HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

HERAS^{AF}
PEP

Spring Security Integration mit HERAS^{AF}

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2010

Autor(en):	Daniel Bobst, Sandro Brändli
Betreuer:	René Eggenschwiler
Projektpartner:	Projekt HERAS^{AF}
Experte:	Wolfgang Giersche
Gegenleser:	Thomas Letsch

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

Kurzfassung der Studienarbeit

Abteilung	**Informatik**
Namen der Studierenden	**Daniel Bobst** **Sandro Brändli**
Studienjahr	**HS 2010**
Titel der Studienarbeit	**Spring Security Integration mit HERAS^{AF}**
Examinatorin / Examinator	**Wolfgang Giersche**
Themengebiet	**Internet-Technologien und -Anwendungen**
Projektpartner	**Projekt HERAS^{AF}**
Institut	**Institut für Internet-Technologien und -Anwendungen**

Kurzfassung

Als Teil des weitverbreiteten Spring Frameworks unterstützt Spring Security die Absicherung von Unternehmensapplikationen durch Autorisierung von Zugriffen auf technische Ressourcen. Allerdings werden die dafür benötigten Regeln normalerweise dezentral im Quellcode definiert und es wird durch die Architektur von Spring Security ein Rollen oder Permission-basiertes Sicherheitskonzept vorgeschrieben.

Eine elegantere Lösung wäre das gänzliche Auslagern der Entscheidung in eine externe Komponente. Im zu schützenden Code selbst werden die verfügbaren Informationen gesammelt, aufbereitet und die Entscheidungsfindung an diese externe Komponente delegiert. Eine solche Komponente bietet HERAS^{AF}, ein am ITA entwickeltes Open Source Projekt. Basierend auf dem XACML 2.0 Standard erlaubt HERAS^{AF} die Definition und zentrale Verwaltung von beliebig komplexen formalen Richtlinien. Es lassen sich weitaus differenziertere Zusammenhänge zwischen Benutzern und Ressourcen modellieren. Zugriffsregeln können von der Codebasis entkoppelt und während der Applikationslaufzeit statt der Entwicklung bestimmt werden.

Das Ziel dieser Studienarbeit war es, durch Spring Security abgesicherte Applikationen in die Lage zu versetzen, die eigentliche Entscheidung an einen HERAS^{AF} Policy Decision Point (PDP) zu delegieren.

Unsere Lösung erweitert Spring Security um die Möglichkeit, Methoden mittels spezieller Expressions abzusichern, die den Zugriffskontext möglichst neutral erfassen und die Zugriffsentscheidung einem zentralen PDP zu übergeben.

Die im Rahmen dieser Studienarbeit entwickelte Lösung ist vielfältig erweiterbar. Denkbar wäre zusätzlich zu Expressions die Unterstützung von Annotationen, Absicherung auf Klassenebene, Unterstützung von Websicherheit oder die Integration mit anderen Frameworks als Spring.

Spring Security Integration mit HERAS^{AF}

Technischer Bericht

Sandro Brändli
Daniel Bobst

Verantwortlicher:
Wolfgang Giersche

Betreuer:
René Eggenschwiler

Versionsgeschichte

Version	Beschreibung	Datum	Autor
1.0	Initialversion und Grundstruktur	06.10.10	sb
1.1	Beginn Arbeitspakete	13.10.10	sb
1.2	Aktualisierung Arbeitspakete	16.10.10	sb
1.3	Beginn Business Analyse	27.10.10	sb
1.4	Beginn Spring Analyse	28.10.10	db
1.5	Beginn Beschreibung Architektur und Design	29.10.10	sb
1.6	Aktualisieren und überarbeiten	26.11.10	sb
1.7	Ausarbeitung Design Beschreibung	03.12.10	db
1.8	Allgemein aktualisieren und überarbeiten	06.12.10	db
1.9	Analyse überarbeitet	10.12.10	sb
2.0	Aktualisieren und überarbeiten	23.12.10	db, sb

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Code Listings

1 Einführung

Dieses Kapitel bietet eine Einführung in die Problemstellung sowie eine Beschreibung der Gliederung dieses Dokuments. Eine detaillierte Analyse der Problemstellung ist im Kapitel 2 zu finden.

1.1 Aufgabenstellung

Die Studienarbeit dreht sich um das Spring Security Framework und um HERAS[AF].

HERAS[AF] ist ein Framework zur Policy-basierten Autorisierung. Spring Security ist ein Framework, welches zur Authentifizierung, Autorisierung und Durchsetzung der Security in Java-Applikationen verwendet wird. Auf HERAS[AF] und Spring Security wird in der Analyse (Kap. 2) genauer eingegangen.

Im Rahmen der Studienarbeit soll es ermöglicht werden, bei mit Spring Security abgesicherten Applikationen HERAS[AF] für die Autorisierung einzusetzen.

Die Aufgabenstellung ist detailliert im Dokument [AUFG] erläutert. Die abzugebenden Dokumente sind im Dokument [ANLEI] beschrieben.

Im Dokument [BEUR] sind die Bewertungskriterien für Studienarbeiten aufgeführt.

1.2 Dokumentstruktur

Folgende Dokumente müssen separat abgeliefert werden:

- Abstract
- Poster
- Getting Started

Die restlichen Teile der Dokumentation werden in Absprache mit den Betreuern in diesem Dokument untergebracht.

Dieses Dokument gliedert sich wie folgt:

- Kapitel 1 - Einleitung:
  Die Einleitung beinhaltet eine Einführung, die Aufgabenstellung und die Beschreibung der Dokumentstruktur.
- Kapitel 2 - Analyse:
  Es wird das Problemumfeld analysiert, dies umfasst das Spring Security Framework, HERAS[AF] sowie die darunterliegende eXtensible Access Control Markup Language (XACML). Ebenso wird der Umfang der Arbeit festgelegt.
- Kapitel 3 - Architektur:
  In diesem Kapitel werden die Komponenten des Policy Enforcement Points (PEP) beschrieben. Ausserdem wird erklärt, wie der PEP sich in Spring Security integriert.
- Kapitel 4 - Design:
  Beschreibt den PEP und seine Klassen im Detail.

- Kapitel 5 - Ausblick:
  Eine kurze Diskussion über mögliche Richtungen, in die diese Arbeit erweitert werden könnte.
- Kapitel 6 - Anhang:
  Beinhaltet Informationen zum Projektmanagement, alle Sitzungsprotokolle, Erfahrungsberichte, referenzierte Dokumente sowie die Erklärung über unabhängiges Arbeiten

2 Analyse

Die Studienarbeit befasst sich mit der Absicherung von Applikationen. Unter Absicherung wird der über Berechtigungen geregelte Zugriff auf Ressourcen verstanden. Eine Ressource ist dabei ein abstrakter Begriff für etwas Schützenswertes innerhalb der Applikation.

Der Fokus der Studienarbeit liegt auf Java-Applikationen. HERAS^{AF} selbst ist in Java entwickelt und wurde für den Einsatz in Enterprise-Applikationen konzipiert.

Security kann in Applikationen auf verschiedene Arten durchgesetzt werden:

- Die Applikation kann vor dem Zugriff auf eine Ressource die Berechtigungen überprüfen
- Die Ressource selbst kann prüfen, ob auf sie zugegriffen werden darf
- Das Security Framework kann sich transparent zwischen Aufrufer und Ressource schalten

Bei dieser Arbeit ist die schützenswerte Ressource eine Business-Methode. Die Sicherheit wird durchgesetzt, indem sich das Framework zwischen Aufrufer und Ressource schaltet (Interception).

