOP) ist ein Programmierparadigma, um generische Funktionalitäten über mehrere Klassen und Methoden hinweg zu verwenden. In der Literatur wird dabei von sogenannten «Cross-Cutting Concerns» gesprochen. *Application Security* bzw. *Zugriffskontrolle* ist ein hervorragendes Beispiel eines solchen «Aspekt's» [47]. Die Logik, welche die Zugriffskontrolle durchführt, sollte innerhalb des Source-Codes einer Query-Methode des Persistenzframeworks unsichtbar sein. Dadurch wird eine klare Trennung der Fachlogik und des «Cross-Cutting Concern» erreicht.

AOP wird in der Regel via dem «Proxy Pattern» [12] oder einer «Bytecode Manipula-

tion» implementiert. Das «Proxy Pattern» wurde in diesem Redesign genutzt, um den Aufruf des Autorisierungssystems durchzuführen. Auf diese Weise muss der Aufruf nicht mehr direkt in der Klasse bzw. der Methode der Persistenzschicht (Query-Methode) durchgeführt werden.

5.2.3. Redesign am Beispiel des *GenericStatementGenerator*

Wie bereits erwähnt, musste für die definierten Use Cases in den Methoden der beiden Klassen *GenericStatementGenerator* sowie *DBSession* ein Redesign vorgenommen werden.

Folgende Schritte wurden durchgeführt, um die alte Berechtigungslösung im *GenericStatementGenerator* durch einen Aufruf des neuen Prototypen abzulösen:

- Entfernen der WHERE-Klause Erweiterungen in den Query-Methoden des *GenericStatementGenerator*.
- Implementation eines Proxy welcher das *StatementGenerator*-Interface implementiert.
- Aufruf des Autorisierungssystems (Prototyp) im Proxy und anschliessendes Filtern der Daten.

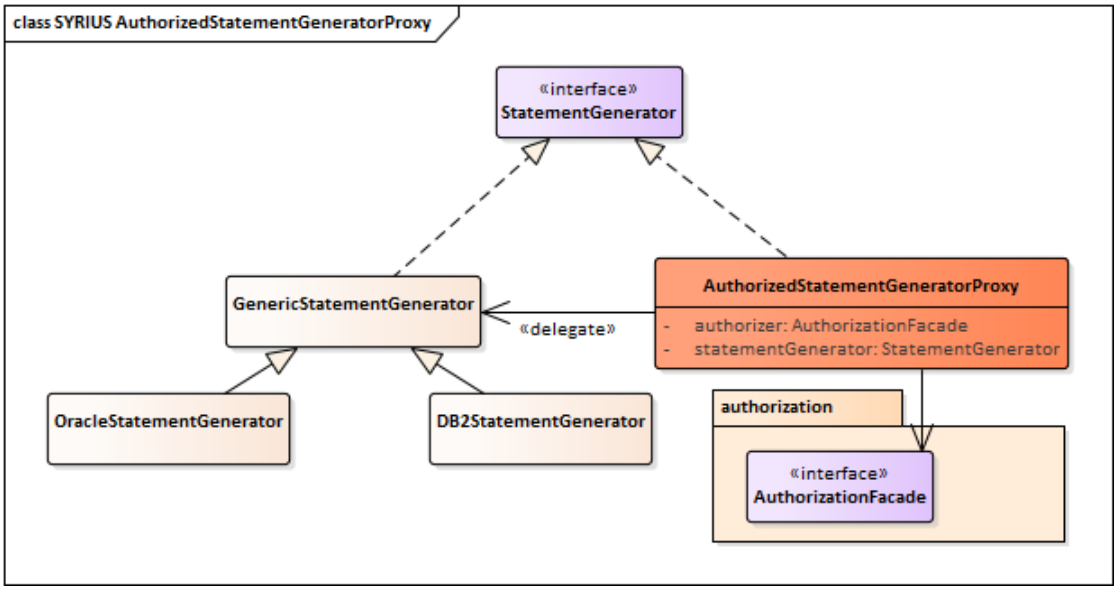

Abbildung 5.4.: AuthorizedStatementGeneratorProxy (UML Klassendiagramm)

Der *AuthorizedStatementGeneratorProxy* (siehe Abbildung 5.4) ruft nun den eigentlichen *StatementGenerator* auf und filtert die Daten welche dieser zurückliefert aufgrund der Antwort, welche er vom Autorisierungssystem erhält. Das Autorisierungssystem wird, wie in Abbildung 5.4 ersichtlich, über die *AuthorizationFacade* aufgerufen. Diese *AuthorizationFacade* benutzt letztendlich die vom Autorisierungsprototypen bereitgestellte Client-Library, um den Autorisierungsserver per RESTful HTTP aufzurufen. Diese Client-Library wird später in diesem Kapitel, bei der Erläuterung des Prototypen des Autorisierungssystems, vorgestellt.

Dieser Prozess, in welchem die Daten aus der Datenbank geladen und gleich anschliessend gefiltert werden, wird in der Abbildung 5.5 aus einer Prozess-Sicht verdeutlicht.

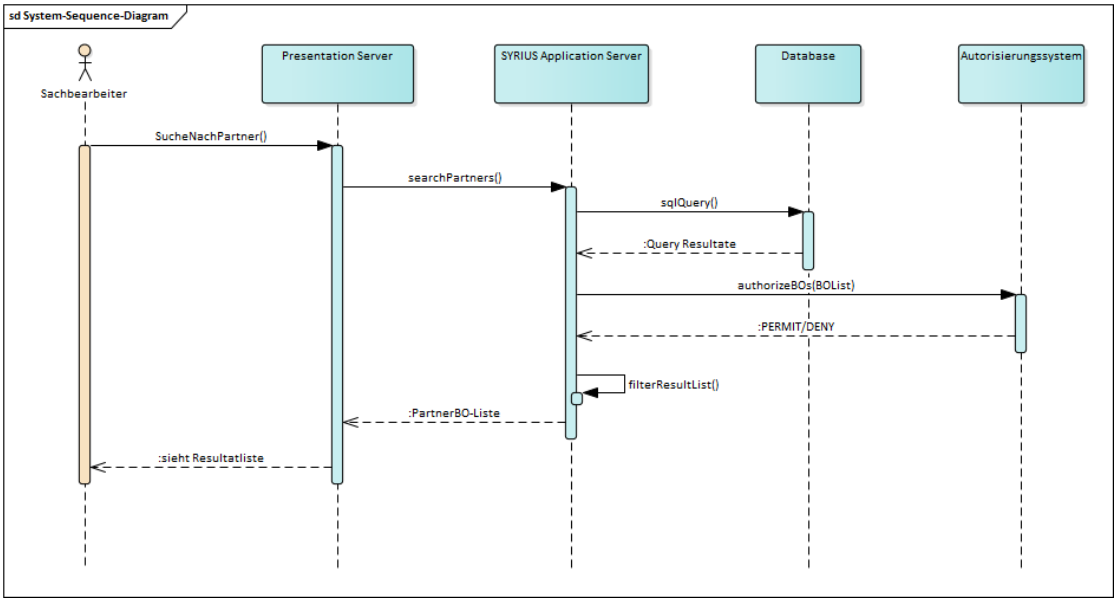

Abbildung 5.5.: Prozess-Sicht: Filtern der Daten (UML Sequenzdiagramm)

Der Applikationsserver lädt die Daten aus der Datenbank und macht anschliessend eine Autorisierungsanfrage an das *Autorisierungssystem*, um die Daten filtern zu können. Er schickt anschliessend nur diejenigen BO's an den Presentation-Server zurück, für welche der Benutzer auch eine Berechtigung besitzt.

5.2.4. Herausforderung des Redesigns

Vor allem für den zweiten und dritten Use Case (siehe Kapitel 4) ist die Implementation der *DBSession* relevant. Dort werden die Daten mittels einer BO-ID, welche der Client bereits kennt, geladen. Ausserdem verwendet die *DBSession* eine zusätzliche Klasse

VOPersistence, welche ebenfalls Datenbankabfragen durchführt. Ziel des Redesigns war ursprünglich, auch bei der *DBSession* das Prinzip des «Proxy Pattern» anzuwenden, so wie dies mit dem *StatementGenerator* implementiert wurde. Während des Redesign musste aufgrund aktueller Gegebenheiten im Code von SYRIUS festgestellt werden, dass diese Umstellung nicht in der zur Verfügung stehenden Zeit durchgeführt werden kann. Dies hat auch damit zu tun, dass die *DBSession* eine zentrale Klasse des Persistenzsystems ist und viele Abhängigkeiten hat. Eine entsprechende Umstellung hätte grosse Änderungen im gesamten Persistenzframework mit sich gezogen.

Aus diesem Grund wurde das Redesign für diese beiden Use Cases vereinfacht. Dies bedeutet, dass der Aufruf des Autorisierungssystems an diesen Stellen nach wie vor direkt in der Implementation der Methoden der *DBSession* gemacht werden.

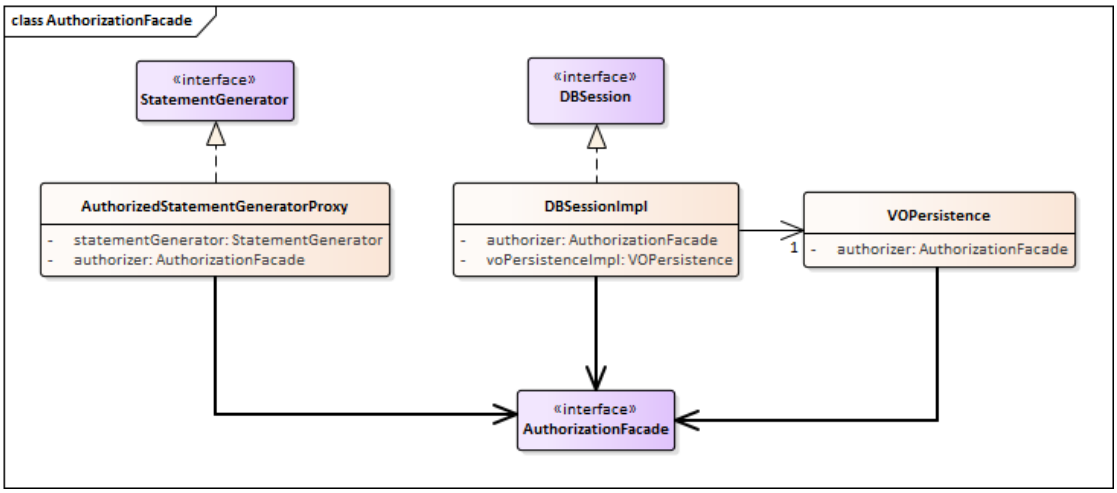

Abbildung 5.6.: Aufrufe der *AuthorizationFacade* (UML Klassendiagramm)

Abbildung 5.6 beantwortet eine wesentliche Frage welche im Rahmen dieser Studienarbeit beantwortet werden soll. Sie zeigt die Stellen, an welchen das neue Autorisierungssystem in der SYRIUS-Persistenzschicht aufgerufen werden muss. Die in dem Redesign eingeführte *AuthorizationFacade* bzw. deren Aufrufer zeigen diese Stellen deutlich auf. In Anhang C wird dies anhand eines ausführlicheren UML-Diagramms noch einmal verdeutlicht.

Allerdings muss erwähnt werden, dass diese Aussage nur für die in dieser Studienarbeit betrachteten Use Cases gemacht werden kann. Es besteht die Möglichkeit, dass weitere Use Cases neue Stellen hervorbringen können. Aufgrund der Komplexität des Persistenzframeworks war es mit der Analyse der definierten Use Cases nicht möglich, die Frage, ob alle Stellen identifiziert wurden, abschliessend zu beantworten. Dies wäre in der verfügbaren Zeit dieser Studienarbeit nicht möglich gewesen. Auf die Schlussfolgerungen dieser Problematik wird im Kapitel 6 genauer eingegangen.

5.3. Schnittstellendefinition

Nachdem das Redesign in SYRIUS erläutert wurde, wird in diesem Kapitel die Schnittstelle des Autorisierungssystems dokumentiert.

5.3.1. API-Modell

Wie früher in diesem Kapitel unter den Design- und Architekturentscheidungen (Abschnitt 5.1) bereits erwähnt, soll die Schnittstelle nicht generisch sein. Sie soll möglichst die fachlichen Begriffe und Objekte aus der SYRIUS-Domäne verwenden. Das API-Modell definiert diese Objekte, welche zur Kommuniktation zwischen dem Autorisierungssystem und dem Aufrufer verwendet werden. Im DDD-Context ist dies die «Published Language».

Der Schnittstelle müssen drei Objekte übergeben werden, um eine Autorisierungsentscheidung treffen zu können:

- *BOIdentifier*: Identifikation des BO.
- *UserIdentifier*: Identifikation des Benutzers.
- *Operation*: Operation welche der Benutzer auf dem BO ausführen möchte.

```
1  BOIdentifier:
2    properties:
3      metaBoId:
4        description: 'Die MetaBoId des BOs welches autorisiert wird.'
5        type: integer
6        format: int64
7      boId:
8        description: 'Die BO-Id des BsO welches autorisiert wird.'
9        type: string
```

Auflistung 5.4: API-Modell: BOIdentifier (Auszug aus Swagger, YAML)

Der BOIdentifier setzt sich genau wie in SYRIUS aus der *MetaBOId* und der *BOId* zusammen (siehe Auflistung 5.4). Die *MetaBOId* definiert, um welchen Typ von BO es sich handelt. Dies kann zum Beispiel ein Partner oder Vertrag sein. Die *BOId* identifiziert hingegen das eindeutige Objekt des entsprechenden Typs.

Der UserIdentifier besteht aktuell lediglich aus einem Benutzernamen, wie aus Auflistung 5.5 hervorgeht.

```
UserIdentifier:
  properties:
    username:
      description: 'Key um den User eindeutig zu identifizieren.'
      type: string
```

Auflistung 5.5: API-Modell: UserIdentifier (Auszug aus Swagger, YAML)

Die *Operation* definiert in einem Request, welche Operation der Benutzer auf den übergebenen BO's ausführen möchte. In unserem Fall sind dies «READ» und «WRITE».

```
Operation:
  description:
    'Die Operation, welche der Benutzer auf einem BO ausführen möchte.'
  type: string
  enum:
    - READ
    - WRITE
```

Auflistung 5.6: API-Modell: Operation (Auszug aus Swagger, YAML)

Die Antwort welche das Autorisierungssystem dem Aufrufer liefert, muss eine Entscheidung beinhalten, ob der Benutzer auf das entsprechende BO zugreifen darf oder nicht. Diese Entscheidung ist über das Objekt *Decision* abgebildet. Auflistung 5.7 zeigt, welche Ausprägungen eine *Decision* annehmen kann.

```
AuthorizationDecision:
  type: string
  enum:
    - PERMIT
    - DENY
    - INDETERMINATE
    - NOTAPPLICABLE
```

Auflistung 5.7: API-Modell: Decision (Auszug aus Swagger, YAML)

Die wichtigen Ausprägungen sind hier «PERMIT» und «DENY», welche angeben, ob der Benutzer auf dem BO die gewünschte Operation ausführen darf. Die *Decision* «INDETERMINATE» wird ausgelöst, wenn der *Policy Decision Point (PDP)* nicht genügend Informationen bzw. Regeln zur Verfügung hat, um die angefragte Entscheidung zu treffen. «NOTAPPLICABLE» ist die *Decision* welche auf einen Request folgt, welcher semantisch nicht korrekt ist und aus diesem Grund gar nicht erst verarbeitet werden kann.

Optional kann die Resource eine *BOAuthorizationResponse* mit einer Liste von *Unauthorized Attributes* anreichern. Mit dieser Funktionalität wird der heutige Attribut-Schutz gelöst. Die Response kann damit um eine Liste von BO-Attributen angereichert werden, welche der Benutzer nicht sehen darf. Es ist dann die Aufgabe der *aufrufenden Applikation*, diese Attribute vor dem Benutzer zu verbergen. Die Liste enthält *BOAttribute*-Objekte, welche sich durch den Namen des zu schützenden Attribut's auszeichnen, wie Auflistung 5.8 zeigt.

```
1  BOAttribute:
2    description: Objekt repräsentiert ein BO-Attribut und wird für die
3     Attributberechtigungen verwendet.
4    properties:
5      name:
6        type: string
7        description: Name des Attributes.
```

Auflistung 5.8: API-Modell: BOAttribute (Auszug aus Swagger, YAML)

Diese Objekte werden im *BOAuthorizationRequest* und der *BOAuthorizationResponse* verwendet.

```
1  BOAuthorizationRequest:
2    properties:
3      userIdentifier:
4        $ref: '#/definitions/UserIdentifier'
5      boIdentifiers:
6        type: "array"
7        items:
8          $ref: '#/definitions/BOIdentifier'
9      operation:
10       $ref: '#/definitions/Operation'
```

Auflistung 5.9: API-Modell: BOAuthorizationRequest (Auszug aus Swagger, YAML)

Ein *BOAuthorizationRequest* zeichnet sich durch einen *UserIdentifier*, mehrere *BOIdentifier's* und der *Operation* aus, wie Auflistung 5.9 verdeutlicht.

Die RESTful HTTP Resource liefert für jeden Request eine Liste von BOAuthorization-Response's zurück. Eine einzelne Response gehört immer zu einem BO.

```
1  BOAuthorizationResponse:
2    properties:
3      boIdentifier:
4        $ref: '#/definitions/BOIdentifier'
5      decision:
6        $ref: '#/definitions/AuthorizationDecision'
7      unauthorized-attributes:
8        type: "array"
9        items:
10         $ref: '#/definitions/BOAttribute'
```

Auflistung 5.10: API-Modell: BOAuthorizationResponse (Auszug aus Swagger, YAML)

Der *BOIdentifier* ist folglich auch ein Attribut der *BOAuthorizationResponse*, mit welchem der Aufrufer die Antwort wieder einem BO zuordnen kann.

5.3.2. RESTful HTTP Resource

Die Schnittstelle besteht aktuell aus einer einzigen Resource zur Autorisierung von BO's, wie Abbilung 5.11 zeigt.

```
basePath: '/authorization-decision-point'
paths:
  /bo:
    post:
      description: "Authorization Decision Point für BO's."
      parameters:
        - name: authorization-request
          in: body
          schema:
            $ref: '#/definitions/BOAuthorizationRequest'
      responses:
        200:
          description: Entscheidung konnte erfolgreich ermittelt werden.
          schema:
            type: "array"
            items:
              $ref: '#/definitions/BOAuthorizationResponse'
```

Auflistung 5.11: RESTful HTTP Resource (Vereinfachter Auszug aus Swagger, YAML)

Die RESTful HTTP Resource wird «Authorization Decision Point» genannt. Dies ist eine Abwandlung der allgemeinen Bezeichnung «Policy Decision Point (PDP)», welche in der Literatur zum Thema *Access Control* und *Authorization* häufig verwendet wird. Der «Policy Decision Point (PDP)» ist dabei die Komponente welche die Autorisierungsentscheidung trifft und der «Policy Enforcement Point (PEP)» die Applikation, welche den PDP aufruft und dessen Entscheidung umsetzen muss. Mit dem Wort «Authorization» soll in unserer Schnittstelle besser zum Ausdruck gebracht werden, dass es sich um eine Resource zur Autorisierung handelt.

Die Resource nimmt einen *BOAuthorizationRequest* entgegen, welcher mehrere BO's autorisieren kann und liefert mehrere *BOAuthorizationResponses* zurück.

5.3.3. Beispiel

Um die Funktionsweise der Schnittstelle besser zu veranschaulichen, wird nachfolgend ein Beispiel eines Request's und einer dazugehörigen Response dargestellt.

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

Auflistung 5.12: Beispiel: BOAuthorizationRequest (JSON)

In diesem Beispiel in Auflistung 5.12 möchte der SYRIUS-Benutzer «skapferer» (in LDAP-Verzeichnis definiert) auf die beiden BO's mit den BoId's *1234* und *5678* zugreifen, welche beide vom Typ Partner (MetaBoId: 3) sind. Die Operation «READ» gibt an, dass der Benutzer die beiden BO lediglich lesen möchte, ohne Mutationen darauf auszuführen.

Die in diesem Request verwendeten Objekte *BOIdentifier*, *UserIdentifier* und *Operation* wurden in den Auflistungen 5.4, 5.5 und 5.6 erklärt.

