Advanced Patterns And Frameworks

Zusammenfassung & Notizen

Hochschule für Technik Rapperswil
Frühjahressemester 2013

Autoren Manuel Alabor (MAL)

Inhaltsverzeichnis

1. Access Control Models 3
   1.1. Authorization 3
   1.2. Role Based Access Control 5

A. Abbildungen, Tabellen & Quellcodes 8

B. Workshops 9

Kapitel 1 **Access Control Models**

1.1. Authorization

Das Authorization Pattern beschreibt auf einfache Art und Weise die Zugriffsberechtigungen eines Subjekts auf ein bestimmtes Objekt. Es spezifiziert zudem die Art des erlaubten Zugriffes (Lesend, schreibend etc.)

Kontext

Jegliche Umgebungen in denen der Zugriff auf enthaltene Objekte kontrolliert werden muss.

Problem

In einer kontrollierten Umgebung muss sichergestellt werden, dass nur berechtigte Subjekte auf entsprechende Objekte zugreifen können. Es stellt sich also die Herausforderung, diese Information losgelöst von den eigentlichen Objekte abzulegen. Dabei soll aber eine gewisse Flexibilität bei der Definition von Berechtigungen, Objekten und Subjekten erhalten bleiben.

Des weiteren sollen diese Informationen so einfach wie möglich im Nachhinein änderbar sein.

Lösung

Strukturell fällt die Lösung zum Authorization Pattern relativ simpel aus:

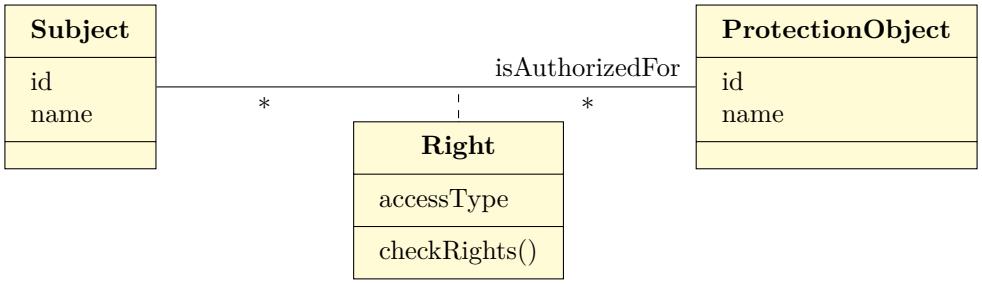


Abbildung 1.1.: Authorization

- Subject beschreibt jegliche Aspekte des zu berechtigenden Subjekts
- Das ProtectionObject ist das zu schützende Objekte
- Right enthält alle Informationen, wie Subject auf ProtectioObject zugriefen darf/-kann

Erweiterungen

Die vorgestellte Struktur kann um komplexere Aspekte erweitert werden. So kann bspw. mittels einem "Copy"-Flag eine Stellvertretung eines Subjektes durch ein anderes ermöglicht werden. Weiter ist die Verwendung eines Prädikats denkbar, welches eine Regel mit zusätzlicher "Intelligenz" austatten kann (-> "Darf nur zugreifen wenn Zeit innerhalb Arbeitszeit")

Diese Anpassungen können direkt auf dem Rights-Objekt modelliert werden.

Vor- & Nachteile

- Durch seine Offen- und Allgemeinheit kann dieses Pattern auf jegliche Umgebung appliziert werden (Filesysteme, Organistaitonsstrukturen, Zugangskontrollen etc.)
- In der beschriebenen Form sind administrative Aufgaben (Änderung der Zugriffsrechte) nicht gesondert definiert. Für bessere Sicherheit ist dies jedoch von Vorteil
- Für viele Subjekte/Objekte müssen entsprechend viele Berechtigungsregeln erfasst und auch verwaltet werden
- Viele Regeln machen die Verwaltung für einen Administrator zu einer heiklen Aufgabe (Verkettung von Berechtigungen etc.)

Beispielanwendungen

- Dateisysteme
- Firewalls greifen teilweise auf dieses Pattern zurück, um Regeln für den analysierten Traffic zu modellieren

1.2. Role Based Access Control

Diese Pattern basiert stark auf dem Authorization Pattern und versucht dessen Nachteile durch einen zusätzlichen Abstraktionslayer auszugleichen. Das ”Role Based Access Control” Pattern definiert Berechtigungen nicht direkt auf Stufe der Subjekte, sondern versucht diese in Gruppen (Aufgabenbereiche, Kaderposition, Arbeitsort etc.) einzuteilen und anschliessend auf dieser Ebene quasi übergeordnet zu berechtigen.

Kontext

Eine Umgebung mit vielen Objekten und Subjekten. Deren Berechtigungen ändern häufig. Zudem ist damit zu rechnen dass eben so oft neue Subjekte und Objekte hinzukommen oder wieder wegfallen.