Als Framework für die Interception wird Spring Security verwendet. Spring im Allgemeinen und Spring Security für die Absicherung sind im Bereich Enterprise-Java weit verbreitet.

Das Ziel der Studienarbeit ist es, Spring Security dazu zu bringen, dass es die Entscheidung, ob ein Zugriff gewährt oder verweigert werden soll, an HERAS^{AF} delegiert.

2.1 Zielarchitektur

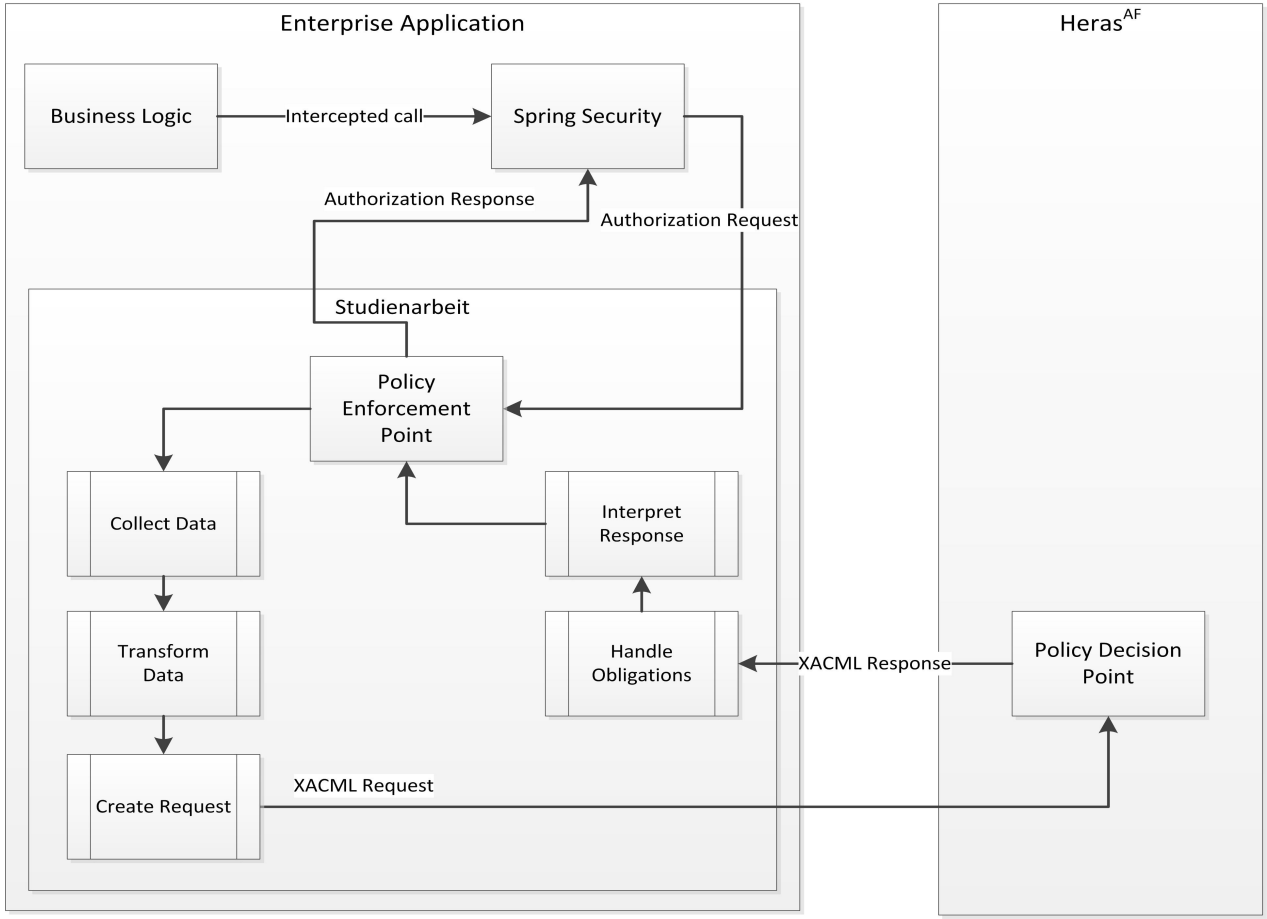


Abbildung 1: Zielarchitektur

Das Diagramm zeigt eine Grobübersicht, wo die Arbeit einzuordnen ist. Der mit "Studienarbeit" gekennzeichnete Teil soll implementiert werden. Er stellt das Verbindungsstück von Spring Security zu HERAS^{AF} dar.

Der Ablauf einer Anfrage gemäss Grafik:

1. Der Zugriff auf eine Ressource (z.B. Methode) wird mittels Spring Security abgefangen.
2. Spring Security leitet die Informationen über den abgefangenen Zugriff an den Policy Enforcement Point (PEP) weiter.
3. Der PEP sammelt daraus die für die Autorisierung relevanten Daten.
4. Die Daten werden in eine applikationsunabhängige Form transformiert.
5. Es wird ein Request erzeugt und an den Policy Decision Point (PDP) gesendet.
6. Der PDP wertet den Request aus, und liefert eine entsprechende Antwort zurück.
7. Falls die Antwort Obligations (siehe Kap. 2.3.2) enthält werden diese abgearbeitet.
8. Die Antwort des PDP wird durch den PEP interpretiert und die Entscheidung ob der Zugriff stattfinden darf an Spring Security zurückgeliefert.

2.2 Spring Security

Spring[1] ist ein OpenSource-Framework, welches in Enterprise-Applikationen häufig benutzte Funktionen zur Verfügung stellt und die Integration mit anderen Enterprise-Frameworks erleichtert.

Spring bietet unter anderem auch die Möglichkeit der aspektorientierten Programmierung (Spring AOP). Es ist damit möglich, sich vor oder nach einem Methodenaufruf einzuklinken und eigenen Code auszuführen.

Spring Security[2] ist ein Teil des Spring-Frameworks und kümmert sich um die Authentisierung, Autorisierung sowie Definition und Durchsetzung der Berechtigungen. Um Methodenaufrufe abzusichern bedient sich Spring Security der Methoden von Spring AOP.

1 http://www.springsource.org/
2 http://static.springsource.org/spring-security/site/index.html

Nachfolgend wird der Aufruf einer mittels Spring Security abgesicherten Methode untersucht:

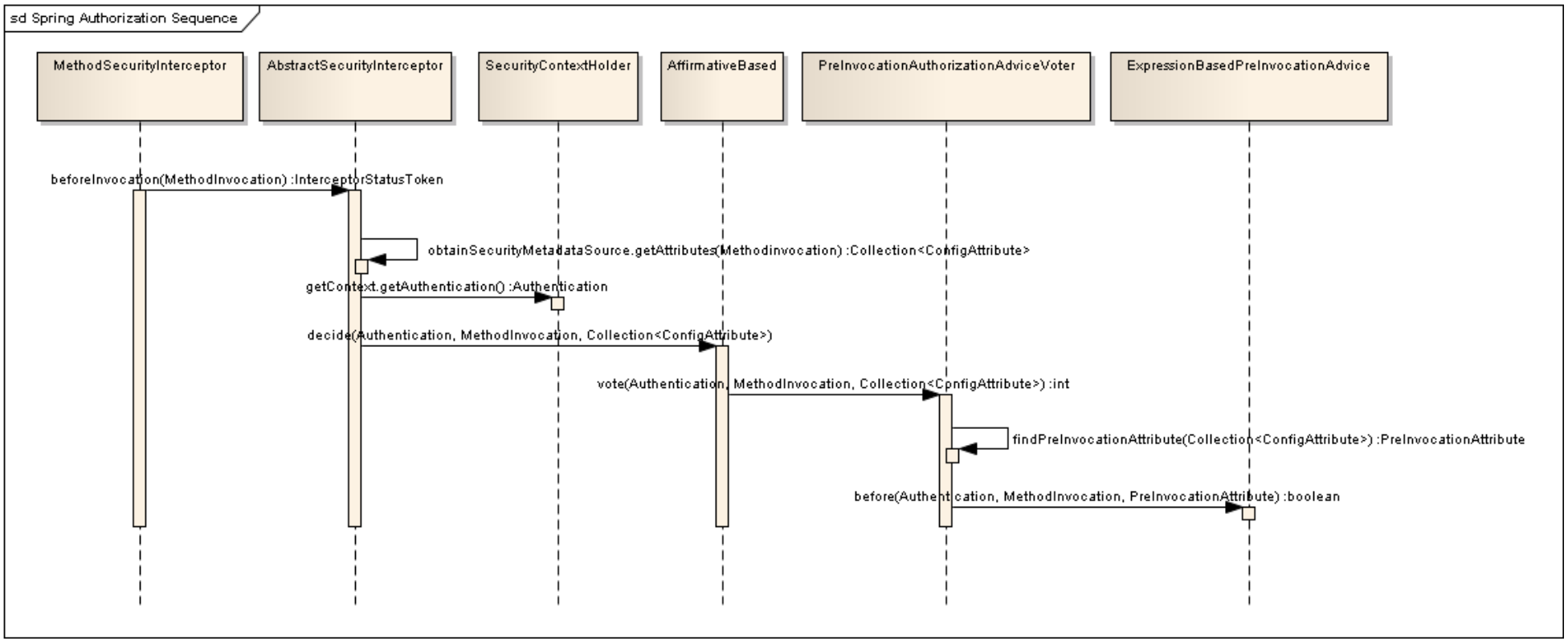


Abbildung 2: Sequenzdiagramm Authentifizierung und Autorisierung durch Spring Security

Der Aufruf einer mittels Spring Security abgesicherten Methode läuft folgendermassen ab:

1. Vor dem Aufruf der Methode wird der *MethodSecurityInterceptor* von Spring Security aufgerufen.
2. Er holt die *Authentication* des aktuellen Threads und fragt den AccessDecisionManager (siehe Kapitel 3.1.2) ob der Zugriff erlaubt werden soll. In diesem Fall handelt es sich um den *AffirmativeBased-AccessDecisionManager*.
3. Der *AccessDecisionManager* fragt die *Voter* (siehe Kapitel 3.1.3), in diesem Fall *PreInvocationAuthorizationAdviceVoter*, ob der Zugriff gestattet werden soll.
4. Der *Voter* wertet die Autorisierungs-Expression (siehe Kapitel 3.1.4) mittels der *ExpressionBasedPreInvocationAdvice* aus.

2.2.1 Methodenaufruf

Spring Security wird durch die Annotation *@PreAuthorize* angewiesen, Zugriffe auf die annotierte Methode zu unterbrechen und vor dem Aufruf eine Sicherheitsprüfung durchzuführen. Dies wird mittels eines dynamischen Proxies[3] oder mittels Byte-Code-Manipulation[4] realisiert. Der Annotation wird eine EL-Expression mitgegeben, welche von Spring Security evaluiert wird. Anhand der Evaluation wird entschieden, ob der Methodenaufruf erlaubt wird oder nicht.

2.2.2 Interception

Der Methodenaufruf wird durch einen *MethodSecurityInterceptor* abgefangen. Dieser gibt den Aufruf in Form einer *MethodInvocation* an seine Superklasse *AbstractSecurityInterceptor* weiter.

In der MethodInvocation enthalten sind unter anderem die Argumente der Methode, der Methodenname sowie der Name der die Methode enthaltenden Klasse.

Der *AbstractSecurityInterceptor* extrahiert aus der MethodInvocation die in der Annotation der Methode angegebene Expression und präpariert sie in Form einer Sammlung von *ConfigAttribute* Objekten für die Weitergabe an die Evaluierungsstelle.

2.2.3 Authentifizierung

Im ThreadLocal namens *SecurityContextHolder* liegen nach der Authentifizierung durch Spring Security alle relevanten Informationen über den aufrufenden Benutzer. ([SPREF] 5.2)

Der *AbstractSecurityInterceptor* übergibt die gesammelten Informationen (*Authentication*, *MethodInvocation* und *ConfigAttributes*) zwecks Autorisierung des Aufrufs an einen *AccessDecisionManager*.

2.2.4 Autorisierung

Ein DecisionManager verwaltet den weiteren Ablauf der Autorisierung und schickt die getroffene Entscheidung an den Interceptor zurück, der den Methodenaufruf abgefangen hat. ([SPREF] 5.5)

Spring Security stellt einige Standard-DecisionManager zur Verfügung: *AffirmativeBased*, *ConsensusBased* und *UnanimousBased*. ([SPREF] 13.2)

3 http://download.oracle.com/javase/6/docs/api/java/lang/reflect/Proxy.html
4 http://cglib.sourceforge.net/

Diese DecisionManager entscheiden alle Voting-basiert. Das bedeutet, dass der DecisionManager einen oder mehrere *Voter* (Kap. 2.2.5) konsultiert und deren individuelle Entscheidungen auswertet, um zu einem Entschluss zu kommen. Jeder *Voter* stimmt entweder einer Zulassung des Methodenzugriffs zu, lehnt sie ab oder enthaltet sich.

Standardmässig verwendet Spring Security einen *AffirmativeBased* DecisionManager. Diesem reicht für einen positiven Entscheid, wenn einer oder mehrere seiner Voter dem Zugriff zustimmen.

Im Gegensatz dazu benötigt der *ConsensusBased* DecisionManager einen einstimmigen positiven oder negativen Entscheid aller Voter, die sich nicht enthalten haben.

Der *UnanimousBased* DecisionManager erlaubt einen Zugriff nur, wenn von den Votern, die sich nicht enthalten haben, keiner den Zugriff ablehnt. Die Ablehnung eines Zugriffs hingegen muss nicht einstimmig beschlossen werden.

2.2.5 Voting

Ein Voter erhält von seinem DecisionManager sämtliche über den Aufruf gesammelte Informationen (*Authentication*, *MethodInvocation* und *ConfigAttributes* Objekte) um eine Entscheidung treffen zu können.

Spring Security stellt standardmässig drei Arten von Voter zur Verfügung: *RoleVoter*, *AuthenticationVoter* und *PreInvocationAuthorizationAdviceVoter*. ([SPREF] 13.2)

- *RoleVoter*:
  Der *RoleVoter* erlaubt den Zugriff, falls der Benutzer eine Rolle erfüllt. Um dies zu entscheiden wird das dem Voter übergebene *Authentication* Objekt nach einem Feld mit einem Wert der Form "ROLE_*" geparst. Ist ein solcher Wert vorhanden und entspricht er dem Wert eines der ebenfalls übergebenen *ConfigAttributes*, wird dies als Zeichen gedeutet, dass der Benutzer eine Rolle verkörpert.
- *AuthenticationVoter*:
  Der AuthenticationVoter wird benutzt um zwischen verschiedenen Arten von authentifizierten Benutzern zu unterscheiden und je nach Grad der Authentifizierung unterschiedlich zu entscheiden. Wird ein solcher Voter verwendet, kann ein Benutzer in der Regel voll authentifiziert sein, anonym oder (im Falle eines webbasierten Aufrufs) remember-me-authentifiziert.
- *PreInvocationAuthorizationAdviceVoter*:
  Dieser Voter trifft seine Entscheidung aufgrund der Evaluation der in der Annotation der Methode angegebenen Expression.
  Dazu übergibt sie die erhaltenen Informationen an eine Implementation eines *PreInvocationAuthorizationAdvice*, im analysierten Fall eine Klasse namens *ExpressionBasedPreInvocationAdvice*.