Eine mögliche Antwort auf diesen Request zeigt Auflistung 5.13. In diesem Fall wird der Lese-Zugriff auf das BO mit der BOId *1234* nicht erlaubt. Der Zugriff auf das BO mit der BOId *5678* wird zwar erlaubt, allerdings mit dem *Unauthorized Attribute* namens *Geburtsdatum*.

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

Auflistung 5.13: Beispiel: BOAuthorizationResponse (JSON)

Nachdem dieser Abschnitt die Schnittstellendefinition vorgestellt hat, geht der nächste Abschnitt nun auf die Mock-Implementation des Autorisierungssystems ein.

5.4. Prototyp Autorisierungssystem

Um die im vorigen Kapitel beschriebene Autorisierungsschnittstelle bereitzustellen, wurde ein Prototyp mit einer «Mock-Implementation» in Java erstellt. Die «Mock-Implementation» enthält statisch codierte Autorisierungsregeln für die definierten Use Cases in Kapitel 4.

5.4.1. Logische Sicht

Die Projektstruktur des Autorisierungsprototypen ist aufgeteilt in ein *API*-, *Server*- und ein *Client*-Projekt.

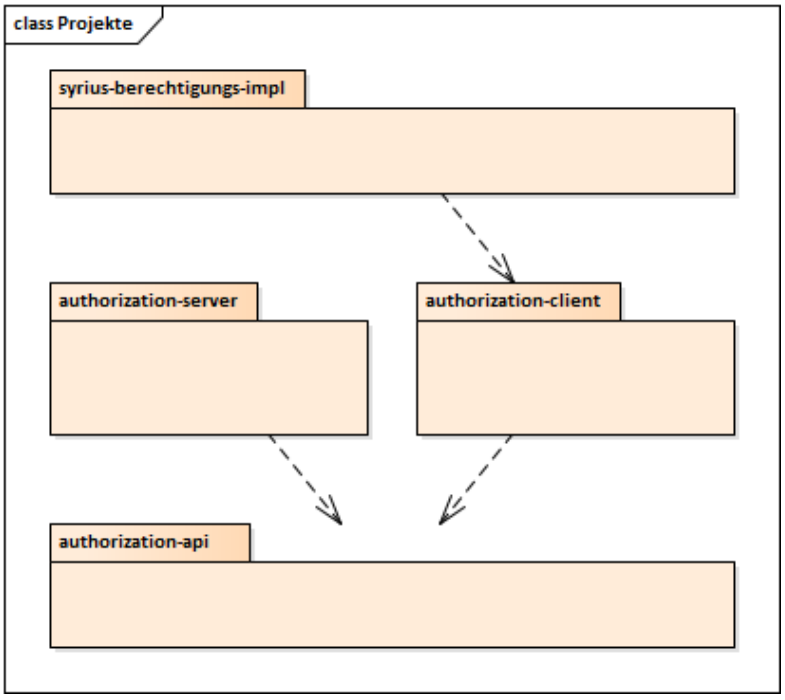

Abbildung 5.7.: Projekt-Struktur (UML Paketdiagramm)

Das Projekt *syrius-berechtigung-impl* ist das Projekt in SYRIUS, welches den Autorisierungsprototyp aufruft und war schon vor dieser Studienarbeit vorhanden. Es verwendet nun die Client-Library (*authorization-client*) für den Aufruf des Prototypen.

API Projekt

Das *API*-Projekt stellt das API-Modell zur Verfügung, welches sowohl vom *Server* zur Bereitstellung der Autorisierungsresource, als auch vom *Client* zum Aufrufen dieser Resource verwendet wird.

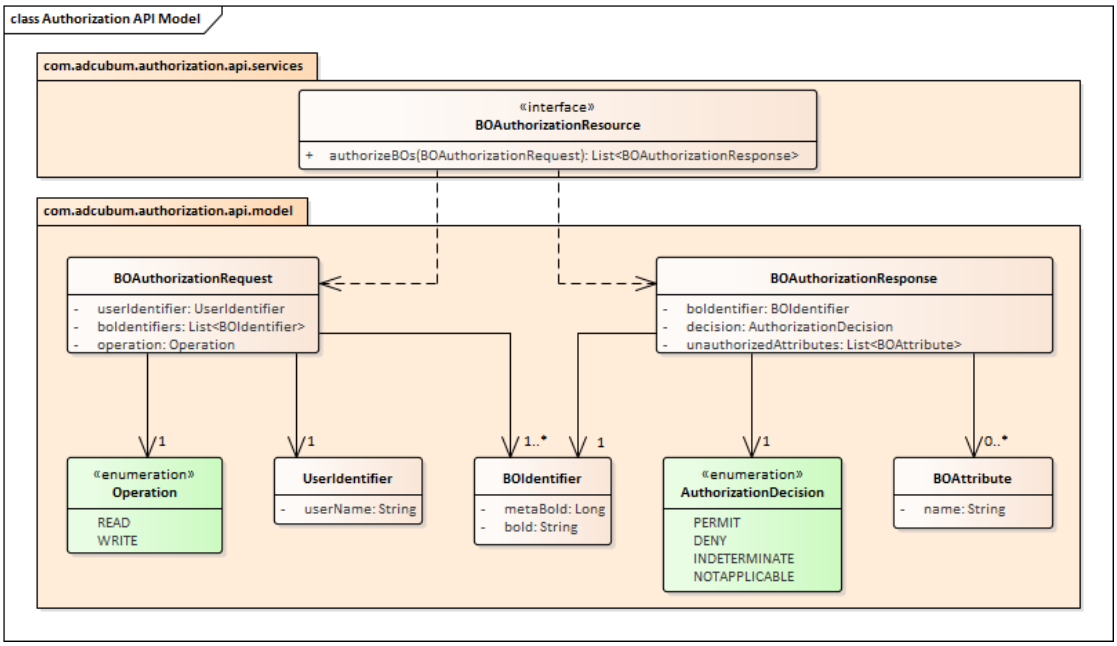

Abbildung 5.8.: API-Modell (UML Klassendiagramm)

Das Modell entspricht dem API-Modell, welches zuvor bereits in der Schnittstellendefinition eingeführt wurde. Die einzelnen Elemente des Modells werden darum an dieser Stelle nicht mehr erklärt. Das Interface *BOAuthorizationResource* spezifiziert die JAX-RS Resource (JAX-RS 2.0 [36]) und wird ebenfalls in *Client* und *Server* verwendet.

Client

Im Client-Projekt wird die Client-Library implementiert, welche in SYRIUS verwendet wird, um das Berechtigungssystem bzw. die RESTful HTTP Resource anzusprechen. Der Client wird als Java-Library (JAR-File) publiziert und in SYRIUS entsprechend als *Abhängigkeit* verwendet. Implementiert wurde der Client mit dem *Client Framework* von RESTEasy [23], einem JBoss [39] Projekt (Versionen, siehe Anhang C).

Server (Mock-Implementation)

Das Server-Projekt stellt eine Mock-Implementation der Autorisierungsresource zur Verfügung. Damit fungiert er im Prototyp als *Policy Decision Point (PDP)*. Auch dieser basiert auf dem RESTEasy [23] Projekt. Die Applikation läuft über einen eingebetteten Jetty [44] Servlet Container.

Die Mock-Implementation antwortet auf die Autorisierungsanfragen aufgrund statisch implementierter Regeln, welche gezielt für die Use Cases dieser Studienarbeit definiert wurden. In [38] wird eine UML-Notation vorgeschlagen, um Attribut-basierte Regeln (ABAC) zu dokumentieren.

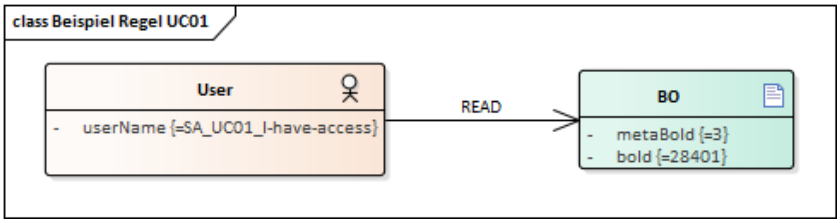

Abbildung 5.9.: Beispiel UML-Notation [38] für ABAC-basierte Regeln (Use Case 1)

Mittels dieser Notation zeigt Abbildung 5.9 beispielhaft eine Regel des ersten Use Cases, wie sie in der Mock-Implementation umgesetzt ist. Diese UML-Darstellung zeigt das *Subjekt* und *Objekt* mit deren Attributen. Ausserdem zeigt es, welche Ausprägungen die Attribute annehmen müssen, damit der Autorisierungsrequest mit «PERMIT» beantwortet wird.

```
1  BOIdentifier useCase01BO = new BOIdentifier(3, "28401");
2
3  private Response authorize(UserIdentifier userId,
4                             BOIdentifier boId,
5                             Operation operation) {
6      if (READ.equals(operation) && useCase01BO.equals(boId) &&
7          "SA_UC01_I-have-access".equals(userId)) {
8          return PERMIT;
9      } else {
10         return DENY;
11     }
12 }
```

Auflistung 5.14: Java Pseudo-Code zur Erklärung von Abbildung 5.9

Zur besseren Verständlichkeit der Notation zeigt Auflistung 5.14, wie eine statische «Pseudo»-Implementation dieser Regel in Java aussehen würde.

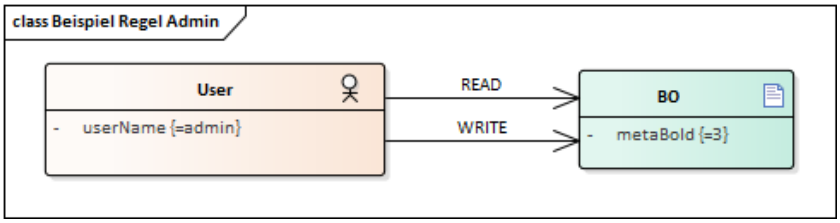

Abbildung 5.10.: Beispiel UML-Notation [38] für ABAC-basierte Regeln (Admin)

Ein weitere Beispiel-Regel zeigt Abbildung 5.10. Diese Regel bringt zum Ausdruck, dass der Benutzer mit dem Namen «admin» sowohl «READ»-, als auch «WRITE»-Rechte auf alle *Partner* BO's besitzt.

Eine detailliertere logische Sicht auf die Implementation des Prototypen befindet sich im Anhang C. Ebenfalls ist dort eine Tabelle mit den verwendeten Fremdbibliotheken inklusive allen Versionen und Lizenzen zu finden.

5.4.2. Deployment

Dadurch, dass das Autorisierungssystem «eigenständig deploybar» ist, werden verschiedene Deployment-Modelle ermöglicht. Das *authorization-server.jar* enthält die Klasse *ApplicationLauncher* mit einer Main-Methode, mit welcher die Applikation «standalone» über den integrierten Jetty [44] gestartet werden kann. Ein .war-File zu erstellen, um die Applikation auf einem anderen Web Server oder Servlet Container zu starten, wäre auch möglich.

Abbildung 5.11 zeigt eine mögliche Deployment-Variante, in welcher der Autorisierungsserver auf einem separaten *Server* läuft. SYRIUS und Elasticsearch [7] laufen ebenfalls auf eigenen Servern und greifen über die Autorisierungsschnittstelle (RESTful HTTP) auf den Autorisierungsserver zu.

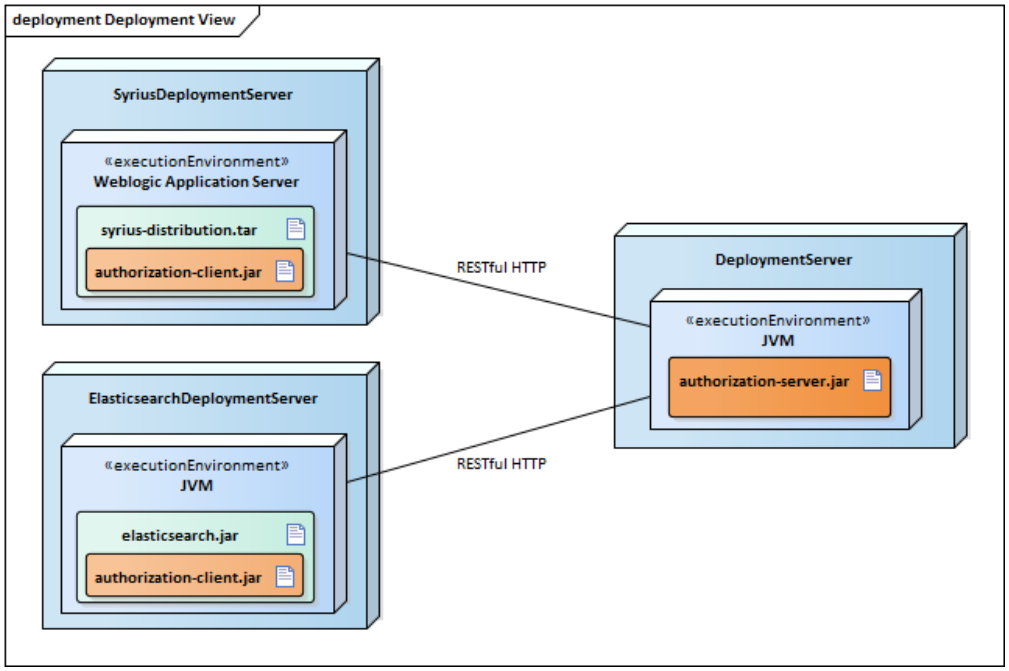

Abbildung 5.11.: Deployment Variante (UML Deployment-Diagramm)

Wie in Kapitel 4 bereits erwähnt, ist die Analyse der Performance nicht Teil dieser Studienarbeit. Die Frage ob dieses Deployment-Modell in der Praxis den Performance-Ansprüchen an die Autorisierungslösung genügt, wird hier deshalb nicht berücksichtigt. Es ist durchaus denkbar, dass aus Performancegründen ein Deployment auf dem selben Server oder sogar dem selben Application Server bevorzugt würde.

Eine weitere Deploymentvariante, in welcher das Autorisierungssystem auf dem selben Server wie SYRIUS läuft, befindet sich in Anhang C.

Nachdem in diesem Kapitel die wichtigsten Design- und Implementationsdetails beschrieben wurden, geht das nächste Kapitel nun auf die Schlussfolgerungen aus dieser Studienarbeit ein. Es bewertet ausserdem die Erfüllung der Anforderungen aus Kapitel 4 und wirft einen Blick auf zukünftige Arbeiten.

6. Ergebnisdiskussion

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

6.1. Bewertung der Anforderungen

In diesem Abschnitt werden die Anforderungen aus Kapitel 4 noch einmal aufgelistet und deren Erfüllungsgrad bewertet.

6.1.1. Funktionale Anforderungen

Die Anforderungen an die Use Cases konnten mit dem Prototypen erfüllt werden. Im Systemtest (siehe Anhang D) konnten alle Szenarien erfolgreich durchgeführt werden. Details dazu können dem Anhang D entnommen werden.

6.1.2. Nichtfunktionale Anforderungen

Die folgende Tabelle bewertet die nichtfunktionalen Anforderungen aufgrund des entwickelten Prototypen.

Tabelle 6.1.: Bewertung nichtfunkionaler Anforderungen

Sicherheit	Der Prototyp hat keine verschlüsselte Verbindung verwendet. Es wurde über HTTP kommuniziert, da in den Implementations-Iterationen die Zeit fehlte das Setup einer HTTPS-Verbindung durchzuführen. Diese NFA ist daher **nicht erfüllt**. Die eingesetzten Frameworks wie RESTEasy [23] und der integrierte Web Server Jetty [44] lassen eine verschlüsselte Verbindung jedoch zu. Damit führt dieses Resultat zu keinem Verlust der Aussagekraft der Projektergebnisse.

Performance	Diese NFA hat gefordert, dass bei Design-Entscheiden und der Auswahl der Schnittstellentechnologie darauf geachtet werden soll, dass später eine gute Performance erreicht werden kann. Die Messung der Performance des Prototypen war nicht Teil der NFA. Werden die Design- und Architekturentscheidungen in Kapitel 5 betrachtet, können keine Entscheidungen entdeckt werden, welche performanceoptimierende Massnahmen verhindern würden. Caching-Mechanismen sind mit der aktuellen Architektur möglich, allerdings wurde die Problematik während dieser Studienarbeit niedrig priorisiert. Binäre Abhängigkeiten wurden wie gefordert vermieden. Aus diesem Grund wird diese NFA als **teilweise erfüllt** betrachtet.
Lizenzen	Die Richtlinie des Industriepartners bezüglich der Verwendung von Fremdbibliotheken [1] wurde geprüft. Alle verwendeten Fremdbibliotheken wurden im Anhang C dokumentiert und der Richtlinie [1] gegenübergestellt. Diese NFA wurde **erfüllt**.
Logging / Audit Trail	Sämtliche Autorisierungsrequests werden per Log4J geloggt. Der Log-Eintrag enthält die Identität des Benutzers, den Identifier des Objekts, die Operation, welche der Benutzer auf dem Objekt ausführen möchte, sowie die vom Autorisierungssystem getroffene Entscheidung (PERMIT/DENY). Dies geht auch aus den Ausführungen des Systemtest (siehe Anhang D) hervor. Diese NFA wurde daher ebenfalls **erfüllt**.
Interoperabilität der Schnittstelle	Die Schnittstelle basiert auf RESTful HTTP. Es wurde keine Schnittstellentechnologie verwendet, welche von SYRIUS oder einem anderen Umsystem nicht angesprochen werden könnte. Die Interoperabilität der Schnittstelle ist gewährleistet und die NFA damit **erfüllt**.
Einfachheit der Schnittstelle	Zwar wurde in der Schnittstelle die Fachlichkeit von SYRIUS abgebildet und darauf geachtet, dass die Schnittstelle für einen SYRIUS-Entwickler einfach verständlich ist, jedoch wurde keine Verifikation über einen *User Test* durchgeführt, um zu überprüfen, ob ein Entwickler die Schnittstelle in der vorgegebenen Zeit versteht. Aus diesem Grund ist diese NFA nur **teilweise erfüllt**.

Externes System	Die Anforderung, dass der Autorisierungsprototyp eine eigenständige Komponente sein soll welche eigenständig «deployed» werden kann, ist **erfüllt**. Es wurde keine Autorisierungslogik innerhalb von SYRIUS implementiert, abgesehen vom Aufruf des Prototypen. Dadurch sind wie gefordert verschiedene Deployment-Modelle möglich, in welchen das Autorisierungssystem nicht auf dem gleichen *Server* wie SYRIUS betrieben werden muss.

6.2. Resultate dieser Studienarbeit

Die Ziele dieser Studienarbeit und deren Liefergegenstände wurden in der Aufgabenstellung (siehe Anhang E) festgelegt. Dieser Abschnitt geht auf die konkreten Resultate der Arbeit ein und bewertet die Zielerreichung mit Hilfe der in der Aufgabenstellung festgelegten Ziele.

6.2.1. Redesign der Persistenzschicht

Eine wichtige Arbeit war die Durchführung des Redesign in SYRIUS, mit dem Ziel festzustellen, an welchen Stellen in der Applikation das Autorisierungssystem aufgerufen werden muss, um die heutige SQL-Lösung zu ersetzen. Zur Umsetzung des Prototypen wurden drei wichtige Use Cases ausgewählt. Für diese Use Cases konnten die entsprechenden Stellen im Source-Code identifiziert werden. Kapitel 5 hat dies bereits ausführlich dokumentiert. Wie ebenfalls schon erwähnt wurde, musste während der Umsetzung allerdings festgestellt werden, dass aufgrund der aktuellen Gegebenheiten im Persistenzframework nicht abschliessend festgehalten werden kann, ob mit diesen Use Cases tatsächlich alle Code-Stellen identifiziert wurden. Es besteht die Möglichkeit, dass weitere Use Cases auch weitere Stellen im Persistenzframework hervorgebracht hätten, an welchen ebenfalls noch auf die Datenbank zugegriffen wird.

Auch wenn die ursprüngliche Fragestellung nicht abschliessend beantwortet werden kann und das Ziel damit nicht vollumfänglich erfüllt wurde, konnte der Industriepartner aus der Arbeit wertvolle Schlussfolgerungen ziehen. Es hat sich gezeigt, dass eine Implementation einer solchen Autorisierungslösung nur in Verbindung mit entsprechenden Anpassungen in der Persistenzschicht umsetzbar ist. Diese Erkenntnis ist wichtig für Planung zukünftiger Arbeiten.