Problem

Die Rechteverwaltung in dem beschriebenen Kontext generiert einen hohen administrativen Aufwand. Um die Anzahl individueller Berechtigungen zu minimieren soll versucht werden, alle Subjekte in Gruppen einzuteilen. Die Einteilung basiert darauf, dass Subjekte mit ähnlichen Aufgaben zumeist auch ähnliche oder identische Berechtigungen benötigen. Trotzdem sollen die Berechtigungen weiterhin präzise abgebildet werden können (”Need to know”).

Lösung

Organisationen bieten normalerweise bereits mehr oder weniger wohldefinierte Gruppenstrukturen (Abteilungen, Aufgabenbereiche). Ein gutes Sicherheitskonzept sollte bestrebt sein, dass jedes Subjekt genau auf die Objekte Zugriff hat, mit welchen es täglich arbeitet (wiederum ”Need to know”).

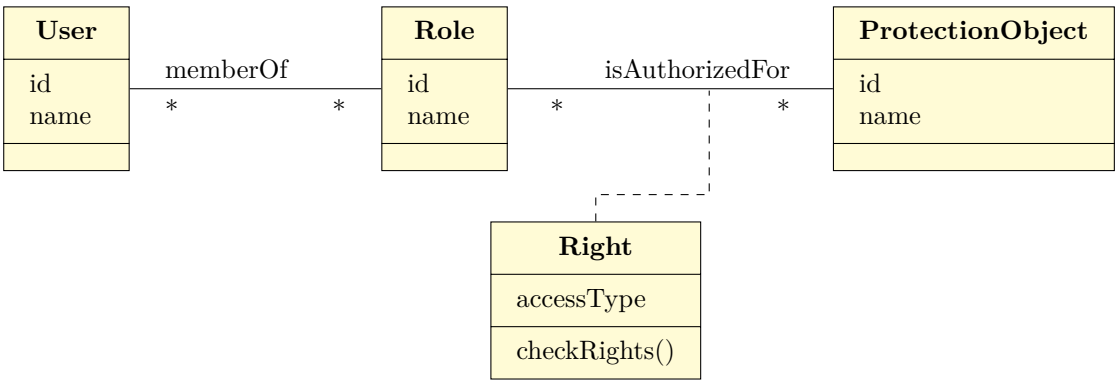


Abbildung 1.2.: Basic Role Based Access Control

Im Vergleich zum Authorization Pattern kommt lediglich ein neues Element hinzu: Die Role fasst mehrere User (Subjekte) zu einer Menge zusammen und berechtigt sie über Right für ein spezifisches ProtectionObject.

Erweiterungen

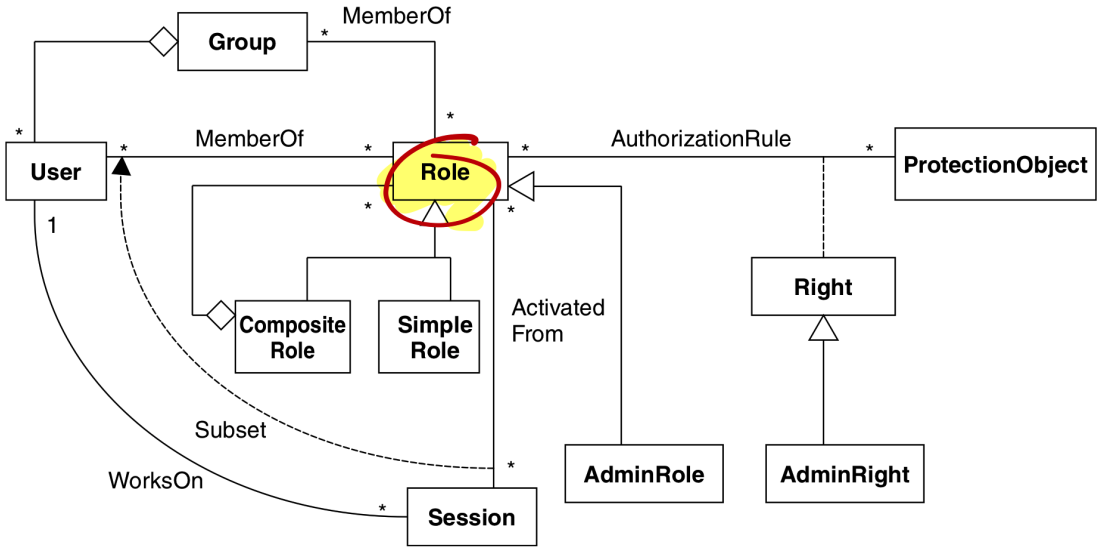


Abbildung 1.3.: RBAC mit Composite, Admins & Abstract Session

Composite Pattern

Statt einer simplen Assoziation zwischen User und Role könnte auch mit dem Composite-Pattern gearbeitet werden, um diese Abhängigkeit zu modellieren.