2.2.6 Expression Handling

Der *ExpressionBasedPreInvocationAdvice* verwaltet die Evaluation der beim Methodenaufruf in der Annotation angegebenen Expression.

Nach dem Erstellen eines *EvaluationContexts* und dem Parsen der Expression durch einen *ExpressionParser* wird die eigentliche Evaluation durch eine Implementation eines *PermissionEvaluators* durchgeführt.

Ein *PermissionEvaluator* akzeptiert Expressions der Form *hasPermission(Object)*. Das heisst, das Interface für den *PermissionEvaluator* ist auf diese Signatur limitiert. Leider genügt diese nicht um die für XACML notwendigen Attribute mitzugeben. Beim Argument in der Expression handelt es sich um ein Objekt, das die zu überprüfende Permission darstellt. Ausserdem kennt der Evaluator den Benutzer über ein erhaltenes *Authentication* Objekt sowie die zu schützende Methode.

Details zum Expression Handling siehe ([SPREF] 15.3)

2.3 XACML

XACML (eXtensible Access Control Markup Language) ist ein Standard, welcher in [XSPEC] definiert ist. XACML dient der Policy-basierten Autorisierung.

Dies bedeutet, dass im Unterschied zu Security Frameworks ohne XACML die Zugriffsregeln nicht im Code definiert, sondern in Policies hinterlegt werden.

Ausserdem lassen sich mittels der Policies beliebige Zugriffsregeln definieren. XACML schreibt kein bestimmtes Sicherheitskonzept vor. Im Gegensatz zu Spring Security, welches rollenbasierte Autorisierung vorschreibt, sind bei XACML Rollen nicht notwendig. Es ist aber auch mit XACML möglich Rollen zu verwenden.

Normalerweise sieht eine mittels Spring Security abgesicherte Methode folgendermassen aus:

```
@PreAuthorize("hasAuthority('ROLE_ACCOUNTANT')")
public Account post(Account account, double amount);
```

Listing 1: Spring Method Security-Beispiel

Der obige Code bestimmt, dass nur Benutzer mit der Rolle "ROLE_ACCOUNTANT" auf die Methode *post()* zugreifen dürfen. Die Zugriffsregel ist im Code definiert.

Die Zugriffskontrolle wird bei XACML in zwei Schritte aufgeteilt. Einerseits müssen die zur Entscheidung relevanten Information gesammelt und dem PDP übermittelt werden. Im zweiten Schritt wird aufgrund der übermittelten Daten eine anwendbare Policy gesucht und ausgewertet. Die Zugriffsregeln sind in der Policy definiert.

Der Code um die entscheidungsrelevanten Informationen mitzugeben sieht mittels des in der Studienarbeit erstellten Policy Enforcement Point (PEP) folgendermassen aus:

```
@PreAuthorize("{subjects({'role', {#authentication.authorities}}),"  +
              "resources('method', {'Accounts.post'})}")
public Account post(Account account, double amount);
```

Listing 2: Spring Method Security XACML-Beispiel

Im vorangehenden Codebeispiel ist nicht ersichtlich, unter welchen Bedingungen der Zugriff gewährt wird. Es werden nur die zur Entscheidung notwendigen Daten weitergegeben. Diese sind die Rollen des Benutzers, welcher die Methode aufrufen will, und der Name der Methode. Es könnten noch weitere Daten mitgegeben werden. In diesem Fall wurden nur die Attribute mitgegeben, welche auf die entsprechende Policy gemappt werden. Wie entschieden werden soll, ist in der Policy definiert.
Die Zugriffsregeln lassen sich somit ändern, ohne dass die Applikation neu ausgeliefert werden muss. Dies ist sinnvoll, da sich die Zugriffsregeln normalerweise schneller ändern als neue Releases einer Applikation entwickelt werden. Deshalb werden die Zugriffsregeln vom Releasezyklus entkoppelt. Weil es sich bei den Zugriffsregeln um einen fachlichen Aspekt handelt, sollten sie nicht im Code definiert werden.

Die entsprechende Policy würde etwa aussehen (vereinfacht):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
  PolicyId="GrantMedicalPersonnelRead" Version="3.0"
  RuleCombiningAlgId=
  "urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          Accounts.post
          </AttributeValue>
          <ResourceAttributeDesignator
            AttributeId="method"
            DataType="http://www.w3.org/2001/XMLSchema#string"
            MustBePresent="true" />
        </ResourceMatch>
      </Resource>
    </Resources>
  </Target>
  <Rule RuleId="DenyByDefault" Effect="Deny">
    <Target />
  </Rule>
  <Rule RuleId="GrantReadAccess" Effect="Permit">
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">
            ROLE_ACCOUNTANT
            </AttributeValue>
            <SubjectAttributeDesignator
              AttributeId="role"
              DataType="http://www.w3.org/2001/XMLSchema#string"
              MustBePresent="true" />
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
  </Rule>
</Policy>
```

Listing 3: XACML Policy-Beispiel

Diese Policy ist auf das obige Aufrufbeispiel anwendbar, da als Methodenname "Accounts.post" mitgegeben wurde und diese Methode als Target der Policy definiert wurde. Standardmässig wird der Zugriff verweigert. Als Kriterium für die Zulassung des Zugriffs ist definiert, dass der Benuter die Rolle "ROLE_ACCOUNTANT" haben muss.
Mit diesem Beispiel wurde Rollen-basierte Autorisierung auf XACML abgebildet. Es liessen sich auch beliebige andere Kriterien in der Policy definieren.

2.3.1 Requests

Der Request besteht aus den Komponenten Subject, Resource, Action und Environment. HERAS[AF] bietet bereits die notwendige Infrastruktur um Requests zu erstellen. Es müssen nur die Daten gesammelt und in den richtigen Datenstrukturen bereitgestellt werden.

Um eine Berechtigungsanfrage zu stellen, muss ein XACML-Request erzeugt werden. Ein XACML-Request sieht beispielsweise folgendermassen aus:

```
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
  xmlns:ns2="urn:oasis:names:tc:xacml:2.0:policy:schema:os">
  <Subject>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:herasaf:user:role">
      <AttributeValue>ROLE_USER</AttributeValue>
    </Attribute>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:herasaf:user:role">
      <AttributeValue>ROLE_SUPERVISOR</AttributeValue>
    </Attribute>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:herasaf:subject:username">
      <AttributeValue>test</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:herasaf:method">
      <AttributeValue>Invocation:roleSecuredMethod</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:herasaf:type">
      <AttributeValue>read</AttributeValue>
    </Attribute>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:herasaf:type">
      <AttributeValue>exec</AttributeValue>
    </Attribute>
  </Action>
  <Environment>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#time"
      AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time">
      <AttributeValue>15:42:59.785+02:00</AttributeValue>
    </Attribute>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#date"
      AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-date">
      <AttributeValue>2010-10-27+02:00</AttributeValue>
    </Attribute>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#dateTime"
      AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-dateTime">
      <AttributeValue>2010-10-27T15:42:59.785+02:00</AttributeValue>
    </Attribute>
  </Environment>
</Request>
```

Listing 4: XACML-Request

Der Request wird an den PDP geschickt. Es gibt vier verschiedene mögliche Antworten auf einen Request (siehe [XSPEC] Kapitel 6.11):

- PERMIT: Zulassen
- DENY: Verweigern
- INDETERMINATE: Request war ungültig und konnte nicht verarbeitet werden
- NOT_APPLICABLE: Keine Policy anwendbar

Zusätzlich kann die Antwort Obligations enthalten, welche im folgenden Kapitel beschrieben werden.

2.3.2 Obligations

Eine Obligation enthält Anweisungen an den PEP, die vor dem eventuellen Ausführen der aufgerufenen Methode befolgt werden müssen. Jede Policy kann eine oder mehrere Obligations enthalten. Daher kann in der Response neben der Entscheidung, ob der Zugriff gewährt werden darf oder nicht, auch eine oder mehrere Obligations zurückgeliefert werden. ([XSPEC] Kapitel 5.45)

Eine Obligation wird gerne dazu verwendet, Informationen über die Policy an den Aufrufer zurückzugeben. Beispielsweise kann der Grund, warum der Zugriff verweigert wurde, in lesbarer Form ausgegeben werden.

2.3.3 HERAS[AF]

HERAS[AF] [5] implementiert den XACML-Standard in Version 2.0.

Die Architektur von HERAS[AF] sieht folgendermassen aus:

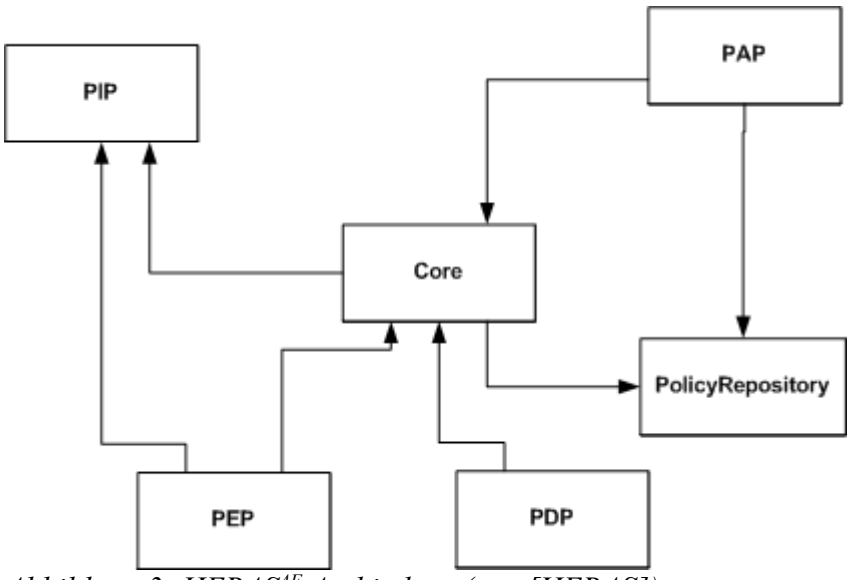


Abbildung 3: HERAS[AF] Architektur (von [HERAS])

PEP: Policy **E**nforcement **P**oint

Der PEP stellt sicher, dass die Entscheidung der passenden Policy eingehalten wird. In unserem Fall wird er vor dem eigentlichen Methodenaufruf dazwischengeschaltet und schickt einen Request an den PDP. Abhängig von der Antwort des PDP wird die Methode danach ausgeführt oder eine Fehlermeldung generiert. Die Antwort des PDP kann auch Obligations beinhalten (siehe 2.3.2).

In dieser Studienarbeit wurde ein PEP implementiert.

Core:

Der Core von HERAS[AF] entspricht von der Funktionalität her einem Policy Decision Point (PDP). Der Core nimmt Requests des PEP entgegen. Er prüft, welche Policies anwendbar sind und wertet diese aus. Er liefert dem PEP die Entscheidung zurück, ob der Zugriff gewährt werden soll. Die Policies bezieht er aus dem PolicyRepository. Falls er zur Auswertung weitere Informationen benötigt (z.B. das Alter eines Benutzers), so bezieht er diese vom PIP.

5 http://www.herasaf.org/

PIP: Policy **I**nformation **P**oint

Der PIP stellt dem Core weitere im Request nicht direkt enthaltene Informationen bereit. Werden beispielsweise detailliertere Angaben über den Benutzer benötigt, kann der PIP konsultiert werden.

PAP: Policy **A**dministration **P**oint

Mit dem PAP können die im PolicyRepository enthaltenen Policies administriert werden. Er ermöglicht das einfache Erstellen, Editieren und Löschen von Policies.

PolicyRepository

Im PolicyRepository sind sämtliche Policies gespeichert. Der Core greift auf das Repository zu, um anhand der Policies über den Zugriff zu entscheiden. Der Inhalt des Repository wird über den PAP verwaltet.

3 Architektur

Nachfolgend werden mögliche Ansatzpunkte an Spring Security diskutiert sowie die daraus resultierende Architektur beschrieben.

3.1 Einstiegspunkt

Eine der wichtigsten Entscheidungen bezüglich der Architektur ist die Auswahl des Einstiegspunktes in den Autorisierungsprozess von Spring Security. In diesem Kapitel werden die möglichen Ansatzpunkte untersucht und die gewählte Variante dokumentiert, auf der schliesslich das Design gründet.