6.2.2. Schnittstellendefinition

Die Analyse und Nachforschungen zum ABAC-Paradigma haben Hinweise geliefert, wie die zukünftige Autorisierungsschnittstelle aussehen könnte. Aus dieser Studienarbeit resultierte eine entsprechende Schnittstellendefinition. Nicht nur das Wissen, welche Informationen dem Autorisierungssystem in Zukunft übergeben werden sollen, ist ein Resultat dieser Studienarbeit. Auch die Design-Entscheidung, dass die Schnittstelle die Fachlichkeit aus SYRIUS wiederspiegeln soll und keine generische ABAC-Schnittstelle entwickelt wird, ist ein wichtiges Resultat welches zusammen mit dem Industriepartner erarbeitet werden konnte.

Dieser Entwurf der Schnittstelle bietet nun die Grundlage für die folgenden Arbeiten, in welchen überprüft werden soll, wie das Autorisierungssystem konkret implementiert werden kann.

6.2.3. Mock-System

Die Aufgabenstellung hat ausserdem ein Mock-System gefordert, welches im Rahmen dieser Studiearbeit entwickelt wurde. Die Mock-Implementation hat die Basis gebildet, um die Schnittstellendefinition und das Redesign in SYRIUS testen zu können. Es hat gezeigt, dass das angestrebte Konzept des externen Autorisierungssystems im SYRIUS-Umfeld umsetzbar ist.

6.2.4. Aufbereitung Architekturwissen

Ein weiteres Ziel der Studienarbeit war die Aufbereitung des erarbeiteten Architekturwissens. Das «Architectural Refactoring» Template von Prof. Dr. Zimmermann wurde dabei verwendet, um das wiederverwendbare Refactoring «From RBAC to ABAC» zu erarbeiten. Durch die Verwendung konnte Herrn Zimmermann während der Studienarbeit entsprechendes Feedback zum Template gegeben werden.

6.2.5. Evaluation von «Service Cutter» und «Structurizr»

Ausserdem sollte die Eignung der Open Source Werkzeuge «Service Cutter», inkl. dessen Kriterienkatalog, und Structurizr evaluiert werden. Auch wenn die Tools im Rahmen der Studienarbeit nicht direkt eingesetzt werden konnten, sind sie für den Industriepartner, in einem diesem Projekt übergeordneten Kontext, sehr interessant und könnten im Rahmen der strategischen Wartung von SYRIUS ihren Einsatz finden. Die detaillierten Resultate der Evaluationen innerhalb dieser Studienarbeit befinden sich in Anhang A.

6.3. Ausblick

Im Anschluss an diese Studienarbeit folgt eine Bachelorarbeit, welche erneut mit Adcubum als Industriepartner durchgeführt wird. Die mit dieser Studienarbeit gelegten Grundlagen sollen in der Bachelorarbeit verwendet werden. Es soll nun genauer betrachtet werden, wie und ob die heutigen Rollen in SYRIUS in ein ABAC-basiertes Autorisierungssystem migriert werden können. Welche Daten das Autorisierungssystem benötigt, um die Autorisierungsentscheidungen treffen zu können, wurde in dieser Studienarbeit nicht berücksichtigt. Es stellt sich die Frage, wie das Attribut-Repository des Autorisierungssystem mit den nötigen Attributen der SYRIUS-Objekte versorgt werden kann.

Das Hauptziel der Bachelorarbeit ist die Erarbeitung der Anforderungen an das zukünfige Autorisierungssystem, damit später vom Industriepartner ein «Make or Buy»-Entscheid getroffen werden kann. Dafür muss der Migrationsaspekt der Berechtigungskonfiguration genauer beleuchtet werden.

A. Evaluation von Tools

Ein Ziel dieser Studienarbeit war die Evaluation des Tools Service Cutter [14] inklusive dessen Kriterienkatalog. Optional sollte auch Structurizr [26] in Betracht gezogen werden. Dieser Anhang dokumentiert die Resultate dieser Evaluationen.

A.1. Service Cutter

A.1.1. Einleitung

Monolithische Systeme in kleinere *Module* oder *Services* aufzuteilen, ist ein häufiges Problem in der Softwareentwicklung. Service Cutter [14, 29, 27] ist ein Tool welches versucht, auf eine strukturierte Art und Weise, Service-Schnitte vorzuschlagen. Damit soll eine Diskussionsgrundlage geschaffen werden, wie eine Software in kleinere Teilsysteme aufgeteilt werden könnte. Service Cutter verwendet *Graph Clustering Algorithmen* um eine solche Aufteilung zu finden.

A.1.2. Vorgehen

Das Vorgehen wird im Tutorial [28] von Service Cutter beschrieben. Entscheidend ist der Input, welchen der Service Cutter braucht, um die bestehende Software zu analysieren, und Service-Schnitte vorzuschlagen. Dieser Input ist ein Domain Model in Form eines ERD, welches in einem spezifischen JSON-Format bereitgestellt werden muss. Beispiele, wie dieses Format aussehen muss, können im Github-Repository [27] von Service Cutter gefunden werden.

A.1.3. Kriterien

Der Service Cutter [14] definiert sogenannte «Coupling Criteria» mit welchen eine gute Trennung (Englisch: «Decomposition») der Services erreicht werden soll.

Dafür wurde in der Bachelorarbeit in welcher der Service Cutter entwickelt wurde ein «Coupling Criteria Catalog» erarbeitet, welcher die Kriterien aus folgenden Perspektiven betrachtet:

- Domain
- Quality
- Physical
- Security

Im Rahmen dieser Studienarbeit wurden die Kriterien der Perspektive «Security» betrachtet und geprüft inwiefern diese Kriterien zu unserem Vorhaben, die Autorisierung in einem eigenen «Bounded Context» zu lösen, passen. Folgende Tabelle zeigt die Beschreibungen der drei Security-Kriterien des Service Cutter.

Tabelle A.1.: Service Cutter: «Security» Coupling Criteria

Name	**Beschreibung**
CC-14 Security Contextuality	A security role is allowed to see or process a group of nanoentities. Mixing security contexts in one service complicates authentication and authorization implementations.
CC-15 Security Criticality	Criticality of an nanoentity in case of data loss or a privacy violation. Represents the reputational or financial damage when the information is disclosed to unauthorized parties. As high security criticality comes at a cost, nanoentities classified with different characteristics should not be composed in the same service.
CC-16 Security Constraint	Groups of nanoentities are semantically related but must not reside in the same service in order to satisfy information security requirements. This restriction can be established by an external party such as a certification authority or an internal design team.

Damit bieter der Service Cutter Indikatoren welche darauf hinweisen wie fachliche Services einer Applikation aufgrund von sicherheitsrelevanten Kriterien getrennt werden sollen. Die Kriterien gehen dabei wenig darauf ein, inwiefern die Implementation der «Security» von der Businesslogik getrennt werden soll. Vielleicht könnten hier zusätzliche Kriterien entwickelt werden welche darauf eingehen wie «Cross-Cutting Concerns», wie zum Beispiel Autorisierung, von den eigentlichen Business-Services getrennt werden sollen. Diese «Supporting Domains» sind für alle Services einer Applikation relevant, und trotzdem soll die Implementation sauber getrennt werden.

A.1.4. Bewertung

Die Security Kriterien des Service Cutter könnten aus Sicht dieser Studienarbeit eventuell erweitert werden, um besser darauf einzugehen, dass das Mischen von Businesslogik und der Implementation sicherheitsrelevanter Themen wie Authentisierung und Autorisierung (generell «Cross-Cutting Conterns») zu Problemen führen kann.

Der Service Cutter wurde am Anfang dieses Projekts als mögliches Tool zur Unterstützung der Studienarbeit vorgeschlagen. Da das Redesign in SYRIUS aber nur ein Teil des Umfangs war und dort bereits klar war dass die Autorisierung einen eigenen «Microservice» bilden soll, wurde der Nutzen innerhalb dieser Studienarbeit für zu klein eingeschätzt. Der Aufwand um die Daten für den Service Cutter aufzubereiten (Domain Model) hätte den Rahmen evtl. auch gesprängt. Im übergreifenden Kontext der strategischen Wartung von SYRIUS bei Adcubum könnte das Tool aber durchaus einen Nutzen bringen.

A.2. Evaluation Structurizr

A.2.1. Einleitung

Structurizr [26] ist ein Tool um die Architektur einer Software auf Basis des Source-Codes zu visualisieren.

A.2.2. Vorgehen

Mit Structurizr [26] kann die Architektur eines bestehenden Systems entweder manuell oder durch Analyse der Codebasis erstellt werden. Es wird ein Programm in Java implementiert welches das Modell exportiert. Anschliessend kann dieses Modell mit der Structurizr API in die Cloud (SaaS) geladen werden um dort Diagramme des Systems zu erstellen.

A.2.3. Extrahieren der Informationen aus der Codebasis

Es gibt einige bereits implementierte Suchstrategien für Komponenten.

Typ-basierte Suche

Mit der *TypeBasedComponentFinderStrategy* können Komponenten aufgrund Namen, Interface-Implementationen, Regex-Suchen oder Annotationen gefunden werden.

Spring-Komponenten

Mit dem *SpringComponentFinderStrategy* können Komponenten aufgrund von Spring-Annotationen wie @RestController, @Component, @Service und @Repository gefunden werden. Setzt natürlich vorraus dass man das Spring-Framework einsetzt.

JEE-Komponenten

Ähnlich wie bei Spring, könnten auch JEE-Annotationen zur Analyse verwendet werden.

Java-Doc

Eine weitere Möglichkeit bietet die Analyse von Java-Doc Kommentaren. Hier geht es allerdings nicht darum Komponenten zu finden, sondern Komponenten mit Informationen aus Java-Doc Kommentaren zu Komponenten hinzuzufügen.

Manuelle Implementation

Die Strategie zur Identifikation kann auch manuell implementiert werden, wenn zum Beispiel Annotationen eigener Frameworks verwendet werden sollen. Dies wäre innerhalb von SYRIUS vermutlich nötig, um ein gutes Modell erstellen zu können.

A.2.4. Bewertung

Innerhalb dieser Studienarbeit wäre der Aufwand welcher in die Verwendung von Structurizr hätte gesteckt werden müssen, in Anbetracht des Nutzens, zu gross gewesen. Der vom Redesign betroffene Sourcecode-Teil in der Persistenzschicht war zu überschaubar (wenige Klassen) als dass sich der Einsatz des Tools direkt gelohnt hätte. Ausserdem benötigt der Einsatz des Tools ein gewisses Know-How darüber wie die Komponenten der Software sinnvoll identifiziert werden können. Die Annotationen und Interfaces der Frameworks in SYRIUS hätten zuerst genau analysiert werden müssen. Der Aufwand die Komponenten innerhalb von SYRIUS zu identifizieren hätte den den Rahmen des zeitlich möglichen gesprengt. Die Verwendung von Structurizr wäre aber wie auch der Service Cutter in einem projektübergreifenden Kontext im Rahmen der strategischen Wartung von SYRIUS durchaus interessant.

B. Anforderungsspezifikation im Detail

B.1. UC01: Partner-BO schützen

B.1.1. Use Case Brief

Ein BO, in diesem Use Case der Partner, muss geschützt werden. Ein Benutzer welcher keine Berechtigung für den Zugriff auf ein BO hat, erhält in der Partner-Suchmaske kein Resultat.

B.1.2. Use Case Fully Dressed

Name	Partner-BO schützen
Primary Actor	Sachbearbeiter
Stakeholders and Interests	Versicherer und Kunde sind daran interessiert dass Kundendaten vor unberechtigten Zugriffen geschützt werden.
Preconditions	User ist eingeloggt, Berechtigungen sind parametriert.
Postconditions	User sieht Partner nicht
Haupt-Szenario mit Benutzer welcher keine Berechtigung für Zugriff auf den Partner hat	1. User öffnet Partnersuche 2. User sucht Partner X 3. System liefert kein Resultat, obwohl Partner existiert
Alternatives Szenario mit Benutzer welcher die Berechtigung besitzt.	Zur Validierung des Use Cases wird ein zweiter User Y erstellt welcher die Berechtigung für den Zugriff auf das entsprechende BO besitzt. In diesem Fall ändert sich der Schritt 3) des Szenarios. Der Benutzer Y findet den Partner und kann diesen anzeigen.

B.1.3. Szenario

Für den UseCase UC01 wurden folgende Benutzer angelegt:

SA_UC01_I-have-access	Hat Zugriff auf den zu schützenden Partner.
SA_UC01_I-dont-have-access	Hat keinen Zugriff auf den zu schützenden Partner.

Partner

Es wurde ein Partner mit einer eigens für diesen Use Case erstellten Partnerschutz-Definition angelegt.

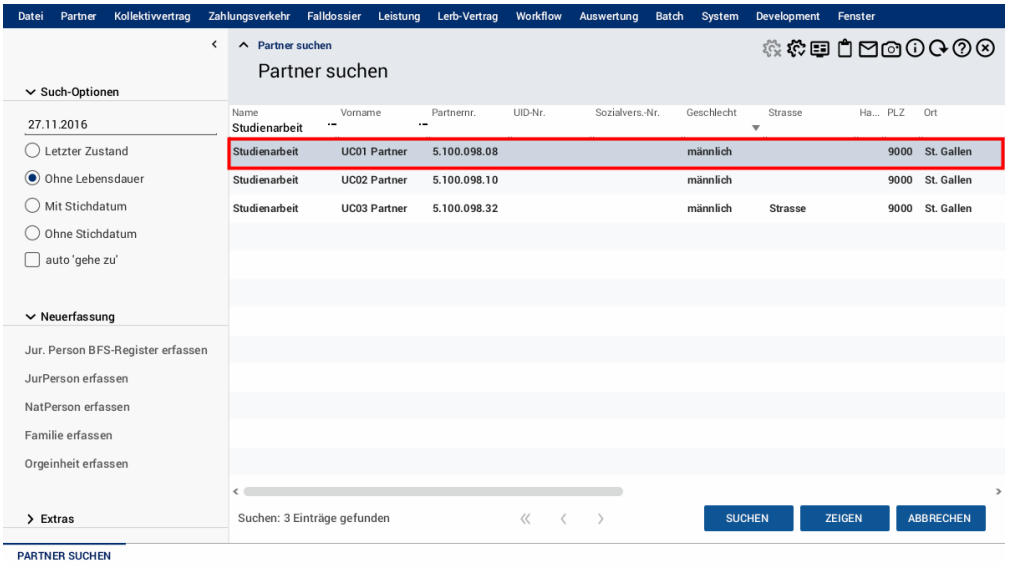

Abbildung B.1.: Partnersuche: Partner UC01

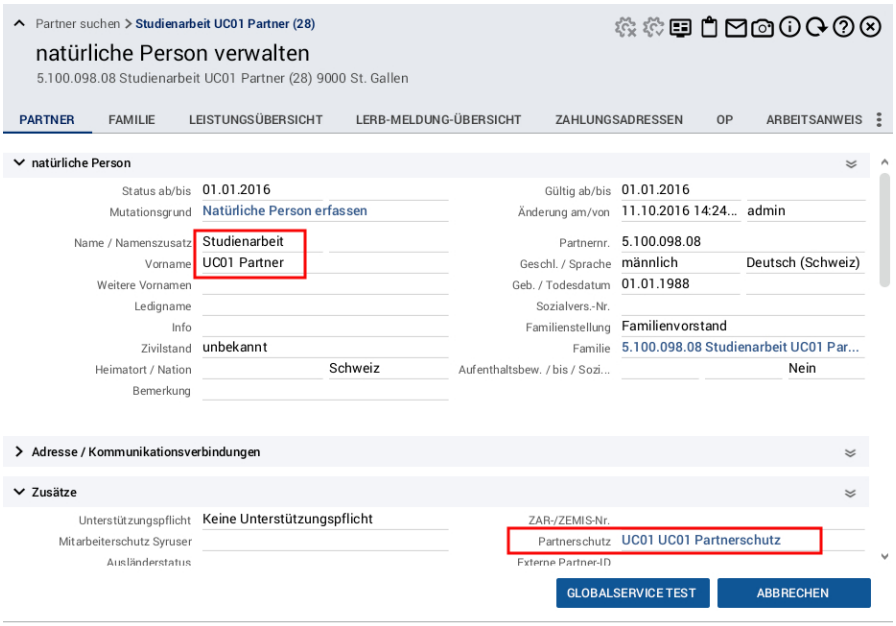

Abbildung B.2.: Partnerübersicht: Partner UC01

Partnersuche mit Benutzer SA_UC01_I-have-access

Der Benutzer SA_UC01_I-have-access hat die Berechtigung den Partner zu sehen:

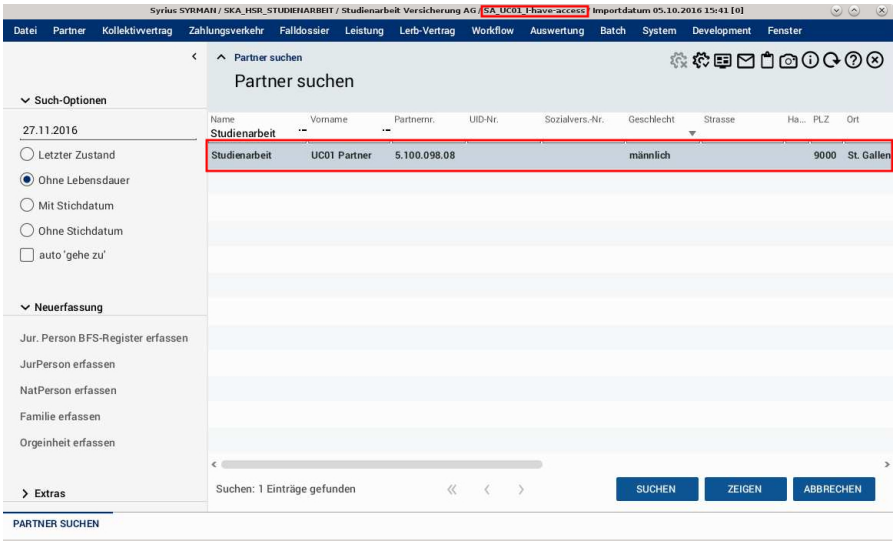

Abbildung B.3.: Partnersuche mit User SA_UC01_I-have-access

Partnersuche mit Benutzer SA_UC01_I-dont-have-access

Der Benutzer SA_UC01_I-dont-have-access kann den Partner hingegen nicht finden:

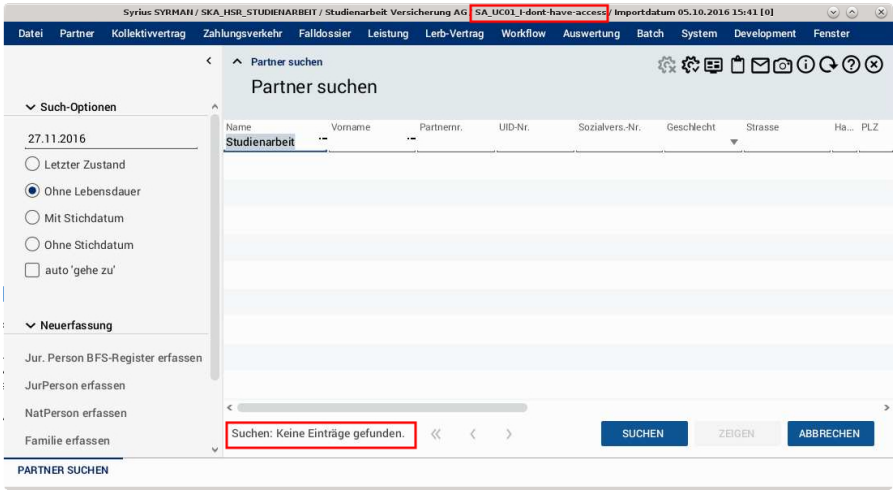

Abbildung B.4.: Partnersuche mit User SA_UC01_I-dont-have-access

B.2. UC02: Partner BO-Attribute schützen (Partnerübersicht)

B.2.1. Use Case Brief

Auf der Partnerübersicht sieht der Sachbearbeiter nur bestimmte Attribute.