Administration

Wie ebenfalls bereits im Authorization-Pattern erwähnt kann auch dieses Modell zielgerichtet um Administrations-Elemente erweitert werden. Auf diese Weise kann zusätzliche Klarheit im System geschaffen werden, wer genau für was zuständig ist.

Abstract Session

Um die Möglichkeiten auf die Spitze zu treiben, sei hier auch das Abstract Session Pattern erwähnt: Die Abhängigkeit einer Session kann so direkt ins Security Modell ”miteinmodelliert” werden.

Vor- & Nachteile

- Die Zusammenfassung zu Gruppen ermöglicht eine vereinfachte Administration der gesamthaft vorhandenen Berechtigungen
- Veränderungen in der realen Organistaionstruktur (Neuzugänge, Abgänge, Jobwechsel etc.) können einfacher auf das Sicherheitskonzept abgebildet werden
- Ein Subjekt kann durch mehrere Sessions verschiedene Funktionen auf einmal wahrnehmen
- Theoretisch können Gruppen wiederum in Gruppen zusammengefasst werden (Yay, even more complexity...)
- Konzeptionelle Komplexität nimmt durch die neuen Elemente wiederum zu!

Beispielanwendungen

- Windows 2000 Rights Management (Group Policies)

Anhang A **Abbildungen, Tabellen & Quellcodes**

Abbildungsverzeichnis

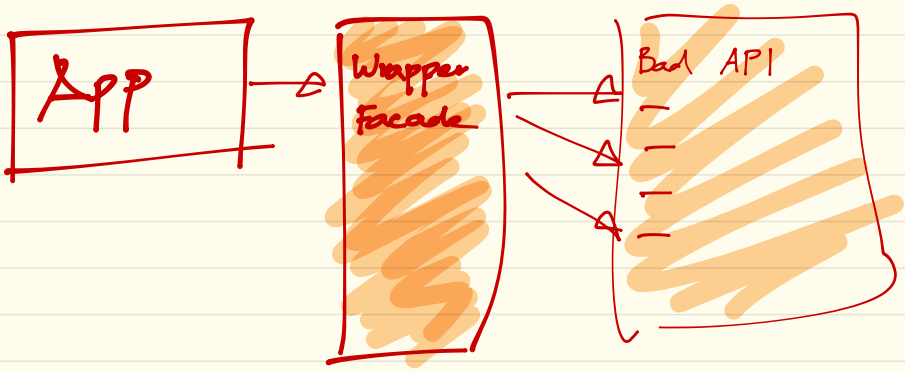
1.1. Authorization 4
1.2. Basic Role Based Access Control 5
1.3. RBAC mit Composite, Admins & Abstract Session 6

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang B **Workshops**

Wrapper Facade



- Kann auch mehr Logik enthalten
- Typensicherheit
- Legacy-Code verpacken für "Heute"
- Nachteile:
  Performance kann zum Problem werden
- Vorteile: Bessere API's
- Knackpunkte:
  Wrapper soll semantisch korrekt bleiben (zusammen was zusammen gehört

Anmerkungen P. Sommerlad:
- Error-Handling ist wichtig ⇒ Auch hier wrappen!
  ↳ Falls nötig Domain-spezifische Errors
- Wann nicht anwendbar?
  - Wrapper vom Wrapper vom Wrapper
- Stichworte:
  Async vs. Sync (Async ist schneller, da Buffer kopieren evtl. gespart werden kann)

Fault Tolerance Systems

Introduction: Zusammenhang Fault, Error & Failure
Fault: Bug, Ursache
Error: Zustand
Failure Effektives Problem
↳ Zu vermeidendes Problem

- Failure definieren sich im Normalfall durch Abweichung von der Spec

- Unterschiedliche Faults können zu gleichen Errors/Failures führen
- Coverage: Wahrscheinlichkeit dass sich ein System innert gegebener Zeit wieder erholen kann: Mean Time To Failure + Mean Time To Recover } Mean Time Between Failure

↳ Relaibility: $e^{-\frac{1}{MTTF}}$

- FIT: $\frac{\#\text{Failures}}{1 \cdot 10^9 h}$ ⇒ Failures in Time

⇒ Stichwort: Server-Zuverlässigkeit

Fail Silent: Bei Fehler übernimmt automatisch andere Komponente
Fail Consistency: Man muss herausfinden welche Systemkomponenten fehlerhaft sind
Malicious Failure: Man kann nicht einfach herausfinden welche Systeme fehlerhaft sind ⇒ Byzantinische Generäle zur Abstimmung