3.1.1 Interceptor

Diese Variante verzichtet auf den Einsatz von Spring Security und es wird stattdessen mittels Spring AOP gearbeitet. Das heisst, die vorhandene Infrastruktur von Spring Security kann nicht verwendet werden. Deswegen ist diese Methode höchstens denkbar, wenn es keine Möglichkeit gibt, mit Spring Security die notwendige Funktionalität zu erreichen.

Die Implementation eines eigenen Interceptors bietet aber auch die grösstmögliche Flexibilität und ist einfach auf andere Frameworks, welche Interceptors bieten, zu portieren.

3.1.2 AccessDecisionManager

Beim *AccessDecisionManager* (ADM) handelt es sich um ein Interface von Spring Security. Der ADM wird vom SecurityInterceptor aufgerufen. Ein ADM aggregiert die Entscheidungen von *Voters*. Jeder *Voter* bestimmt, ob er den Zugriff zulassen will. Der ADM definiert, wie mit den Antworten der *Voter* umgegangen werden soll, ob zum Beispiel der Zugriff gewährt werden soll, sobald ein *Voter* zustimmt, oder nur wenn alle *Voter* zustimmen.

Durch die Implementation eines eigenen ADMs ginge des Konzept der Voter und der unterschiedlichen ADMs verloren. Es ist deshalb vorzuziehen, weiter unten anzusetzen.

Im ADM sind alle zur Entscheidung einer Autorisierung notwendigen Daten vorhanden, namentlich der Principal, Methodenaufruf, Parameter, die Klasse sowie Security-Expressions. Der ADM hat auch die Möglichkeit, Exceptions bis zum aufrufenden Code zu werfen. Dies wird zu Behandlung einiger Obligations notwendig sein.

3.1.3 Voter

Der *Voter* ist im Callstack eine Ebene unter dem *AccessDecisionManager* angesiedelt. Durch einen *Voter* bliebe dem Entwickler die Möglichkeit erhalten, über die Mechanismen von Spring Security gleichzeitig andere Autorisierungsmechanismen zu nutzen. Durch die Wahl eines ADMs könnte dann das Priorisierungs- / Überschreibungsverhalten der unterschiedlichen Voter definiert werden.

Das Problem eines *Voters* ist, dass keine Exceptions an den Aufrufer zurückgeliefert werden können. Dies liegt daran, dass das interface des *Voters* keine Exceptions erlaubt. Somit lassen (mit dem aktuellen *Voter*-Interface) durch einen *Voter* keine Obligations (Kap. 2.3.2) abarbeiten.

3.1.4 Expression Language

Wenn nur die Expression Language angepasst und die restliche Infrastruktur beibehalten wird, ergibt sich wieder das selbe Problem wie beim *Voter*. Es lassen sich keine Obligations abarbeiten. Die Implementation von eigenen Expressions alleine reicht nicht aus, allerdings ist die Verwendung von Expressions im Zusammenhang mit einem eigenen *AccessDecisionManager* vielversprechend.

3.1.5 PermissionEvaluator

Bei einem neuen *PermissionEvaluator* würde die Implementation der Methoden der Expression Language ausgetauscht. Bei der Expression *@PreAuthorize('hasPermission(...)')* würde als die Methode, welche für *hasPermission()* ausgeführt wird ausgetauscht.

Bei dieser Variante ist das Problem, dass nur die Implementation der der Funktion ausgetauscht werden kann. Die Signatur der Methode kann dabei nicht verändert werden und es können auch keine neuen Methoden hinzugefügt werden.

Dieser Ansatz reicht daher bei weitem nicht aus, um alle Informationen zu sammeln, welche für die Erstellung eines XACML-Requests notwendig sind.

3.1.6 Entscheidung

Von den untersuchten Varianten gibt es nur eine, mit welcher sich die im XACML-Standard definierte Funktionalität zufriedenstellend abbilden lässt und trotzdem ein Grossteil der von Spring Security bereitgestellten Infrastruktur weiterverwenden lässt. Es handelt sich dabei um die Variante 3.1.2 AccessDecisionManager.

3.2 Projektstruktur

Das Projekt gliedert sich grob in vier Komponenten. Der Grund dafür ist, dass es Teile des PEPs gibt, welche voneinander nicht direkt abhängig sind und auch in einem anderen Kontext verwendet werden könnten. Um beispielsweise einen PEP für ein anderes Framework als Spring Security zu implementieren, wäre es sinnvoll, die Teile, welche nicht Spring Security spezifisch sind, wieder zu verwenden. Deshalb wurden diese Teile getrennt.

3.2.1 Generische Bibliothek

Die Hauptfunktionalität des Policy Enforcement Point (PEP) ist in der generischen Bibliothek untergebracht. Die Requesterzeugung, Berechtigungsanfrage und Auswertung geschieht hier. Die generische Bibliothek hat keine Abhängigkeiten zu Spring oder der Spring-spezifischen Bibliothek.

Es werden Interfaces bereitgestellt, mit deren Implementation der PEP in verschiedene Umgebungen integriert werden kann.

3.2.2 Springspezifische Bibliothek

In dieser Bibliothek sind alle Klassen untergebracht, welche mit der Spring Integration zu tun haben. Diese Bibliothek ist der Einstiegspunkt bei der Verwendung von HERAS[AF] mit Spring Security. Auch die Klassen um aus dem Methodenaufruf die Informationen zur Requesterzeugung auszulesen befinden sich hier.

3.2.3 Transformer