B.2.2. Use Case Fully Dressed

Name	Partner BO-Attribute schützen (Partnerübersicht)
Primary Actor	Sachbearbeiter
Stakeholders and Interests	Versicherer und Kunde sind daran interessiert dass bestimmte Attribute der Kundendaten nicht für jeden Sachbearbeiter sichtbar sind.
Preconditions	Sachbearbeiter ist eingeloggt, Berechtigungen sind parametriert, Sachbearbeiter hat die Berechtigung um das BO zu sehen

Postconditions	Sachbearbeiter kann das Partner-BO zwar sehen, jedoch nicht alle BO-Attribute.
Haupt-Szenario bei welchem der Sachbearbeiter nur beschränkte Berechtigungen auf dem Partner-BO hat. Er darf nicht alle Attribute sehen.	1. Sachbearbeiter öffnet Partnersuche 2. Sachbearbeiter sucht Partner X und kann diesen auch finden bzw. die Partnerübersicht öffnen 3. System liefert zeigt dem Sachbearbeiter nur bestimmte Attribute des BO's an.
Alternatives Szenario bei welchem der Sachbearbeiter alle Attribute des Partners sehen darf.	In diesem alternativen Szenario ändert sich Schritt 3 des Haupt-Szenarios und der Benutzer sieht alle Attribute des Partners.

B.2.3. Szenario

Für den UseCase UC02 wurden folgende Benutzer angelegt:

SA_UC02-I-can-see-all-attributes	Hat Zugriff auf alle BO-Attribute des Partners.
SA_UC02_I-cannot-see-all-attributes	Hat nur auf bestimmte BO-Attribute des zu schützenden Partner-BO's Zugriff.

Partner

Es wurde ein Partner mit einer eigens für diesen Use Case erstellten Partnerschutz-Definition angelegt.

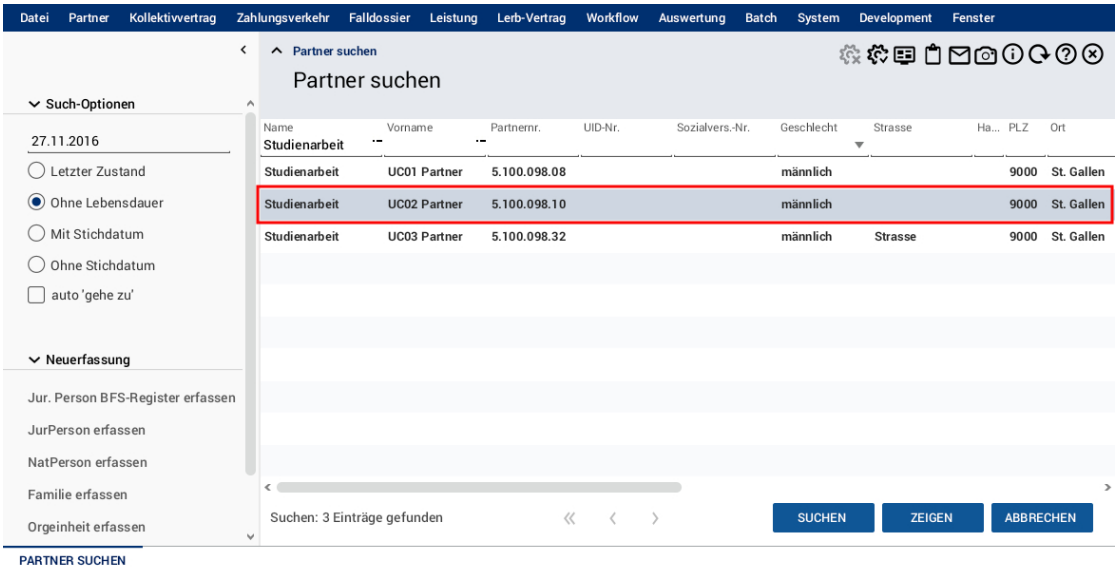

Abbildung B.5.: Partnersuche: Partner UC02

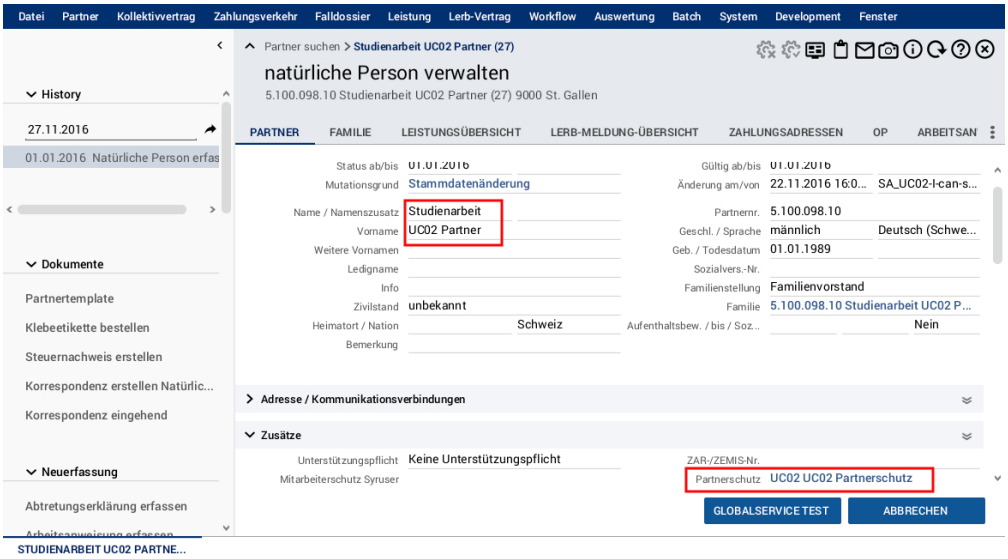

Abbildung B.6.: Partnerübersicht: Partner UC02

Partnersuche mit Benutzer SA_UC02-I-can-see-all-attributes

Der Benutzer SA_UC02-I-can-see-all-attributes hat die Berechtigung alle Attribute des Partners zu sehen:

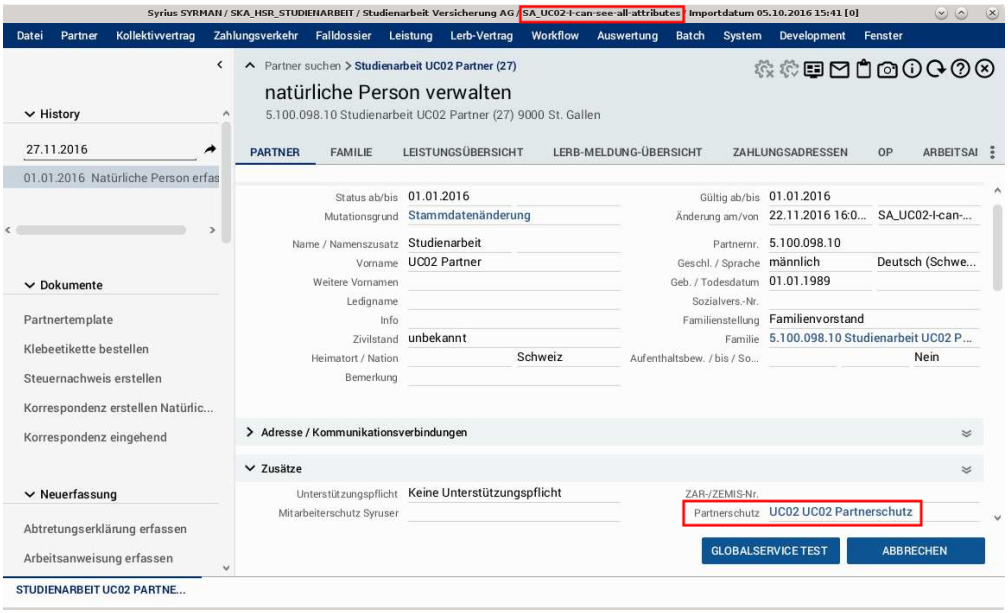

Abbildung B.7.: Partnerübersicht mit User SA_UC02-I-can-see-all-attributes

Partnersuche mit Benutzer SA_UC02_I-cannot-see-all-attributes

Der Benutzer SA_UC02_I-cannot-see-all-attributes darf hingegen nur bestimmte Attribute des Partners sehen. Alle anderen Attribute sind mit '#####' gekennzeichnet:

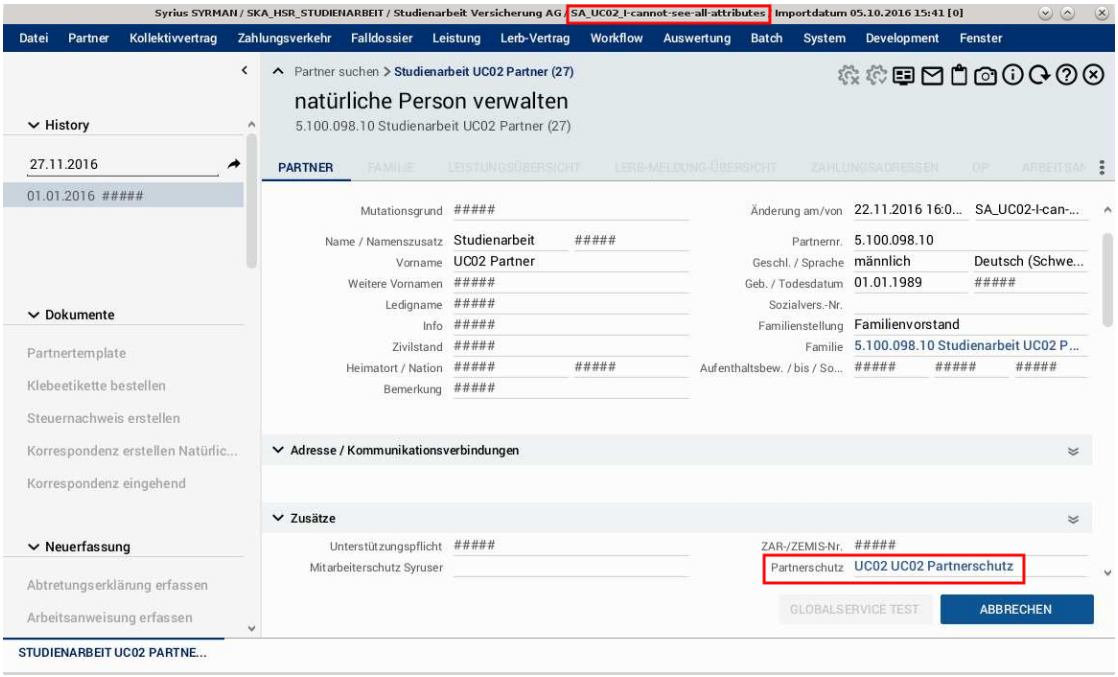

Abbildung B.8.: Partnerübersicht mit User SA_UC02_I-cannot-see-all-attributes

B.3. UC03: Partner-BO vor unbefugten Mutationen schützen (Adressmutation)

B.3.1. Use Case Brief

Ein Benutzer kann Lese-Zugriff auf ein BO besitzen, jedoch keinen Schreib-Zugriff. Hat der Benutzer keinen Schreibzugriff, muss der Aufruf des entsprechenden Update-Statements unterbunden werden.

Der Aufruf des Updates auf dem BO kann nicht über den UTC-Client erfolgen, da dieser für dieses Szenario die entsprechenden Tasks auf dem GUI sperrt, Damit wir überprüfen können ob die Persistenz-Schicht von SYRIUS einen Update auch blockiert, müssen wir einen Webservice-Aufruf (SOAP) machen.

B.3.2. Use Case Fully Dressed

Name	Partner-BO vor unbefugten Mutationen schützen (Adressmutation)
Primary Actor	Sachbearbeiter
Stakeholders and Interests	Versicherer und Kunde sind daran interessiert, dass bestimmte Sachbearbeiter keine Möglichkeit haben die Daten eines Kunden zu mutieren.
Preconditions	Sachbearbeiter ist eingeloggt, Berechtigungen sind parametriert, Sachbearbeiter hat die Berechtigung das BO zu sehen, jedoch keine Berechtigung es zu mutieren
Postconditions	Sachbearbeiter kann Partner-BO zwar sehen, jedoch keine Mutationen darauf ausführen.
Hauptszenario bei welchem der Sachbearbeiter keine Mutationsberechtigung auf dem Partner besitzt.	1. Unter dem Sachbearbeiter X wird ein Service-Aufruf (SOAP-Service) getätigt, welcher die Adresse des Partners mutieren soll. 2. Das System lässt die Operation nicht zu und gibt einen Fehler zurück.
Alternatives Szenario mit einem Sachbearbeiter Y welcher die Berechtigung besitzt den Partner zu mutieren.	In diesem Szenario ändert sich der Schritt 2) und das System gibt keinen Fehler sondern eine Erfolgsmeldung zurück. Die Mutation wurde durchgeführt.
Alternatives Szenario über UTC-Client anstatt SOAP-Service (Sachbearbeiter X)	1. User öffnet Partnersuche 2. User sucht den Partner und öffnet diesen. 3. Alle Mutations-Tasks sind auf dem UI deaktiviert.

B.3.3. Szenario

Für den UseCase UC03 wurden folgende Benutzer angelegt:

SA_UC03_I-can-read-and-write	Hat Lese- und Schreibzugriff auf dem Partner-BO.
SA_UC03_I-cannot-write	Hat nur Lesezugriff auf den Partner. Der Sachbearbeiter darf keine Mutationen vornehmen.

Partner

Es wurde ein Partner mit einer eigens für diesen Use Case erstellten Partnerschutz-Definition angelegt.

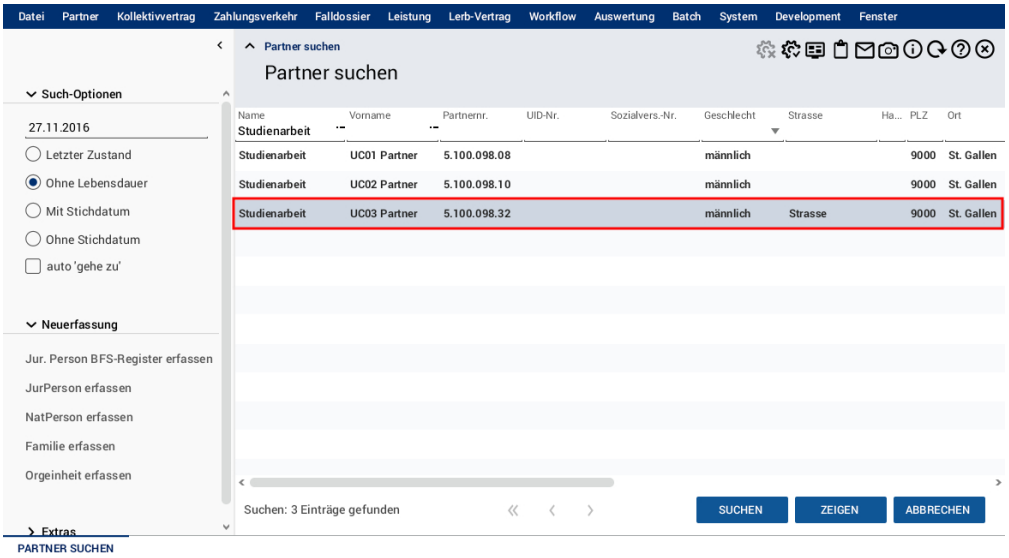

Abbildung B.9.: Partnersuche: Partner UC03

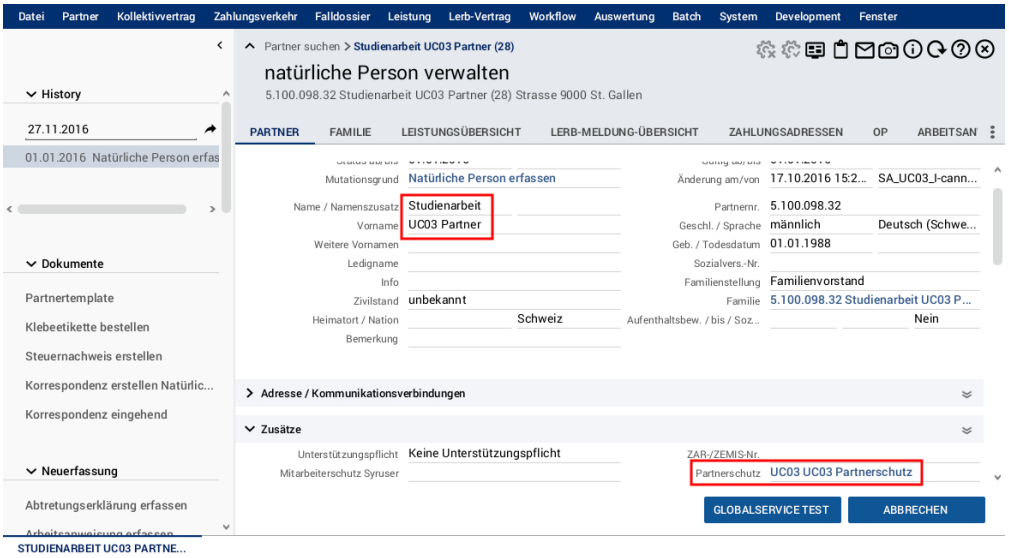

Abbildung B.10.: Partnerübersicht: Partner UC03

Ändern der Adresse mit Sachbearbeiter SA_UC03_I-can-read-and-write

Der Benutzer SA_UC03_I-can-read-and-write hat die Berechtigung den Partner zu ändern. Über den Task 'Adressänderung' kann er zum Beispiel die Strasse ändern:

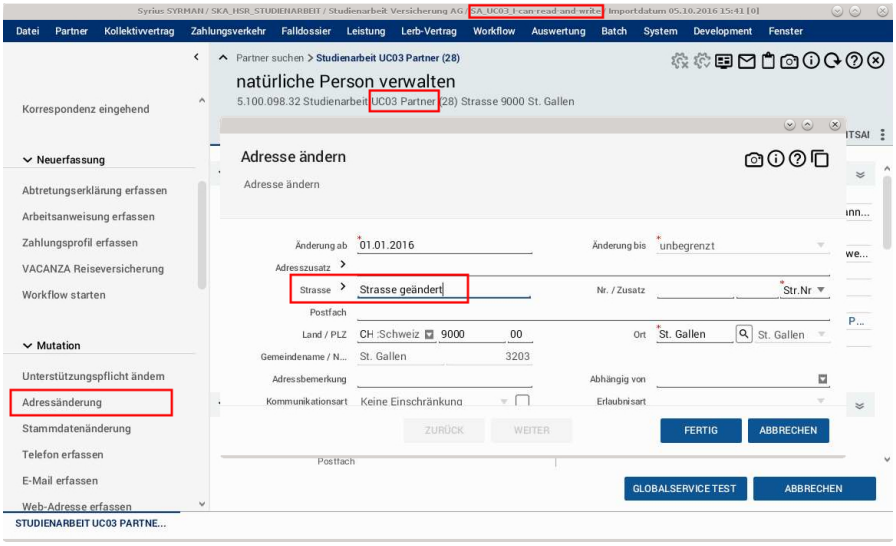

Abbildung B.11.: UC03: Partneradresse ändern

Durch abschliessen des Task, wird die Änderung persistiert:

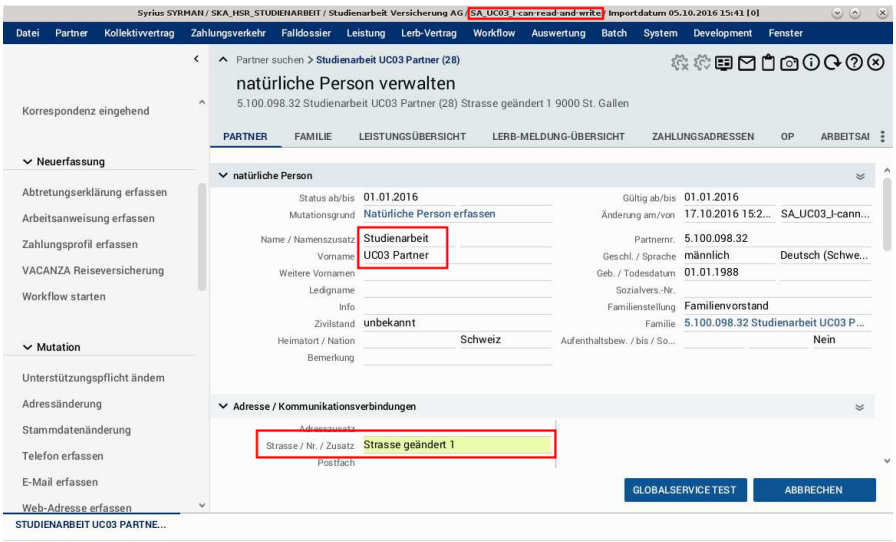

Abbildung B.12.: UC03: Partneradresse geändernt

Adressänderung mit Sachbearbeiter SA_UC03_I-cannot-write nicht möglich (GUI)