Der *Transformer* wandelt mit Hilfe eines *Dictionary* die Informationen aus dem Aufrufkontext in die von einem HERAS[AF]-Request benötigte Form um. Im Dictionary werden technische Namen ihren formalen Entsprechungen gegenübergestellt. Dadurch können die technischen Bezeichnungen aus dem Aufrufkontext übersetzt werden, so dass bei der Abfrage des PDP die korrekten Policies gefunden werden.

3.2.4 Integrationtests

In diesem Projekt wird das Zusammenspiel der gesamten im PEP zur Verfügung gestellten Funktionalität getestet.

4 Design

Dieses Kapitel beschreibt die Klassen und Zuständigkeiten im PEP sowie die Packageaufteilung.

4.1 Klassendiagramm

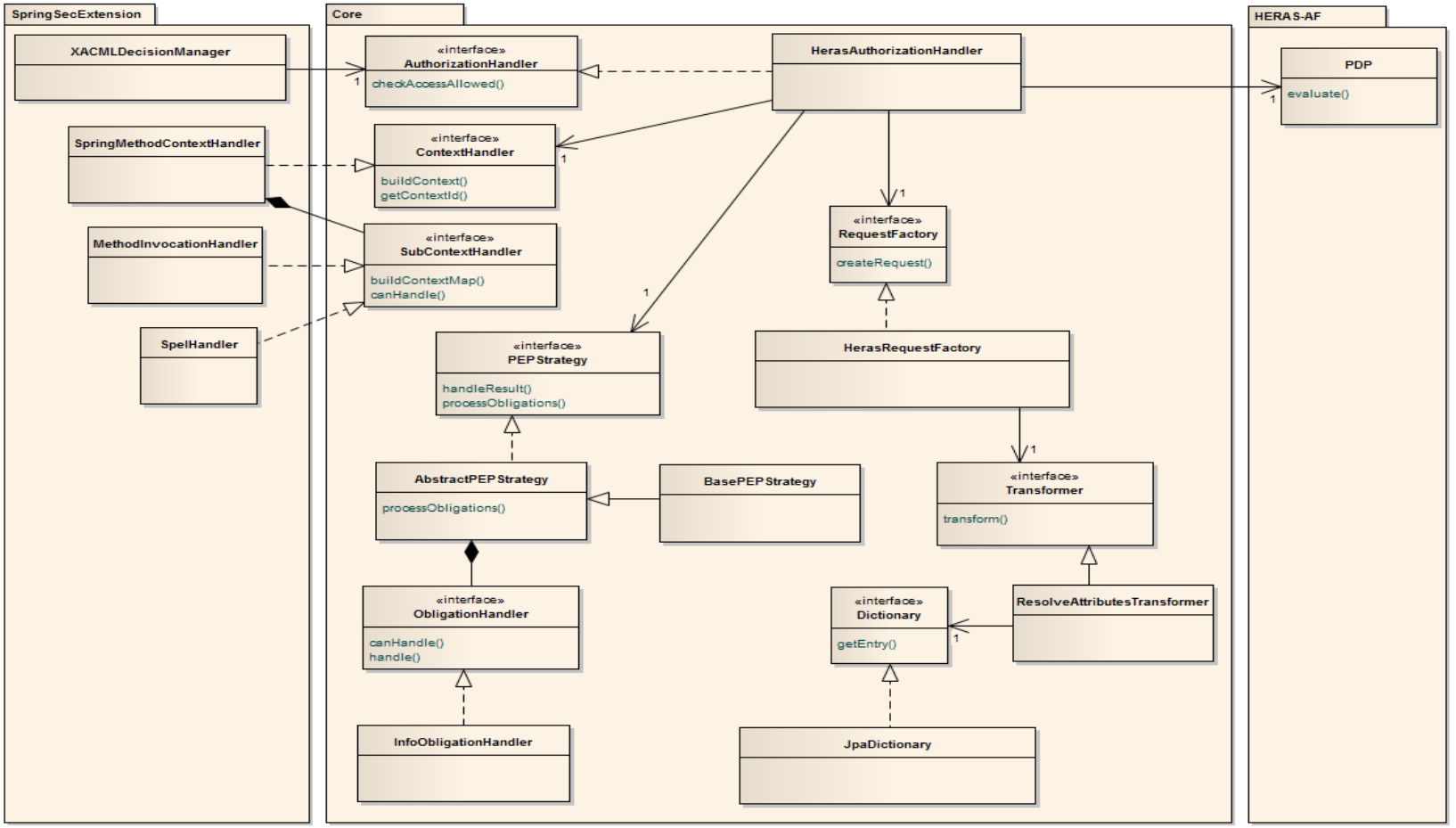


Abbildung 4: Klassendiagram

Beim Design der Architektur wurde darauf geachtet, dass der Aufwand, um einen PEP für ein anderes Framework (z.B. EJB) zu erstellen, möglichst klein gehalten wird. Deshalb gibt es die Aufteilung in eine Bibliothek für Spring und eine Bibliothek für HERAS[AF]. Die generische Bibliothek hat keinerlei Abhängigkeiten zu Spring Security.
Die Aufgabe der Bibliothek für Spring Security ist es vor allem, die über den AuthorizationHandler präsentierte Funktionalität des Kernpakets für Spring zu adaptieren. Daher enthält sie nur wenig Code. Der Hauptteil der Infrastruktur für HERAS[AF] befindet sich in der generischen Bibliothek.

4.2 Übersicht PEP Verhalten

Anhand der untenstehenden Sequenzdiagramme soll hier ein Überblick über den Ablauf einer Autorisierungsanfrage gegeben werden. Zu beachten ist, dass der Aufruf von *evaluate()* in Abbildung 6 streng genommen noch zur Requestformulierung gehört. Die Antwort auf den Aufruf hingegen zählen wir bereits zum Response-Ablauf.

4.2.1 Request an PDP

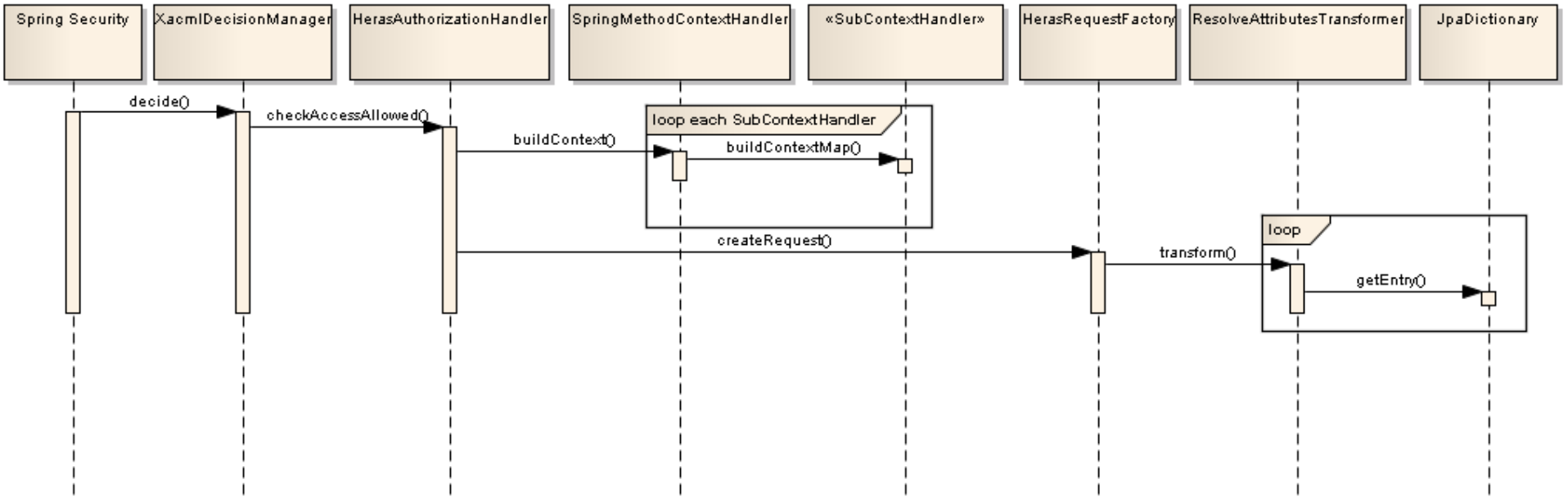


Abbildung 5: Bau des Requests an PDP

Die Requesterzeugung besteht aus folgenden Schritten:

1. Der *XacmlDecisionManager* wird von Spring Security aufgerufen.
2. Er delegiert die Entscheidung an den *HerasAuthorizationHandler*.
3. Mit Hilfe des *ContextHandlers* werden die zur Entscheidung notwendigen Informationen aus dem Aufrufkontext gewonnen.
4. Der *ContextHandler* benutzt *SubContextHandler*, welche jeweils aus Teilen des Aufrufkontexts Informationen zur Requesterzeugung gewinnen.
5. Die Informationen werden der *RequestFactory* übergeben, diese delegiert die Requesterzeugung an den *Transformer*.
6. Um technische Attributnamen in formale umzuwandeln bedient sich der *Transformer* des *Dictionary*.

4.2.2 Response von PDP

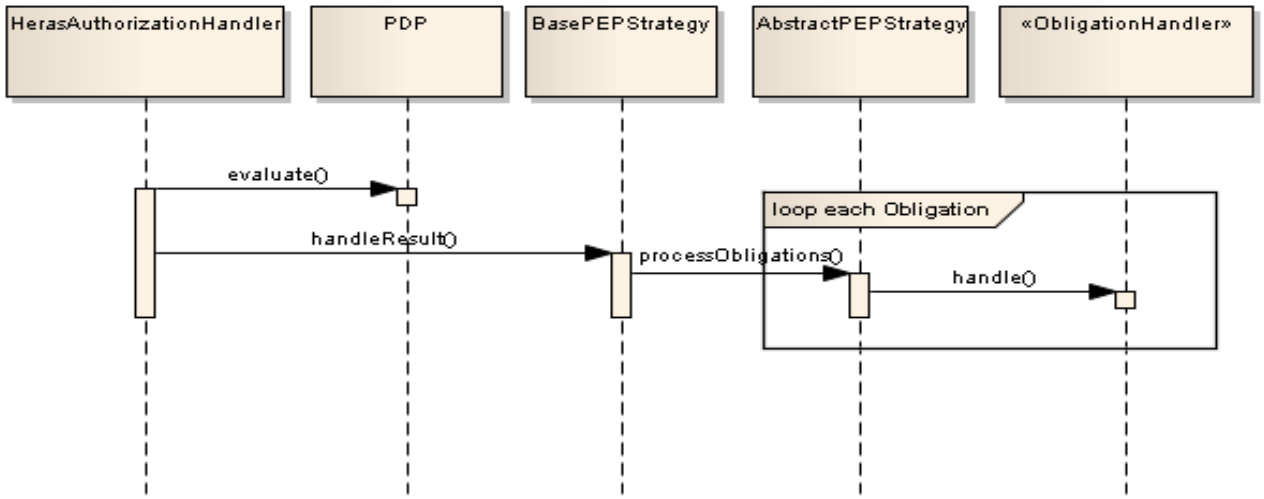


Abbildung 6: Verarbeitung der Response des PDPs

Das Diagramm zeigt wie der PEP Antworten des PDP verarbeitet. Dies geschieht in folgenden Schritten:

1. *HerasAuthorizationHandler* nimmt Antwort des PDP entgegen und übergibt sie zur Weiterverarbeitung an die *PEPStrategy* (siehe Kapitel 4.3.8).
2. Die *PEPStrategy* delegiert die Bearbeitung der Obligations an jeweils den entsprechenden *ObligationHandler* (siehe Kapitel 4.3.9).
3. Der *HerasAuthorizationHandler* liefert eine *PEPResponse* zurück, welche bestimmt, ob der Zugriff stattfinden darf. Die Entscheidung ist davon abhängig, welche *PEPStrategy* verwendet wird und ob alle Obligations abgearbeitet werden konnten.

4.3 Beschreibung Klassen

In diesem Kapitel werden die Klassen der springspezifischen sowie der generischen Bibliothek bezüglich ihrer Aufgabe und Abhängigkeiten beschrieben.

4.3.1 AuthorizationHandler

Der *AuthorizationHandler* ist die Klasse, welche den PEP steuert. Er nimmt die Daten von Spring entgegen, liest die Informationen aus und erstellt damit einen Request für HERAS[AF]. Er sendet den Request und empfängt die Antwort. Schliesslich liefert er dem *AccessDecisionManager* (ADM) die Information, ob der Zugriff gewährt oder verweigert wird. Falls HERAS[AF] neben der Entscheidung auch noch Obligations zurückgeschickt hat, werden diese ebenfalls an den ADM zurückgegeben.

Dazu verwendet er die folgenden Klassen:

- *PEPStrategy*: Definiert wie die *PEPResponse* verarbeitet werden soll.
- *ContextHandler*: Stellt die Informationen zur Requesterzeugung zusammen.
- *RequestFactory*: Erzeugt den XACML-Request.
- PDP: Wertet den Request aus.

Laut XACML-Spezifikation kann eine Antwort eines PDP in Form eines *ResponseType* mehrere Entscheidungen enthalten. (siehe [XSPEC] Kap. 6.9) Jede Entscheidung ist wiederum in einem *ResultType* enthalten.

In der vorliegenden Implementation des PDP durch HERAS[AF] wird nur ein *ResultType* in der Response zurückgegeben. Daher wird im realisierten *AuthorizationHandler* namens *HerasAuthorizationHandler* auch direkt das erste Element der Liste von ResultTypes, die im *ResponseType* enthalten ist, entnommen.

4.3.2 XACMLDecisionManager