Der Sachbearbeiter SA_UC03_I-cannot-write hat hingegen die Berechtigung für Mutationen nicht. Er kann auf dem GUI keine Mutationstasks starten:

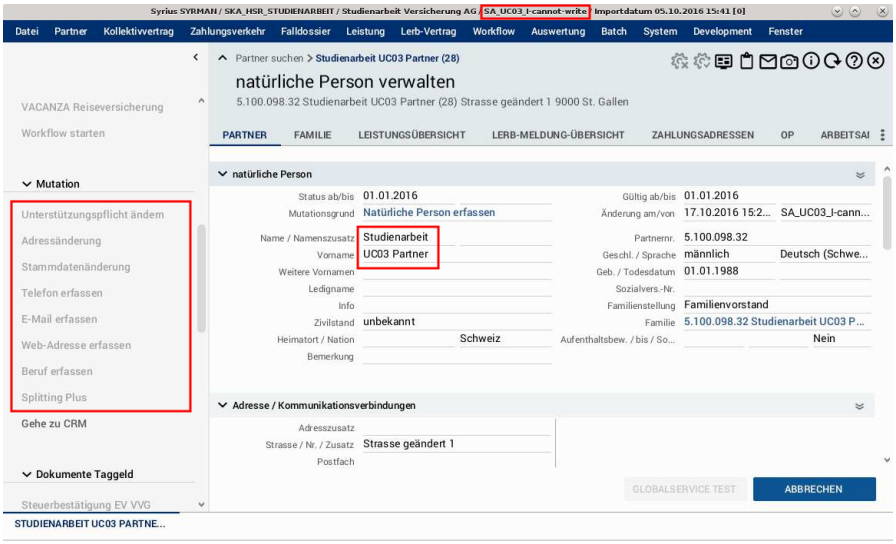

Abbildung B.13.: UC03: Tasks deaktiviert für SA_UC03_I-cannot-write

Adressänderung mit Sachbearbeiter SA_UC03_I-cannot-write nicht möglich (SOAP-Service)

Zur Verifikation ob die Persistenz-Schicht von SYRIUS einen Update-Aufruf auch wirklich verhindert, rufen wir einen SOAP-Service auf mit welchem eine Adressänderung ausgelöst werden soll.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <soapenv:Envelope>
3     <soapenv:Header>
4        <urn:SoapLoginInformation>
5           <Password>1234</Password>
6           <Username>SA_UC03_I-cannot-write</Username>
7        </urn:SoapLoginInformation>
8     </soapenv:Header>
9     <soapenv:Body>
10       <glob:replicatePartner>
11          <ns4:message>
12             <Adressen>
13                <AdressStates>
14                   <AdressOrder id="-36200" />
15                   <Hausnummer></Hausnummer>
16                   <Land>CH</Land>
17                   <Ort>St. Gallen</Ort>
18                   <PLZ>9000</PLZ>
19                   <PLZ_zusatz>00</PLZ_zusatz>
20                   <PlzId>5189</PlzId>
21                   <!-- Versuch, die Strasse zu ändern: -->
22                   <Strasse>Strasse ändern</Strasse>
23                </AdressStates>
24                <AdresseTyp id="-43800" />
25                <ExtAdresseId>UC03-Partner-Adresse</ExtAdresseId>
26             </Adressen>
27             <ExtPartnerId>UC03-Partner</ExtPartnerId>
28             <MessageID>SA_Test_213</MessageID>
29          </ns4:message>
30       </glob:replicatePartner>
31    </soapenv:Body>
32 </soapenv:Envelope>
```

Auflistung B.1: UC03: Adressmutation mit User SA_UC03_I-cannot-write über SOAP-Service

Über den SOAP-Request aus Auflistung B.1 wird versucht die Adresse des Partners zu ändern, wie auf Zeile 22 ersichtlich ist. Ebenfalls zeigt Auflistung B.1 auf Zeile 6 dass der Benutzer *SA_UC03_I-cannot-write* verwendet wird, um die Parteradresse zu ändern.

Wie dies nun zu erwarten ist, verweigert SYRIUS den Update des Partners. Auflistung B.2 zeigt die entsprechende Antwort von SYRIUS mit der Exception welche ausgelöst wurde.

```
<soapenv:Envelope>
   <soapenv:Body>
      <soapenv:Fault>
         <faultcode>soapenv:Server</faultcode>
         <faultstring>SYR-3291 Fehler: Keine Mutationsberechtigung für
                  ''Strasse   9000 St. Gallen'' (Id: 125581, Typ: Adresse,
                  Schutzobjekt: PartnerSchutzDef'UC03 UC03 Partnerschutz')
                  (ERROR_ID="1072335f-b190-458f-9c4d-e41d7ba31938")
         </faultstring>
         <detail>
               <alert>
                  <name>SYR-3291</name>
               </alert>
               <texts>
                  <text>Keine Mutationsberechtigung für
                        '%1' (Id: %2, Typ: %3, Schutzobjekt: %4)</text>
                  <Locale language="de" country="" variant=""/>
               </texts>
               <templateArguments>'Strasse   9000 St. Gallen'
               </templateArguments>
               <templateArguments>125581</templateArguments>
               <templateArguments>Adresse</templateArguments>
               <templateArguments>PartnerSchutzDef'UC03 UC03
                Partnerschutz'</templateArguments>
               <stacktrace>syrius.util.exception.ApplicationException:
                SYR-3291 Fehler: Keine Mutationsberechtigung für
                ''Strasse   9000 St. Gallen'' (Id: 125581, Typ:
                Adresse, Schutzobjekt: PartnerSchutzDef'UC03 UC03
                Partnerschutz')
                (ERROR_ID="1072335f-b190-458f-9c4d-e41d7ba31938")
                at syrius.modul_bl.service.soap.service.impl.SoapHandler
                <!-- ... (gekürtzt) ... -->
               </stacktrace>
         </detail>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

Auflistung B.2: UC03: Adressmutation mit User SA_UC03_I-cannot-write nicht erlaubt

C. Design und Implementation im Detail

Dieser Anhang beschreibt zusätzliche Details zum Design und der Implementation, welche im Kapitel 5 nicht, oder nicht in diesem Detaillierungsgrad, erwähnt wurden.

C.1. Logische Sicht

Die Logische Sicht auf den Autorisierungsprototypen und das in SYRIUS durchgeführte Redesign der Persistenzschicht wurden im Hauptbericht bereits erwähnt, sollen an dieser Stelle aber noch vertieft werden.

C.1.1. Autorisierungssystem (Prototyp)

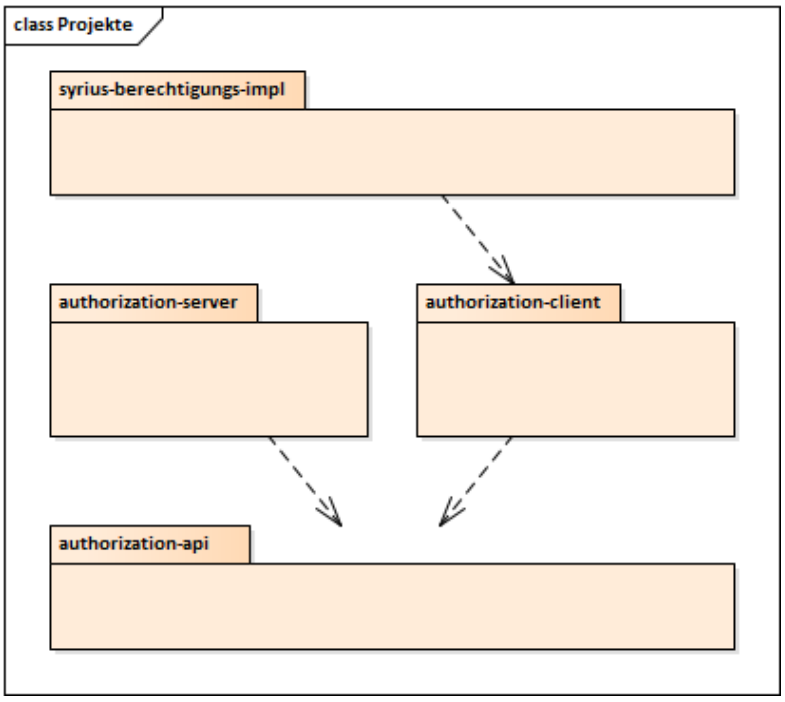

Abbildung C.1.: Projekt-Struktur (UML)

Die Projektstruktur des neuen Berechtigungssystem, wie in Abbildung C.1 gezeigt, ist aufgeteilt in ein *API*-, *Server*- und ein *Client*-Projekt. Das API-Projekt enthält die Definitionen (Modell und Interfaces) der Schnittstelle, welche sowohl vom Server sowie auch vom Client benötigt werden. Im Server-Projekt wird der Server implementiert, welcher die Autorisierungsschnittstelle als RESTful HTTP Schnittstelle zur Verfügung stellt. Berechtigt wird hier akuell aufgrund einer Mock-Implementation in welcher die Regeln statisch hinterlegt sind. Im Client-Projekt wird die Client-Library implementiert welche in SYRIUS verwendet wird, um das Berechtigungssystem bzw. die RESTfull HTTP Schnittstelle, anzusprechen.

Diese logische Sicht auf den Autorisierungsprototypen wird nun detaillierter betrachtet.

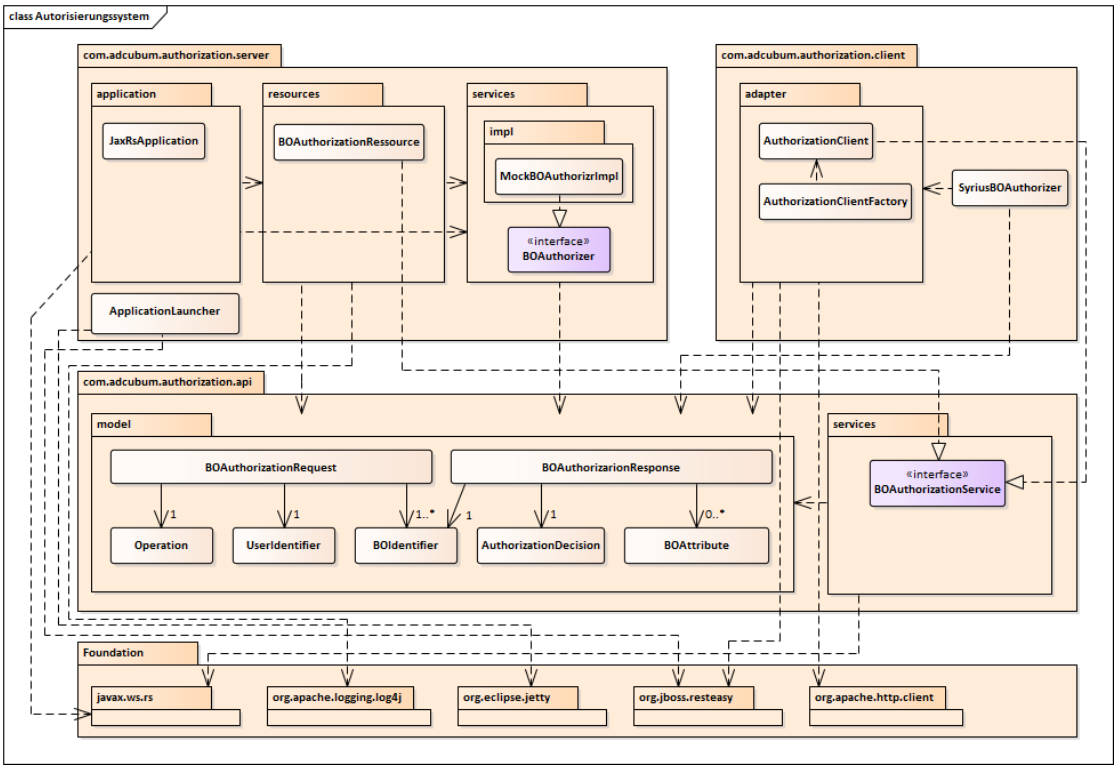

Abbildung C.2.: Logische Sicht Autorisierungssystem im Detail (UML)

Abbildung C.2 zeigt sowohl die internen Abhängigkeiten der Packages, als auch die Abhängigkeiten zu Fremdbibliotheken. Das Package *Foundation* enthält die Fremdbibliotheken, und zeigt auf, welche Packages von ihnen abhängig sind. Die darüberliegende Schicht unter dem Package *com.adcubum.authorization.api* enthält die API-Definitionen (Model und Services) der RESTful HTTP Schnittstelle. Diese werden sowohl vom Client als auch vom Server verwendet, da der Server die RESTful HTTP Resourcen zur

Verfügung stellen, und der Client diese aufrufen muss.

Unter *Server* verstehen wir im Kontext dieser Arbeit also das Autorisierungssystem oder den *Policy Decision Point (PDP)*, welcher die RESTful HTTP Resource für Autorisierungsrequests, in der Implementation *BOAuthorizationResource* genannt, bereitstellt und die Requests beantwortet. Der *Client* hingegen ist die Java-Library (JAR-File) welche SYRIUS und andere Umsysteme benutzen, um den *Policy Decision Point* aufzurufen.

Abbildung C.2 zeigt entsprechend die Implementation des Servers und dessen Abhängigkeiten unter dem Package *com.adcubum.authorization.server*. Der Client ist im Package *com.adcubum.authorization.client* implementiert.

Fremdbibliotheken

In der Abbildung C.2 wird ersichtlich, dass für die Implementation des Prototypen einige Fremdbibliotheken verwendet werden. In der folgenden Tabelle werden diese Bibliotheken mit deren Versionen und Lizenz aufgeführt. Die Richtlinie von Adcubum bezüglich Lizenzen von Fremdbibliotheken [1] wurde überprüft und der Einsatz der hier verwendeten Bibliotheken sind gemäss der Richtlinie erlaubt.

Tabelle C.1.: Fremdbibliotheken

Bibliothek	Beschreibung	Version	Lizenz
JAX-RS API 2.0	Java API für RESTful Web Services (Annotations)	1.0.0.Final	GNU General Public License, Version 2 with the Classpath Exception [11]
Log4J	Logging Framework	2.7	Apache Licence 2.0 [35]
Jetty	Embedded Server um Applikation *'standalone'* starten zu können.	9.3.14	Apache Licence 2.0 [35]
RestEasy	JBoss Projekt welches Framework bereitstellt um RESTful HTTP Applikationen zu bauen. (Implementation der JAX-RS API)	3.0.19.Final	Apache Licence 2.0 [35]
Apache HTTP Client	Wird vom RestEasy-Client benötigt um die RESTful HTTP Aufrufe machen zu können.	4.3.6	Apache Licence 2.0 [35]

C.1.2. Redesign der Persistenzschicht in SYRIUS

Die logische Sicht auf die Struktur der Packages und Klassen welche beim Redesign innerhalb von SYRIUS betroffen waren, wird nachfolgend ebenfalls noch detaillierter dokumentiert.

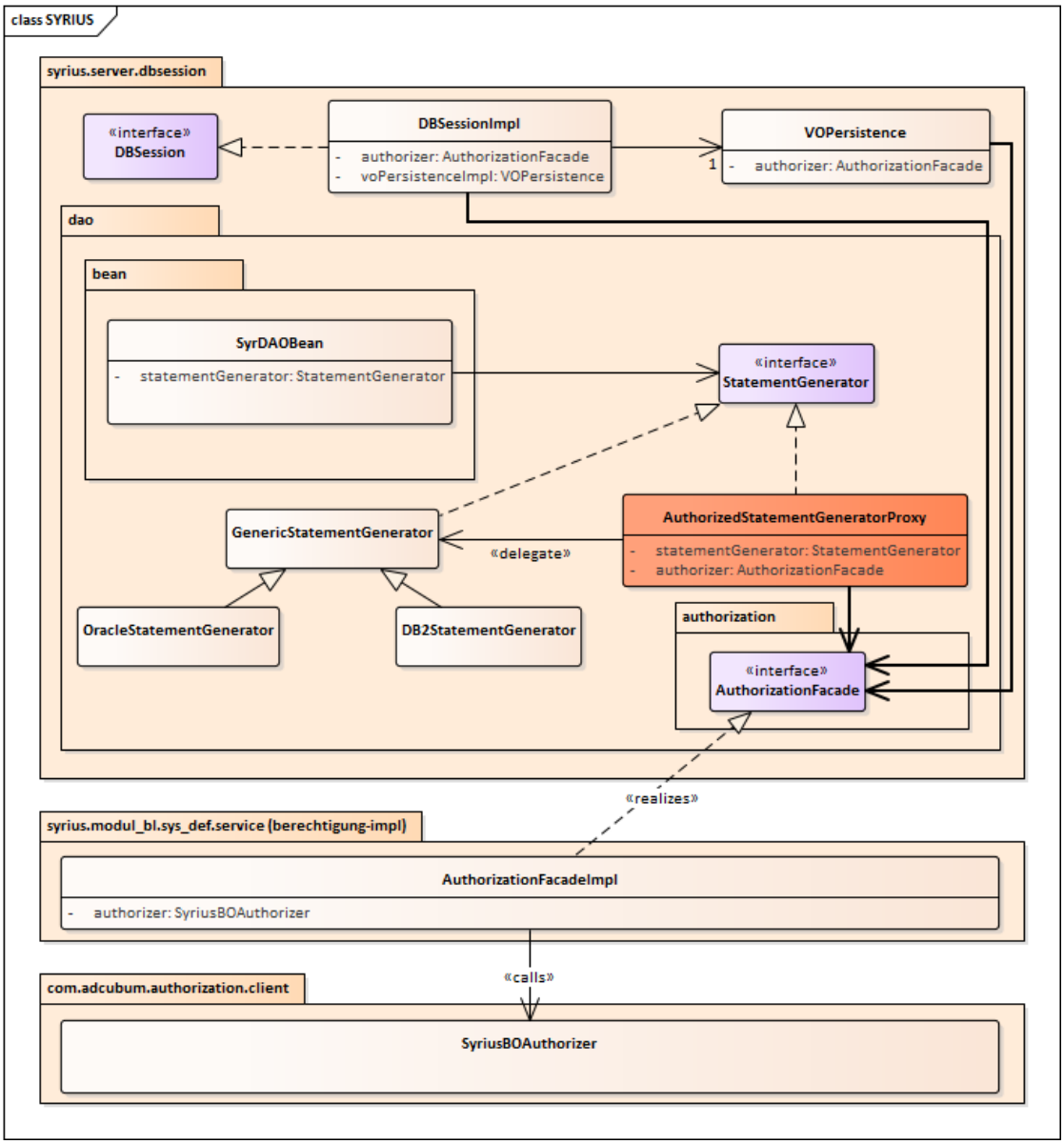

Abbildung C.3.: Logische Sicht SYRIUS (UML)

In SYRIUS werden alle Datenbankabfragen (Queries) über einen sogenannten *Statement-Generator* durchgeführt (siehe Interface in Abbildung C.3). Das Filtern der nicht autorisierten Daten ist bis Heute Aufgabe des *GenericStatementGenerator*. Dieser erweitert die SQL-Statements mit einer WHERE-Klausel, welche sicherstellt, dass nur Objekte aus der Datenbank geladen werden, welche der Benutzer auch sehen darf. Diese Logik wurde nun aus dem *GenericStatementGenerator* extrahiert. Ausserdem können BO's direkt über die *DBSession* geladen werden, wenn deren BO-Id bereits bekannt ist. Diese verwendet zusätzlich die Klasse *VOPersistence*, welche ebenfalls auf die Datenbank zugreift. Auch in diesen beiden Klassen wurde die bestehende Objektschutzfunktionalität ausgebaut.

Im Kapitel 5 wurde bereits darauf eingegangen dass das Konzept des «Proxy Pattern» bei der *DBSession* nicht umgesetzt werden konnte. Es konnte also keine Entkopplung der Persistenzlogik und der Autorisierungslogik erreicht werden, da die Aufrufe der *AuthorizationFacade* nach wie vor in den Methoden der Persistenz gemacht wird. Trotzdem zeigt Abbildung C.3 noch einmal auf dass die Stellen, an welchen die Autorisierungslösung aufgerufen werden muss, identifiziert wurden. Zumindest für die in dieser Studienarbeit betrachteten Use Cases. Es wurde bereits erwähnt dass diesbezüglich ein Risiko besteht, mit diesen Use Cases nicht alle Stellen identifiziert zu haben.

C.2. Prozess-Sicht

Ein Systemsequenzdiagramm welche den Autorisierungsprozess aufzeigt wurde bereits im Kapitel 5 vorgestellt. Der Applikationsserver lädt die Daten aus der Datenbank und macht anschliessend eine Autorisierungsanfrage an das *Autorisierungssystem*, um die Daten filtern zu können. Er schickt anschliessend nur diejenigen BO's an den UTC zurück, für welche der Benutzer auch eine Berechtigung verfügt.

In der Abbildung C.4 wird der Ablauf aus technischer Sicht innerhalb von SYRIUS, im Detail dargestellt.

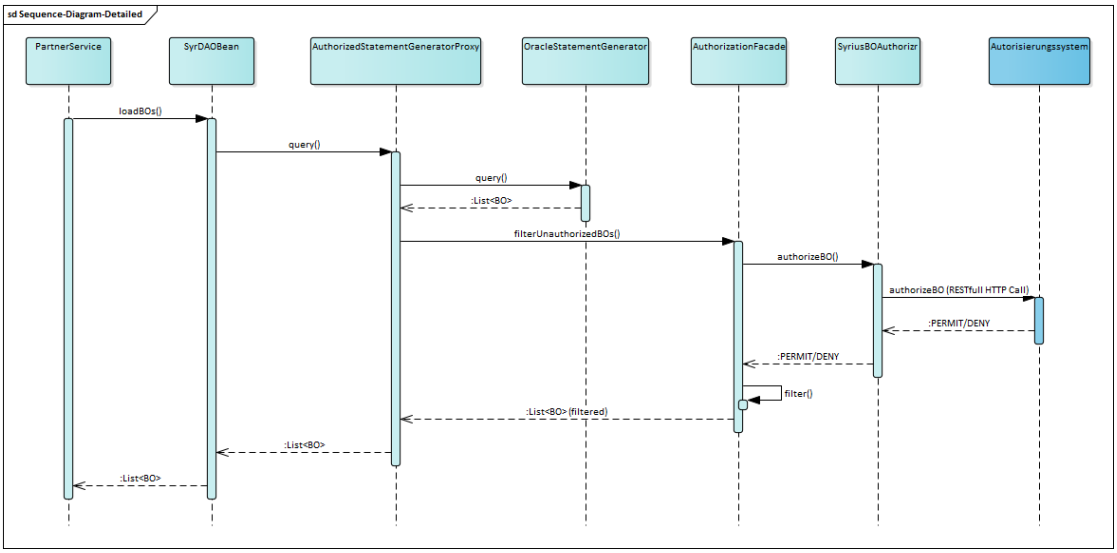

Abbildung C.4.: Prozess Sicht: Sequenzdiagram SYRIUS (UML)

Daraus wird ersichtlich dass der *OracleStatementGenerator* die Daten ungefiltert zurückliefert und der *AuthorizedStatementGeneratorProxy* die Daten anschliessend filtert. Der Ablauf ist in den Fällen in welchen nicht der *AuthorizedStatementGeneratorProxy* sondern die *DBSession* die Datenabfrage macht, der selbe. Aus Prozess-Sicht ändert sich in diesem Fall nichts, aus diesem Grund wird hier nicht noch ein weiteres Sequenzdiagramm aufgeführt.

Bei dem Autorisierungssystem, welches in Abbildung C.4 ganz rechts dargestellt ist, handelt es sich um unseren Prototypen und damit ein externes System. Die dargestellte Kommunikation zwischen dem *SyriusBOAuthorizer (Client)* und dem *Autorisierungssystem (Server)* stellt somit die neue RESTful HTTP Schnittstelle dar.

C.3. Deployment-Sicht

Die folgende Abbildung C.5 zeigt ein *mögliches* Deployment des Autorisierungssystems. Diese Variante wurde im Hauptbericht bereits erläutert.

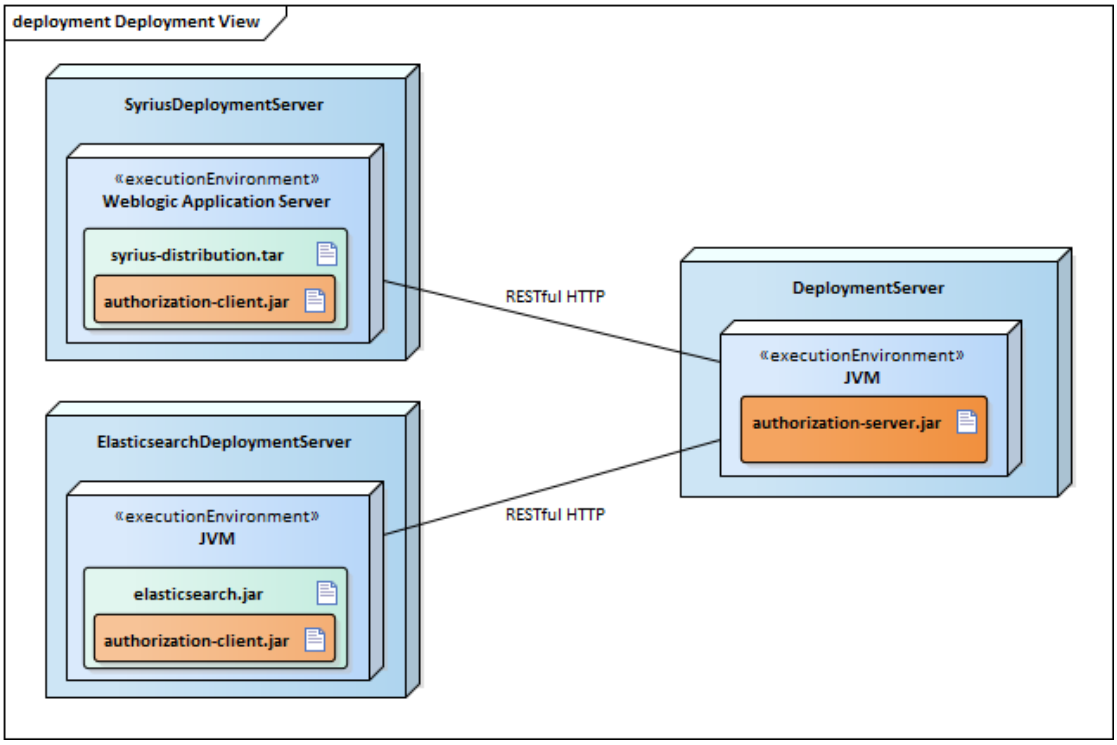

Abbildung C.5.: Deployment Sicht: Variante 1 (UML Deploymentdiagramm)

In diesem Fall besteht das Deployment aus dem Autorisierungsserver, SYRIUS und einer Elasticsearch-Instanz [7]. Sowohl SYRIUS als auch Elasticsearch kommunizieren über RESTful HTTP mit dem Autorisierungsserver um Berechtigungsabfragen zu machen.

Entscheidend ist die Tatsache dass diese Komponenten auf verschiedenen *Servern* laufen können, wenn dies erwünscht ist. Sollte dies nicht notwendig sein, ist ein Deployment zusammen mit SYRIUS auf dem selben *Server* ebenfalls denkbar, wie Abbildung C.6 zeigt.

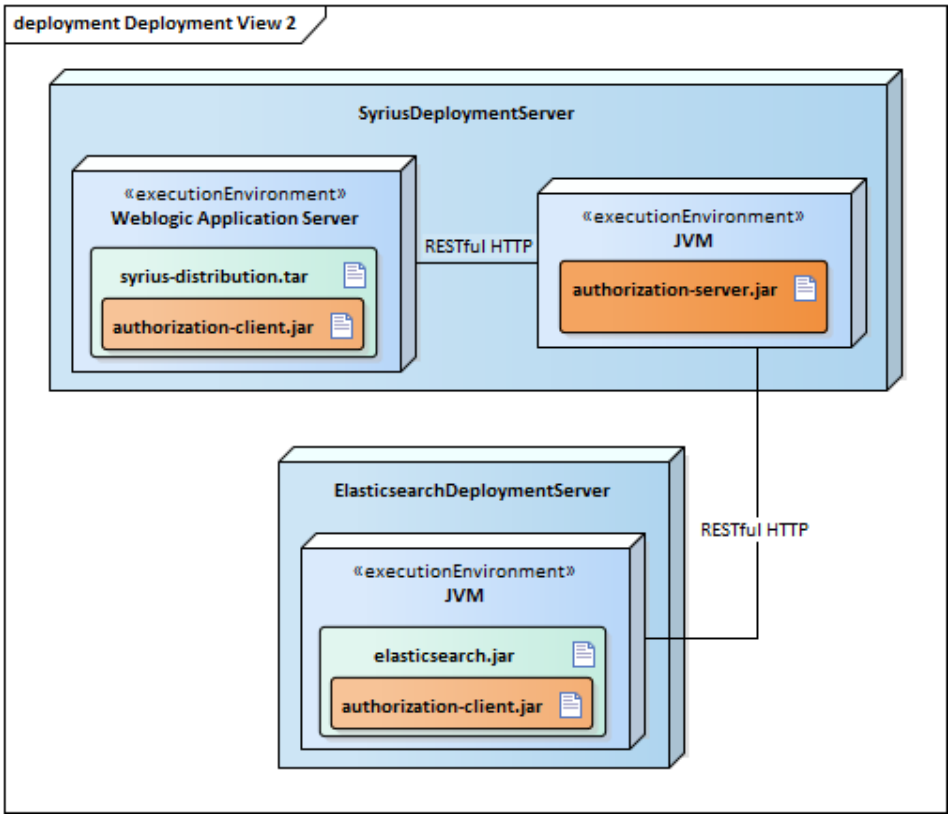

Abbildung C.6.: Deployment Sicht: Variante 2 (UML Deployment-Diagramm)

Dieses Modell kann auch aus Performance-Gründen bevorzugt werden.

C.4. Entwicklungsumgebung

C.4.1. Technologie / IDE

Der Prototyp wurde auf Basis von Java 8 entwickelt. Als IDE wurde Eclipse Mars verwendet, da dieses in der Entwicklungsumgebung von Adcubum bereits zur Verfügung steht. Für die Entwicklung der Unit Tests wurde JUnit [25], Mockito [31] und Hamcrest [15] eingesetzt.

C.4.2. Build-System

Als Build-System wurde Gradle [13] verwendet, da es das Standard Build-Tool bei Adcubum ist. Es wird verwendet um das Projekt zu kompilieren, die Unit Tests auszuführen und um die JAR-Files zu erstellen und im Artefakt-Repository zu publizieren.

C.4.3. Versionierung

Wie beim Industriepartner Adcubum üblich, wurde der Source-Code in einem Git-Repository versioniert.

C.4.4. Code Reviews

Während der *Construction*-Phasen wurden regelmässig Code-Reviews mit dem Industriepartner (Alex Gfeller) durchgeführt. Am Ende des Projekts wurde noch ein Review über die gesamte Code-Basis des Prototypen durchgeführt. Die Feedbacks und Schlussfolgerungen der Reviews wurden jeweils in der nächsten Iteration umgesetzt.

C.4.5. Codestyle

Der Code wurde nach den Codestyle-Regeln des Industriepartners formatiert. Dieser wird in der IDE (Eclipse) bei jedem Speichern des Source-Codes angewendet.

C.4.6. Continuous Integration

Adcubum setzt als CI-Tool Jenkins [24] ein. Damit sichergestellt ist, dass nach jedem Commit ins Git-Repository ein Build und damit alle Unit-Tests ausgeführt werden, wurde auf dem Jenkins [24] ein Job eingerichtet welcher den Gradle-Build ausführt und die generierten Artefakte bei erfolgreichem Build im Artefakt-Repository publiziert. Dadurch werden Kompilier-Fehler oder Unit-Tests welche fehlschlagen kurz nach dem Commit erkannt.

D. System Test

D.1. Motivation

Mit der Durchführung dieses Systemtests wurde geprüft ob die funktionalen Anforderungen entsprechend der Anforderungsspezifikation erfüllt wurden. Es wurde überprüft ob sich das laufende System mit dem neuen Autorisierungsprototypen gleich verhält wie in den Szenarien in Anhang B welche am Anfang des Projekte erstellt wurden.

Das Testziel ist daher, dass sich die Use Cases nach dem durchgeführten Redesign noch gleich verhalten, wie mit der ursprünglichen Berechtigungslösung. Der Systemtest wurde am Ende der Implementationsphase einmalig und manuell durchgeführt.

Der Systemtest hätte grundsätzlich automatisiert werden können, indem am Anfang des Projekts die Szenarien mit der Testautomations-Lösung von Adcubum erfasst worden wären. Allerdings wäre der Einarbeitungsaufwand in das Tool in Anbetracht der überschaubaren Use Cases im Rahmen der Studienarbeit zu gross gewesen. Aus diesem Grund wurden die Szenarien in Anhang B detailliert beschrieben und konnten so einfach manuell nachgestellt werden.

D.2. Voraussetzungen und Vorbereitungen

D.2.1. Adcubum SYRIUS

Seitens SYRIUS, müssen keine speziellen Vorbereitungen für den Systemtest getroffen werden. Der Prototyp für diese Studienarbeit wurde in den Syrius-Repositories auf dem Feature-Branch *feature/objektschutz-refactoring-prototype* durchgeführt. Es muss lediglich sichergestellt werden dass Syrius mit dem Code-Stand dieses Feature-Branches gebaut und ausgeführt wird.

Ausserdem muss sichergestellt werden dass die HTTP-Verbindung zum neuen Berechtigungssystem (Mock-Implementation) funktioniert.

D.2.2. Berechtigungssystem (Prototyp)

Um den Systemtest durchführen zu können, muss der Prototyp des Berechtigungssystems ausgeführt werden, und von SYRIUS per HTTP erreichbar sein. Bei unserem Systemtest wurde sowohl SYRIUS, als auch der Berechtigungsprototyp, direkt aus der Entwicklungsumgebung ausgeführt. Das Berechtigungssystem kann über die Klasse *ApplicationLauncher* als normale Java-Applikation (main-Methode) ausgeführt werden. In Auflistung D.1 ist die Ausgabe der Applikation ersichtlich, wenn sie erfolgreich gestartet wurde.

```
1  INFO: RESTEASY002225: Deploying javax.ws.rs.core.Application: ...
2  Nov 27, 2016 4:47:46 PM org.jboss.resteasy.spi.ResteasyDeployment...
3  INFO: RESTEASY002205: Adding provider class com.adcubum.tools.jax.rs....
4  Nov 27, 2016 4:47:46 PM org.jboss.resteasy.spi.ResteasyDeployment...
5  INFO: RESTEASY002220: Adding singleton resource com.adcubum.author...
6  Nov 27, 2016 4:47:46 PM org.jboss.resteasy.spi.ResteasyDeployment...
7  INFO: RESTEASY002220: Adding singleton resource com.adcubum.autho...
8  2016-11-27 16:47:46.454:INFO:oejsh.ContextHandler:main: Started ...
9  2016-11-27 16:47:46.464:INFO:oejs.AbstractConnector:main: Starte...
10 2016-11-27 16:47:46.464:INFO:oejs.Server:main: Started @668ms
```

Auflistung D.1: Konsolenausgabe Prototyp

D.3. Protokoll der Testdurchführung

In diesem Abschnitt werden die Durchführungen der Systemtests der einzelnen Use Cases dokumentiert. Die detaillierten Beschreibungen der Use Cases befinden sich im Anhang B.

D.3.1. Use Case 1: Partner-BO schützen

Benutzer mit Zugriff

Wie in der Anforderungsspezifikation in Kapitel 4 festgehalten, muss der Benutzer *SA_UC01_I-have-access* Zugriff auf den Partner *UC01 Partner* haben:

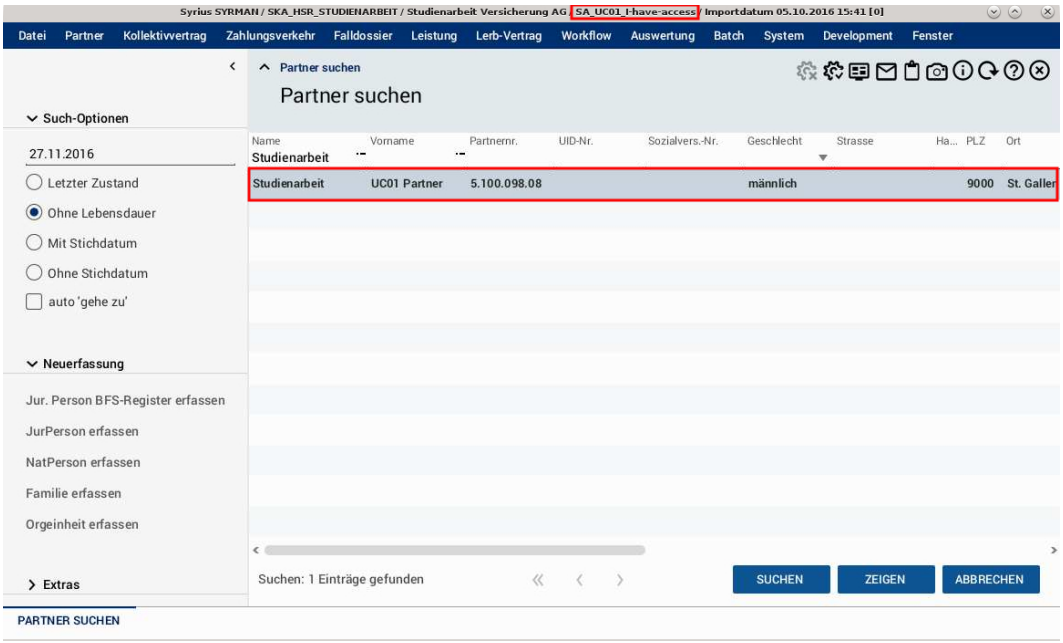

Abbildung D.1.: UC01: Benutzer SA_UC01_I-have-access hat Zugriff auf Partner

Auflistung D.2 zeigt die Konsolenausgabe der entsprechenden Autorisierungsabfrage, aus welcher ersichtlich wird, dass der Prototyp mit *PERMIT* geantwortet hat.

```
18:22:53.490 [qtp2050404090-127] INFO   - PERMIT READ UserIdentifier
 [username=SA_UC01_I-have-access] BOIdentifier [metaBoId=3, boId=28401]
```

Auflistung D.2: Konsolenausgabe Autorisierungsrequest (SA_UC01_I-have-access)

Benutzer ohne Zugriff

Der Benutzer *SA_UC01_I-dont-have-access* hingegen hat keinen Zugriff auf den Partner:

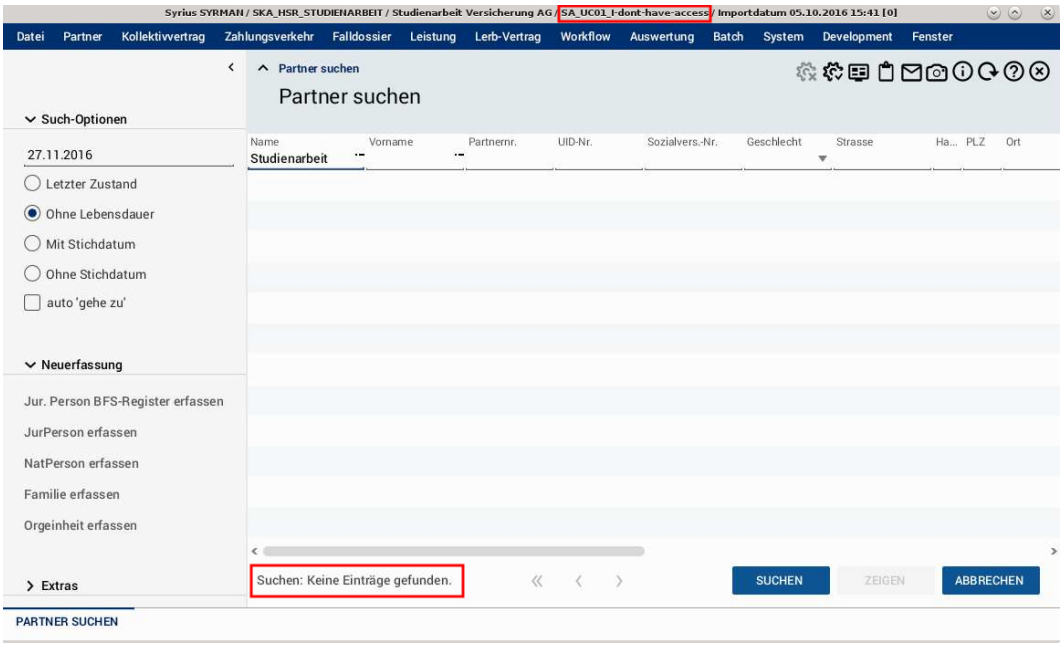

Abbildung D.2.: UC01: SA_UC01_I-dont-have-access hat keinen Zugriff auf den Partner

Auflistung D.3 zeigt die Antwort (DENY) des Prototypen.

```
18:45:01.681 [qtp2050404090-146] INFO   - DENY READ UserIdentifier
 [username=SA_UC01_I-dont-have-access] BOIdentifier [metaBoId=3,
  boId=28401]
```

Auflistung D.3: Konsolenausgabe Autorisierungsrequest (SA_UC01_I-dont-have-access)

D.3.2. Use Case 2: Partner BO-Attribute schützen

Benutzer mit Zugriff auf alle Attribute

Zur Verdeutlichung des Attributschutzes wurden auch bei diesem Use Case zwei Benutzer verwendet. Der Benutzer *SA_UC02_I-can-see-all-attributes* hat dabei Zugriff auf alle Attribute des Partners:

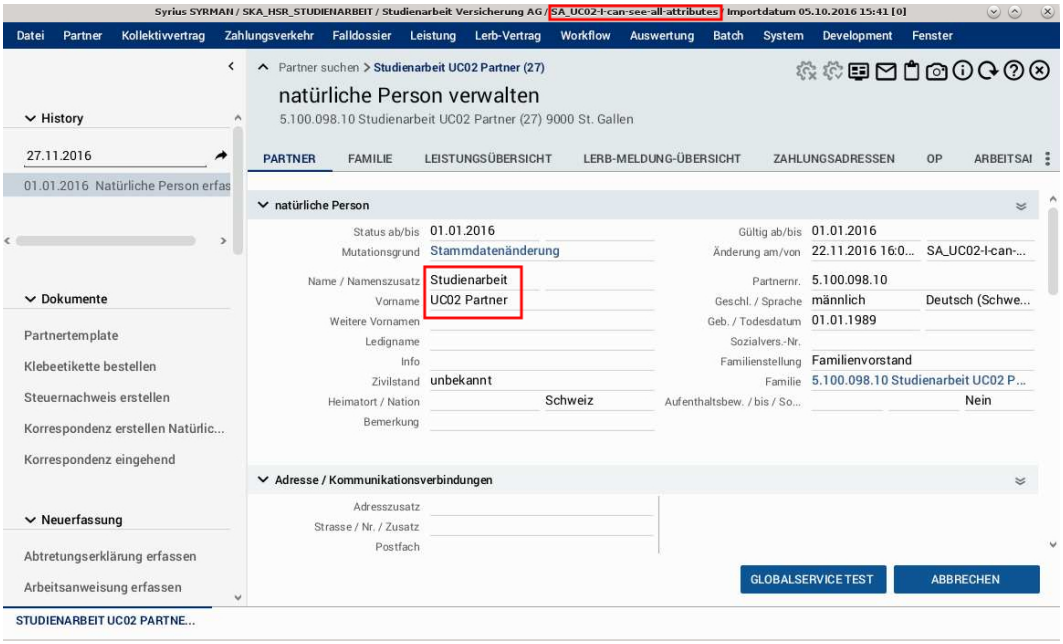

Abbildung D.3.: UC02: SA_UC02_I-can-see-all-attributes hat Zugriff auf alle Attribute

Auflistung D.4 zeigt die Konsolenausgabe für den entsprechenden Zugriff, welche keine Restriktionen auf BO-Attribute enthält.

```
19:11:00.721 [qtp2050404090-25] INFO   - PERMIT READ UserIdentifier
 [username=SA_UC02-I-can-see-all-attributes] BOIdentifier [metaBoId=3,
  boId=28421]
```

Auflistung D.4: Autorisierungsrequest (SA_UC02_I-can-see-all-attributes)

Benutzer mit Attribut-Restriktionen

Der Benutzer *SA_UC02_I-cannot-see-all-attributes* hingegen, sieht nicht alle Attribute, wie dies in der Anforderungsspezifikation in Kapitel 4 schon gefordert wurde:

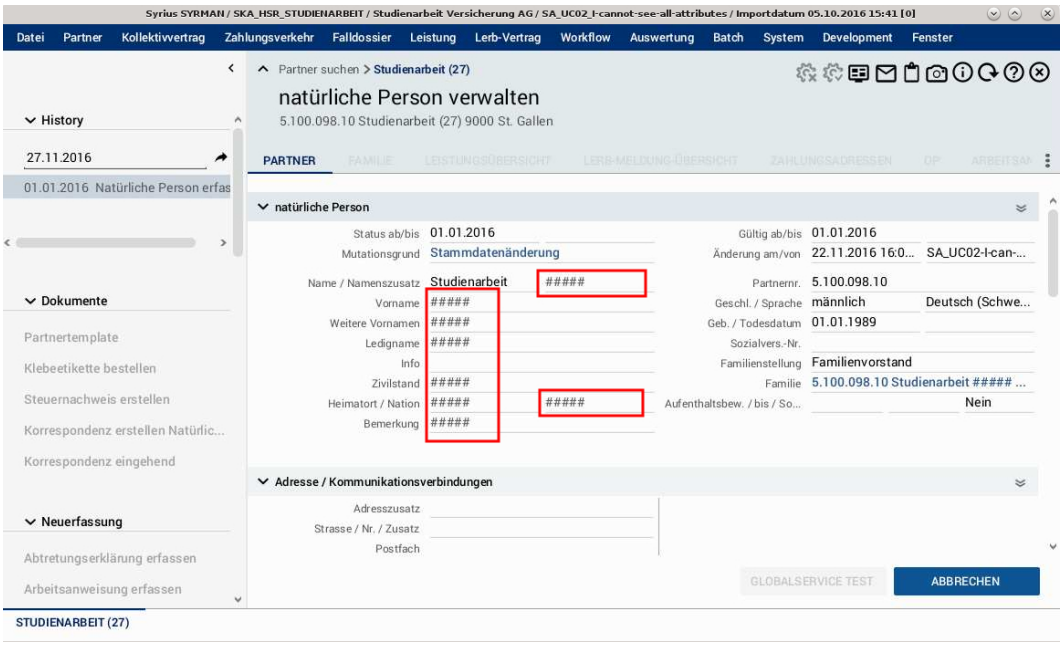

Abbildung D.4.: SA_UC02_I-cannot-see-all-attr... hat nicht auf alle Attribute Zugriff

In Auflistung D.5 sind in der Konsolenausgabe dieses Autorisierungsrequests auch die Attribut-Restriktionen sichtbar.

```
19:24:10.187 [qtp2050404090-51] INFO   - PERMIT READ UserIdentifier
 [username=SA_UC02_I-cannot-see-all-attributes] BOIdentifier [metaBoId=
  3, boId=28421] WITH UNAUTHORIZED ATTRIBUTES [[BOAttribute
  [name=Vorname], BOAttribute [name=WeitereVornamen],
  BOAttribute [name=NameZusatz], BOAttribute
  [name=LedigName], BOAttribute [name=Zivilstand],
  BOAttribute [name=Heimatort], BOAttribute
  [name=Nationalitaet], BOAttribute [name=Bemerkung1],
  BOAttribute [name=Bemerkung2]]]
```

Auflistung D.5: Autorisierungsrequest (SA_UC02_I-cannot-see-all-attributes)

Wird dieser Fall mit der Anforderungsspezifikation in Anhang B verglichen, fällt auf,

dass die gesperrten Attribute nicht exakt übereinstimmen. Der Use Case wurde an dieser Stelle leicht vereinfacht, wie dies später in diesem Anhang noch genauer erläutert wird.

D.3.3. Use Case 3: Partner-BO vor unbefugten Mutationen schützen (Adressmutation)

Benutzer mit Mutationsberechtigung

Auch bei diesem Use Case wurden zwei Benutzer erstellt. Der erste Benutzer *SA_UC03_I-can-read-and-write* besitzt eine Mutationsberechtigung und kann auf dem Partner entsprechend eine Adressänderung vornehmen:

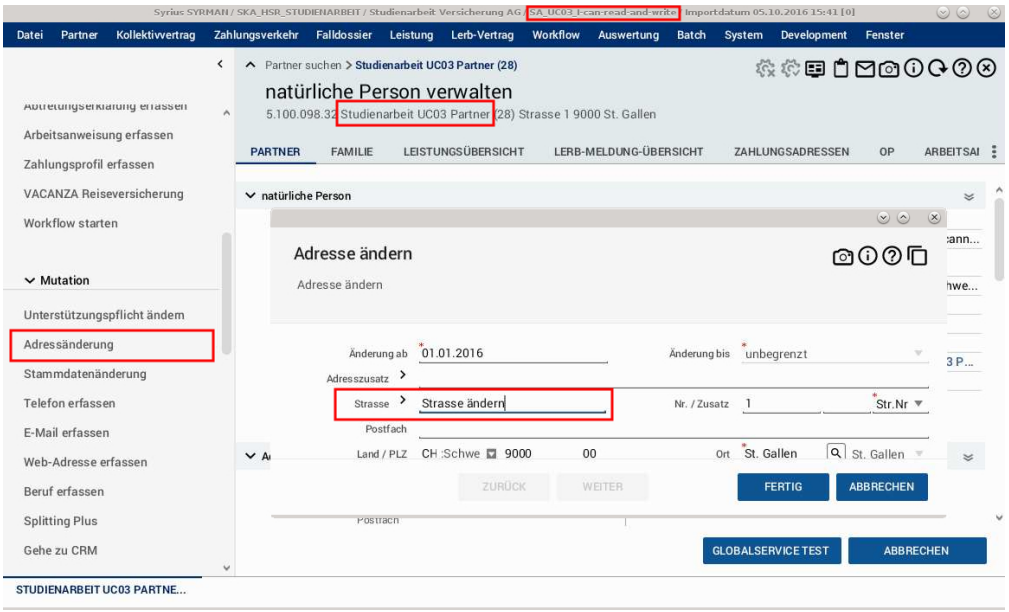

Abbildung D.5.: Benutzer SA_UC03_I-can-read-and-write darf Adresse mutieren

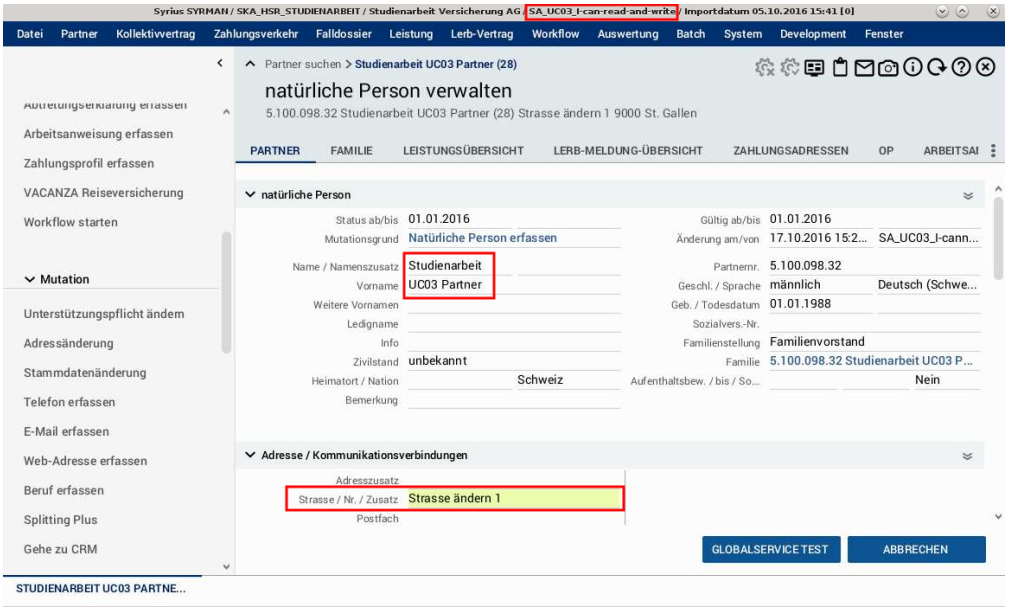

Abbildung D.6.: UC03: Geänderte Adresse

Dass diese Mutation vom Prototypen autorisiert wurde, zeigt die Konsolenausgabe aus Auflistung D.6.

```
20:09:40.495 [qtp2050404090-25] INFO   - PERMIT WRITE UserIdentifier
 [username=SA_UC03_I-can-read-and-write] BOIdentifier [metaBoId=-7,
  boId=125581]
```

Auflistung D.6: Konsolenausgabe Autorisierungsrequest (SA_UC03_I-can-read-and-write)

Benutzer ohne Mutationsberechtigung

Der Benutzer *SA_UC03_I-cannot-write* hingegen darf den Partner nicht mutieren, und kann daher keine Mutationstasks aufrufen:

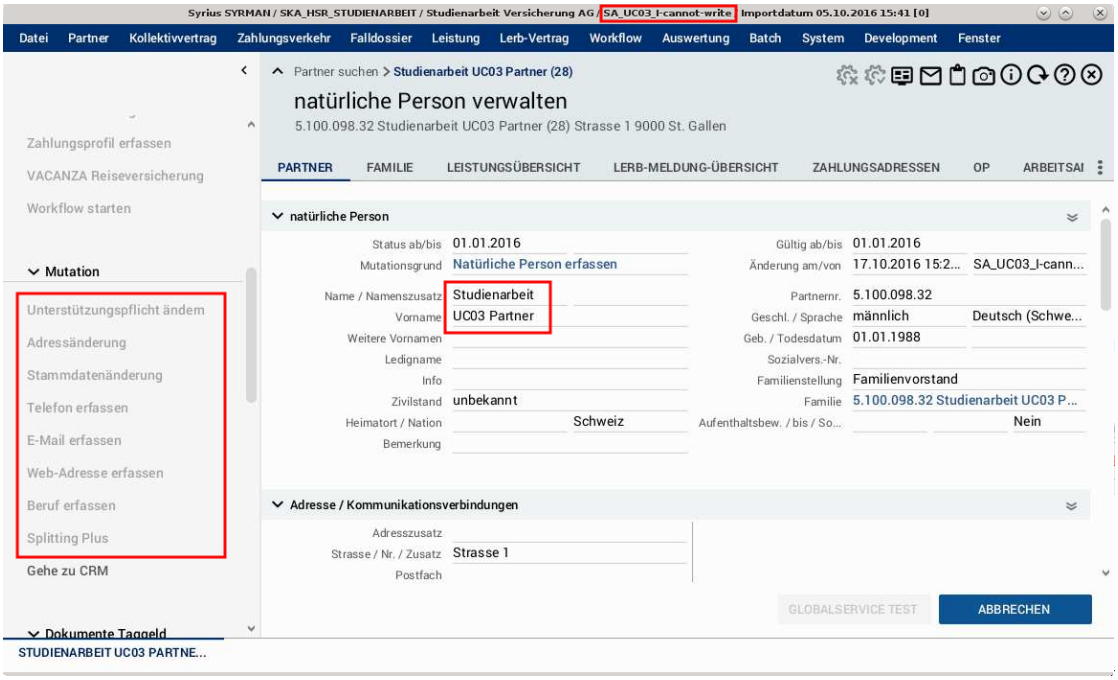

Abbildung D.7.: UC03: Mutationstasks blockiert für Benutzer SA_UC03_I-cannot-write

Allein durch diesen Test, können wir noch nicht mit Sicherheit sagen dass die Persistenzschicht von SYRIUS einen Update verbieten würde.

```
20:19:39.792 [qtp2050404090-102] INFO   - PERMIT READ UserIdentifier
 [username=SA_UC03_I-cannot-write] BOIdentifier [metaBoId=-7,
  boId=125581]
```

Auflistung D.7: Lesen ist auch für SA_UC03_I-cannot-write erlaubt.

Aus diesem Grund prüfen wir die Persistenzschicht mit einem direkten Service-Aufruf via SOAP-Service, wie bereits in der Anforderungsspezifikation in Anhang B erwähnt.

Mittels SOAP-UI setzen wir den Request aus Auflistung D.8 ab:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope>
   <soapenv:Header>
      <urn:SoapLoginInformation>
         <Password>1234</Password>
         <Username>SA_UC03_I-cannot-write</Username>
      </urn:SoapLoginInformation>
   </soapenv:Header>
   <soapenv:Body>
      <glob:replicatePartner>
         <ns4:message>
            <Adressen>
               <AdressStates>
                  <AdressOrder id="-36200" />
                  <Hausnummer></Hausnummer>
                  <KontaktPersAnr id="-46300" type="-14"/>
                  <Land>CH</Land>
                  <MutationsGrund id="1442" type="-355"/>
                  <Ort>St. Gallen</Ort>
                  <PLZ>9000</PLZ>
                  <PLZ_zusatz>00</PLZ_zusatz>
                  <PlzId>5189</PlzId>
                  <Sprache language="de" country="CH" variant=""/>
                  <!-- Versuch, die Strasse zu ändern: -->
                  <Strasse>Strasse ändern</Strasse>
               </AdressStates>
               <AdresseTyp id="-43800" />
               <ExtAdresseId>UC03-Partner-Adresse</ExtAdresseId>
               <GueltigAb>2016-01-01</GueltigAb>
               <GueltigBis>3000-01-01</GueltigBis>
            </Adressen>
            <ExtPartnerId>UC03-Partner</ExtPartnerId>
         </ns4:message>
      </glob:replicatePartner>
   </soapenv:Body>
</soapenv:Envelope>
```

Auflistung D.8: UC03: Mutation mit SA_UC03_I-cannot-write über SOAP-Service

Auf diesen Request antwortet Syrius mit einer Response welche eine entsprechenden Fehlermeldung enthält, und uns darauf hinweist dass der Benutzer die entsprechende Mutationsberechtigung nicht besitzt:

```
<soapenv:Envelope>
   <soapenv:Body>
      <soapenv:Fault>
         <faultcode>soapenv:Server</faultcode>
         <faultstring>SYR-3291 Fehler: Keine Mutationsberechtigung für
                ’-7’ (Id: 125581, Typ: -7, Schutzobjekt: DENY)
                (ERROR_ID="97021be1-d412-4560-833b-6985a70bb7e2")
         </faultstring>
         <detail>
            <ns344:ApplicationException>
               <alert>
                  <name>SYR-3291</name>
               </alert>
               <texts>
                  <text>Keine Mutationsberechtigung für  ’%1’
                        (Id: %2, Typ: %3, Schutzobjekt: %4)
                  </text>
                  <Locale language="de" country="" variant=""/>
               </texts>
               <templateArguments>-7</templateArguments>
               <templateArguments>125581</templateArguments>
               <templateArguments>-7</templateArguments>
               <templateArguments>DENY</templateArguments>
               <stacktrace>syrius.util.exception.ApplicationException:
                SYR-3291 Fehler: Keine Mutationsberechtigung für  ’-7’
                (Id: 125581, Typ: -7, Schutzobjekt: DENY)
                (ERROR_ID="97021be1-d412-4560-833b-6985a70bb7e2")
                at syrius.modul_bl.service.soap.service.impl.SoapHandler
                at syrius.modul_bl.service.soap.service.proxy.SoapHandle
                at syrius.modul_bl.service.soap.SOAPHandler.handleBody
                <!-- (gekürzt) -->
               </stacktrace>
            </ns344:ApplicationException>
         </detail>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

Auflistung D.9: UC03: Adressmutation mit User SA_UC03_I-cannot-write nicht möglich

Die Ausgabe der Konsole des Prototyps bestätigt dass der Schreibzugriff mit einem *DENY* beantwortet wurde, wie Auflistung D.10 zeigt.

```
20:31:16.045 [qtp2050404090-103] INFO   - DENY WRITE UserIdentifier
 [username=SA_UC03_I-cannot-write] BOIdentifier [metaBoId=-7,
  boId=125581]
```

Auflistung D.10: Schreiben ist für SA_UC03_I-cannot-write nicht erlaubt

D.4. Resultate und Schlussfolgerungen

D.4.1. Übersicht

Tabelle D.1.: Übersicht der Testresultate

Use Case 1: Partner-BO schützen	Die funktionalen Anforderungen an Use Case 1 wurden **erfüllt**.
Use Case 2: Partner BO-Attribute schützen	Die Anforderungen an Use Case 2 wurden **mit Einschränkungen erfüllt**.
Use Case 3: Partner-BO vor unbefugten Mutationen schützen	Die funktionalen Anforderungen an Use Case 2 wurden **erfüllt**.

Die Use Cases 1 und 3 haben exakt das selbe Verhalten wie in den spezifizierten Szenarien gezeigt. Damit können diese als **erfüllt** betrachtet werden.

D.4.2. Einschränkungen Use Case 3: Attributschutz

Aus dem Test von Use Case 2 (Schutz von BO-Attributen) wurde ersichtlich dass die geschützten Attribute nicht exakt die selben sind wie in der Anforderungsspezifikation bzw. der Ausgangslage. Dies beruht auf der Tatsache dass im ursprünglichen Objektschutz von SYRIUS mit einer Whitelist gearbeitet wurde. Sprich, es wurden in der Parametrierung alle Attribute aufgelistet, welche der Benutzer sehen darf. Die Schnittstelle des neuen Prototypen arbeitet nun mit einer Blacklist. Wir geben der Autorisierungsresponse eine Liste der Attribute mit, welche der Benutzer nicht sehen darf. Eine Whitelist wäre ebenfalls möglich gewesen, allerdings wäre der Aufwand grösser gewesen, in der Mock-Implementation alle Attribute aufzulisten. Aus diesem Grund wurde dieser Use Case etwas vereinfacht, mit der Einschränkung dass die geschützten Attribute nun nicht mehr

exakt die selben sind. Dies wurde mit dem Industriepartner abgesprochen und für gut befunden. Für den Prototypen ist diese Lösung ausreichend.

D.4.3. Zusammenfassung

Die Durchführung des Systemtests hat gezeigt dass die funktionalen Anforderungen, mit einer kleinen Einschränkung, erfolgreich erfüllt wurden. Da die Einschränkung mit dem Industriepartner abgesprochen war und bewusst auf eine strikte Einhaltung des entsprechenden Szenario verzichtet wurde, können die Use Cases zusammenfassend als **erfüllt** betrachtet werden.

In der Anforderungsspezifikation in Kapitel 4 wurde darauf hingewiesen dass es zu den spezifizierten Use Cases in der Realität alternative Szenarien gibt. Zum Beispiel kann ein Partner-BO über verschiedene Services geladen oder aktualisiert werden. Ausserdem gibt es auch im UTC-Client verschiedene Orte an welchen Partner-Daten angezeigt werden.

In der Anforderungsspezifikation wurde gefordert dass die Berechtigungsprüfung weiterhin an einer zentralen Stelle gemacht wird, sodass auch alle alternativen Szenarien abgedeckt sind. Während der Implementation des Prototypen hat sich allerdings gezeigt, dass es aufgrund aktueller Gegebenheiten im Code von SYRIUS, nicht so einfach ist alle diese Stellen zu identifizieren. Dies wäre in der zur Verfügung stehenden Zeit dieser Studienarbeit nicht möglich gewesen. Aus diesem Grund musste der Fokus auf die spezifizierten Use Cases gelegt werden. Es kann daher zum aktuellen Zeitpunkt nicht ausgeschlossen werden, dass es noch Stellen in der Persistenz-Schicht gibt, an welchen die Berechtigungsprüfung fehlt.

Daher können die Anforderungen an die alternativen Szenarien aus der Anforderungsspezifikation **nicht als erfüllt** betrachtet werden. Im Hauptbericht wurde darauf bereits eingegangen. Die daraus gezogene Schlussfolgerung ist dass eine Umsetzung des Konzepts dieser Studienarbeit ein grösseres Redesign der Persistenzschicht zur Folge hat.

E. Aufgabenstellung

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

E.1. Ausgangslage

In SYRIUS, einer geschichteten ERP-Lösung für Versicherungen, sind Zugriffsberechtigungen in der Datenbank abgelegt. Bei jedem Datenzugriff wird die WHERE-clause des SQL-Statements um die entsprechende Berechtigungsüberprüfung ergänzt. Solange nur Daten berechtigt werden müssen, welche in der Datenbank abgelegt sind, ist diese Lösung ausreichend. Für die Berechtigung von Zugriffen auf externe Daten ist der aktuelle Ansatz jedoch unangemessen kompliziert; schon heute speichert SYRIUS gewisse Datenteilbestände in Elasticsearch als externem Index.

Domain-Driven Design (DDD) ist eine in der Praxis bewährte Vorgehensweise, um monolithische Anwendungen schrittweise zu komponentisieren und als Microservices flexibel deploybar zu machen. Microservices ist ein aktueller Implementierungsansatz für Serviceorientierte Architekturen (SOA), der strategisches DDD als eine mögliche Methodik für die Serviceidentifikation vorschlägt.

E.2. Ziele der Arbeit und Liefergegenstände

Im Rahmen der strategischen Wartung von adcubum SYRIUS werden die einzelnen Module immer aufs Neue geschärft. Mit Hilfe von DDD werden dabei die Schnitte der einzelnen Applikationskomponenten analysiert und wenn nötig angepasst. Aktuell führen diese Überlegungen dazu, dass das aktuelle Berechtigungssystem extrahiert werden soll, um einen eigenen DDD „Bounded Context“ zu bilden. Dieser soll dann als eigener Microservice zur Verfügung stehen. Dieses neue Berechtigungssystem soll auf dem „Attribute-based Access Control“ (ABAC) Paradigma basieren.

Im Rahmen der Studienarbeit soll überprüft werden, an welchen Stellen das neue ABAC-Berechtigungssystem aufgerufen werden muss, um die heutige SQL-Lösung zu ersetzen. Dabei ist zu überprüfen, welche Informationen dem zukünftigen System übergeben wer-

den müssen; daraus ist eine Schnittstellendefinition zu erstellen. Das zukünftige System kann dabei grundsätzlich als Blackbox betrachtet bzw. eine Mock-Implementierung erstellt werden. Die Migration der heutigen Berechtigungsinformationen muss nicht betrachtet werden. Die Liefergegenstände der Arbeit sind der refaktorierte SYRIUS-Code, das Mock-System und die Schnittstellendefinition.

Ein weiteres Ziel der Arbeit besteht in der Aufbereitung des im Rahmen der Arbeit gesammelten Architekturwissens in Form von Patterns und Architectural Refactorings. Dazu kann das am IFS im Rahmen einer Masterarbeit entwickelte webbasierte Wissensmanagementwerkzeug ART (https://eprints.hsr.ch/503/9, https://github.com/bisigc/art) genutzt und gegebenenfalls auch erweitert werden.

Weiterhin soll die Eignung des Open Source Werkzeugs Service Cutter (BA HS 2015) inklusive Kriterienkatalog evaluiert werden (z.B. fehlen Kriterien, Schnittstellen, oder Features?) Optional soll auch der Einsatz des Open Source Projekts Structurizr zur teilautomatisierten Komponenten- und Servicevisualisierung des Altsystems und des neuen Berechtigungskonzeptes in Betracht gezogen werden.

Literaturverzeichnis

[1] Adcubum AG. Anbindung Fremdbibliotheken. `https://confluence.adcubum.com/display/SYRFRONT/Anbindung+Fremdbibliotheken`. [Adcubum Intern; Abgerufen am 27.10.2016].

[2] AuthZForce. AuthZForce (Community Edition). `https://authzforce.ow2.org/bin/view/Main/`, 2016. [Online; Abgerufen am 19.12.2016].

[3] Axiomatics. Axiomatics Policy Server. `https://www.axiomatics.com/solutions/products/authorization-for-applications/axiomatics-policy-server.html`, 2016. [Online; Abgerufen am 19.12.2016].

[4] H. b. Shen and F. Hong. An attribute-based access control model for web services. In *2006 Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'06)*, pages 74–79, Dec 2006.

[5] Christian Bisig. Ein werkzeugunterstütztes Knowledge Repository für Architectural Refactoring. Master's thesis, University of Applied Sciences HSR, Rapperswil, 2016.

[6] N. Dan, S. Hua-Ji, C. Yuan, and G. Jia-Hu. Attribute based access control (abac)-based cross-domain access control in service-oriented architecture (soa). In *2012 International Conference on Computer Science and Service System*, pages 1405–1408, Aug 2012.

[7] Elastic. Elasticsearch. `https://www.elastic.co/products/elasticsearch`. [Online; Abgerufen am 29.10.2016].

[8] Eric Evans. DDD and Microservices: At Last, Some Boundaries! `https://www.infoq.com/presentations/ddd-microservices-2016`, 2016. [Online; Abgerufen am 19.12.2016].

[9] Evans. *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[10] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[11] Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-

1301 USA. GNU GENERAL PUBLIC LICENSE, Version 2. `http://repository.jboss.org/licenses/gpl-2.0-ce.txt`, 1991. [Online; Abgerufen am 28.11.2016].

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition, 1994.

[13] Gradle Inc. Gradle Build Tool - Modern Open Source Build Automation. `https://gradle.org/`, 2016. [Online; Abgerufen am 10.12.2016].

[14] Michael Gysel, Lukas Kölbener, Wolfgang Giersche, and Olaf Zimmermann. *Service Cutter: A Systematic Approach to Service Decomposition*, pages 185–200. Springer International Publishing, Cham, 2016.

[15] hamcrest.org. Hamcrest - Matchers that can be combined to create flexible expressions of intent. `http://hamcrest.org/`, 2016. [Online; Abgerufen am 10.12.2016].

[16] http://raml.org/. RESTFul API Modelling Language (RAML). `http://raml.org/`, 2016. [Online; Abgerufen am 05.12.2016].

[17] https://jan.newmarch.name. HOPP: Half Object plus Protocol. `https://jan.newmarch.name/distjava/hopp/lecture.html`, 2007. [Online; Abgerufen am 29.11.2016].

[18] http://www.gnu.org. GNU General Public License 3. `https://www.gnu.org/licenses/gpl-3.0.de.html`, 2007. [Online; Abgerufen am 04.12.2016].

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

[20] Marc Hüffmeyer and Ulf Schreier. Formal comparison of an attribute based access control language for restful services with xacml. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies*, SACMAT '16, pages 171–178, New York, NY, USA, 2016. ACM.

[21] Marc Hüffmeyer 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.

[22] European Telecommunications Standards Institute. Interoperability. `http://www.etsi.org/standards/why-we-need-standards/interoperability`. [Online; Abgerufen am 27.10.2016].

[23] JBoss Community. RESTEasy (a JBoss Project). `http://resteasy.jboss.org/`, 2016. [Online; Abgerufen am 06.12.2016].

[24] Jenkins Project. Jenkins - Build great things at any scale. `https://jenkins.io/`, 2016. [Online; Abgerufen am 10.12.2016].

[25] junit.org. JUnit - Simple Java Unit Testing Framework. `http://junit.org/junit4/`, 2016. [Online; Abgerufen am 10.12.2016].

[26] Structurizr Limited. Structurizr - Visualise, document and explore your software architecture. `https://www.structurizr.com/`. [Online; Abgerufen am 23.10.2016].

[27] Gysel Michael and Kölbener Lukas. Service Cutter. `https://github.com/ServiceCutter/ServiceCutter`, 2015. [Online; Abgerufen am 31.10.2016].

[28] Gysel Michael and Kölbener Lukas. Service Cutter Tutorial. `https://servicecutter.github.io/`, 2015. [Online; Abgerufen am 31.10.2016].

[29] Gysel Michael, Kölbener Lukas, Zimmermann Olaf, and Giersche Wolfgang. Service Cutter - A Structured Way to Service Decomposition. Bachelor Thesis, University of Applied Sciences HSR, Rapperswil Switzerland, 2015.

[30] Michael Stal. Software Architecture Refactoring, OOPSA 2007 Tutorial 1. `http://www.sigs.de/download/oop_08/Stal%20Mi3-4.pdf`, 2008.

[31] mockito.org. Mockito - Tasty mocking framework for unit tests in Java. `http://site.mockito.org/`, 2016. [Online; Abgerufen am 10.12.2016].

[32] OASIS Open. 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`, 2013. [Online; Abgerufen am 03.12.2016].

[33] 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].

[34] S. W. Oh and H. S. Kim. Decentralized access permission control using resource-oriented architecture for the web of things. In *16th International Conference on Advanced Communication Technology*, pages 749–753, Feb 2014.

[35] Opensource.org. Apache License, Version 2.0. `https://opensource.org/licenses/Apache-2.0`, 2004. [Online; Abgerufen am 28.11.2016].

[36] Oracle Corporation and/or its affiliates. Java API for RESTful Services (JAX-RS). `https://jax-rs-spec.java.net/`, 2016. [Online; Abgerufen am 06.12.2016].

[37] The Open Web Application Security Projekt OWASP. Guide to Cryptography. `https://www.owasp.org/index.php/Guide_to_Cryptography`. [Online; Abgerufen

am 27.10.2016].

[38] 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.

[39] Red Hat, Inc. JBoss Projects. `http://www.jboss.org/`, 2016. [Online; Abgerufen am 06.12.2016].

[40] Ravi Sandhu. The authorization leap from rights to attributes: Maturation or chaos? In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, SACMAT '12, pages 69–70, New York, NY, USA, 2012. ACM.

[41] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad. *Security Patterns - Integrating Security and Systems Engineering*. John Wiley & Sons Ltd, 2006.

[42] Sun Microsystems, Inc. Sun's XACML Implementation. `http://sunxacml.sourceforge.net/`, 2016. [Online; Abgerufen am 19.12.2016].

[43] Swagger.io. Swagger. `http://swagger.io/`, 2016. [Online; Abgerufen am 12.11.2016].

[44] The Eclipse Foundation. Jetty - Servlet Engine and HTTP Server. `http://www.eclipse.org/jetty/`, 2016. [Online; Abgerufen am 06.12.2016].

[45] Vaughn Vernon. *Implementing Domain-Driven Design*. Pearson Education, Inc., New Jersey, USA, 2013.

[46] Scott Millett with Nick Tune. *Patterns, Principles, and Practices of Domain-Driven Design*. John Wiley & Sons, Inc., Indianapolis, Indiana, USA, 2015.

[47] Wolfgang Giersche. APPLICATION SECURITY - FROM A DEVELOPER'S POINT OF VIEW). `https://github.com/smurve/hsr2015/raw/master/SpringSecurity/Security.pdf`, 2015. [Vorlesungsfolien auf Github; Abgerufen am 06.12.2016].

[48] Eric Yuan and Jin Tong. Attributed based access control (abac) for web services. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '05, pages 561–569, Washington, DC, USA, 2005. IEEE Computer Society.

[49] Olaf Zimmermann. Architectural refactoring for the cloud: a decision-centric view on cloud migration. *Computing*, pages 1–17, 2016.

[50] Olaf Zimmermann. Architectural Refactoring for the Cloud (ARC). `https://www.ifs.hsr.ch/index.php?id=12044&L=4`, 2016. [Online; Abgerufen am 05.12.2016].

[51] Olaf Zimmermann, Mark Tomlinson, and Stefan Peuser. *Perspectives on Web Services - Applying SOAP, WSDL and UDDI to Real-World Projects*. Springer-Verlag Berlin Heidelberg, 2003.

Glossar

ABAC Attribute-based access control (ABAC) ist ein Zugriffskontroll-Paradigma bei welchem Zugriffsrechte über die Attribute der zu schützenden Objekte und der Benutzer kontrolliert werden. 2, 6, 7, 11–14, 18, 23, 25–27, 29, 33, 47, 48, 53, 54, 103, 105, 106, 108, 109, 112

Architectural Refactoring Ein Architectural Refactoring schlägt eine Architekturumbaumassnahme vor, um bestimmten Smells welche im System vorhanden sind, entgegenzuwirken. 2

Attribut-Repository Das *Attribut-Respository* ist ein Begriff aus dem ABAC-Paradigma. Es bezeichnet eine Datenbank oder System, aus welchem sich der *Policy Decision Point (PDP)* die nötigen Attribute bezieht um eine Autorisierungsentscheidung zu treffen. 7, 18, 24, 54

BO Ein BO ist ein Business Objekt in SYRIUS. 1, 106

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. 106

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. 4, 106

JEE 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. 3, 106

LDAP LDAP ist ein Verzeichnisdienst welcher im Kontext dieses Projekts als zentrale Benutzerverwaltung verwendet wird . 6, 8, 24, 43, 106

Objektschutz *Objektschutz* ist die Bezeichnung der aktuellen Berechtigungslösung in der Persistenzschicht von SYRIUS, welche den Schutz der BO's sicherstellt. 4, 5, 15, 34, 94

Policy Decision Point (PDP) Der *Policy Decision Point* ist das System welches die Autorisierung durchführt bzw. die Berechtigungsregeln evaluiert und daraus eine Entscheidung trifft. Die Systeme welche den PDP aufrufen, werden in dieser Terminologie Policy Enforcement Point (PEP) genannt. Im Kontext dieser Studienarbeit ist der Prototyp des neuen Autorisierungssystems der *Policy Decision Point*. 40, 42, 76, 103, 104

Policy Enforcement Point (PEP) Der *Policy Enforcement Point* ist ein System welches eine Autorisierungsentscheidung von einem Policy Decision Point (PDP) erhalten hat und diese Entscheidung durchsetzen muss. Er ist dafür verantwortlich das die Entscheidung des PDP in der Applikation eingehalten wird. Im Kontext dieser Studienarbeit ist SYRIUS (Persistenzschicht) als Aufrufer des Autorisierungssystems der *Policy Enforcement Point*. 42, 104

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. 2, 10–12, 14, 25, 26, 33, 106, 112

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. 4, 106

Schutzdefinition Die *Schutzdefinition* ist ein Konfigurationsobjekt in SYRIUS, welches für bestimmte BO's die Berechtigungen festlegt. 4, 5, 13

Schutzobjekt 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 BO's innerhalb des aktuellen Schutzpfades. Die anderen BO's im Schutzpfad beziehen die Berechtigungskonfiguration über dieses Schutzobjekt. 4, 5, 13, 105

Schutzpfad 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 Schutzobjekt. 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 BO's das Schutzobjekt auch indirekt über diverse andere BO's referenzieren können, wird bei Adcubum von einem Schutzpfad gesprochen. Dabei muss sichergestellt werden dass das Schutzobjekt für ein BO eindeutig ermittelt werden kann. 4, 18, 104

SOAP SOAP (ursprünglich für Simple Object Access Protocol) ist ein Netzwerkprotokoll, mit dessen Hilfe Daten zwischen Systemen ausgetauscht und Remote Procedure Calls durchgeführt werden können. SOAP ist ein industrieller Standard des World Wide Web Consortiums (W3C). 19, 32, 33, 107

UTC UTC steht für 'Ultra-Thin Client' und bezeichnet eine Präsentationskomponente in Adcubum SYRIUS. 4, 16, 19, 107

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. 14, 29, 31, 107, 111

Abkürzungsverzeichnis

ABAC Attribute-based access control, siehe Glossar: ABAC. 2, 6, 7, 11–14, 18, 23, 25–27, 29, 33, 47, 48, 53, 54, 103, 105, 108, 109, 112

AOP Aspektorientierte Programmierung. 34

API Application Programming Interface (Programmierschnittstelle). 75

BO Business Objekt, siehe Glossar: BO. 1, 2, 4, 13, 15, 16, 18, 19, 22, 34, 36, 38–44, 48, 78, 104, 105

DDD Domain-Driven Design. 7, 38

EJB Enterprise JavaBean. Siehe Glossar: EJB. 3, 108

HOPP Half Object plus Protocol. 4

IIOP Internet Inter-Orb Protocol. Siehe Glossar: IIOP. 4

JEE Java Platform, Enterprise Edition: JEE. 3

LDAP Lightweight Directory Access Protocol, siehe Glossar: LDAP. 6, 8, 24, 43

NFA Nichtfunktionale Anforderung. 20, 31, 32, 50, 51

OWASP Open Web Application Security Project. 20

RBAC Role-based access control, siehe Glossar: RBAC. 2, 10–12, 14, 25, 26, 33, 112

RMI Remote Method Invocation. Siehe Glossar: RMI. 4

RPC Remote Procedure Call (Aufruf einer fernen Prozedur). 32

SOAP Simple Object Access Protocol, siehe Glossar: SOAP. 19, 32, 33

UI User Interface. 4

UTC Ultra-Thin Client. Siehe Glossar: UTC. 4, 16, 19

XACML eXtensible Access Control Markup Language, siehe Glossar: XACML. 14, 29, 31, 111

Abbildungsverzeichnis

Auflistungsverzeichnis

Tabellenverzeichnis