Der *XACMLDecisionManager* fungiert als Adapter für den *AuthorizationHandler*. Im implementierten Fall adaptiert er den *AuthorizationHandler* zur Verwendung mit Spring Security. Er hat keine weitere Funktionalität, sondern delegiert übergebene Informationen von Spring Security an den *AuthorizationHandler* und umgekehrt.

4.3.3 ContextHandler

Der *ContextHandler* definiert den Kontext, in welchem der PEP eingesetzt wird. Für jede Art von Kontext muss ein entsprechender *ContextHandler* implementiert werden. Bereits während der Studienarbeit implementiert wurde ein *ContextHandler* für mittels Spring Security abgesicherte Methodenaufrufe (*SpringMethodContextHandler*).

Um Zugriffe auf URLs abzusichern müsste ein neuer *ContextHandler* erstellt werden. Eine Instanz des PEPs ist jeweils für einen Kontexttyp zuständig. Das heisst, es ist immer nur ein *ContextHandler* registriert.

Das Sammeln von für die Autorisierung relevanten Information delegiert der *ContextHandler* an *SubContextHandler*.

4.3.4 SubContextHandler

Ein *SubContextHandler* ist dafür zuständig jeweils aus einem kleinen Teil des Aufrufkontexts Daten zur Requesterzeugung zu extrahieren. Zum Beispiel extrahiert ein *SubContextHandler* den angemeldeten Benutzer aus der Authentication.

Über den *ContextHandler* erhält ein *SubContextHandler* den Kontext des Ressourcenzugriffs. Der *ContextHandler* wandelt die im Kontext enthaltenen Daten in eine einheitliche Form zur Weiterverarbeitung um.

Dazu gibt es folgende Methode:

AuthorizationContext buildContext(Object... contextObjects)

Ihr wird beispielsweise die *Authentication* von Spring Security übergeben. Daraus können dann der eingeloggte Benutzer und seine Rollen ausgelesen werden. Diese Daten werden dann in einem *AuthorizationContext* zusammengefasst, welcher von der *RequestFactory* weiterverarbeitet wird.

4.3.5 RequestFactory

Die *RequestFactory* delegiert das Erstellen eines XACML-konformen Requests. Dazu werden ihr sämtliche Informationen über den Kontext der Anfrage wie Subjekt, Ressourcen, Aktionen und weitere Informationen zur Umgebung übergeben.

Der *RequestFactory* wird ein *Transformer* injiziert. An diesen gibt die Factory die erhaltenen Informationen weiter und erhält einen für den Policy Decision Point verständlichen *RequestType* zurück.

Der erstellte *RequestType* wird an den *AuthorizationHandler* zurückgegeben, welcher ihn an den PDP schickt.

4.3.6 Transformer

Der *Transformer* ist dazu da, verschiedene Modelle von Attributen zu ermöglichen. Er kann technische Attributnamen aus dem Code in formale für den PDP umwandeln. Es ist auch möglich, dass der *Transformer* gewisse Attribute zusammenfasst oder aufteilt.

Er iteriert durch die Attribute der übergebenen Maps. Mithilfe des injizierten *Dictionaries* werden technische Attributnamen und -werte durch formale ersetzt. Anschliessend erstellt der *Transformer* aus den Attributen einen XACML-*RequestType* und gibt diesen zurück.

4.3.7 Dictionary

Beim *Dictionary* handelt es sich grundsätzlich um einen Lookup-Service, der technische Attributnamen und -werte entgegennimmt und ihre formale Darstellung, wie sie in den Policies des PDP verwendet werden, zurückliefert. Die Datenquelle, auf die das *Dictionary* zugreift und die das Mapping beinhaltet, kann durch den Kunden beliebig umgesetzt werden.

4.3.7.1 JpaDictionary

Diese Implementation der *Dictionary*-Schnittstelle zeigt eine mögliche Realisierung der Datenhaltung mittels Java Persistence API (JPA).

4.3.8 PEPStrategy

Aufgrund des Requests sucht der Policy Decision Point die anzuwendende Policy und gibt deren Inhalt in Form eines *ResponseType* zurück. Darin enthalten sind ein oder mehrere *ResultTypes*.

In dieser Implementation des PEP wird jeweils nur das erste zurückgegebene *ResultType* weiterverarbeitet, da in der gegenwärtigen Implementation von HERAS^{AF} nur ein *ResultType* erstellt wird. Theoretisch erlaubt XACML jedoch auch eine Response mit mehreren *ResultTypes* (siehe [XSPEC] 6.9).
Ein *ResultType* enthält die Entscheidung des PDP, die Id der angeforderten Ressource sowie optional eine oder mehrere Obligations als auch ein Statuscode bezüglich der Verarbeitung des Requests.
Falls Obligations zurückgeliefert wurden, kann es sein, dass eine positive Entscheidung des PDP (das heisst der Methodenzugriff wird erlaubt) noch geändert und der Zugriff verweigert wird. Dies ist abhängig davon, ob alle Obligations verarbeitet werden können und was für eine Strategie eingesetzt wird.

Der XACML-Standard definiert drei Strategien: Permit-biased, Deny-Biased und Base. (siehe [XSPEC] S.82)

- Permit-biased: Der PEP soll, falls die Entscheidung "Deny" lautet, den Zugriff verweigern. Falls Obligations mitgeliefert wurden, wird der Zugriff nur verweigert, wenn alle verstanden wurden und verarbeitet werden konnten.
  In allen anderen Fällen wird der Zugriff erlaubt.
- Deny-biased: Der PEP soll, falls die Entscheidung "Permit" lautet, den Zugriff erlauben. Falls Obligations mitgeliefert wurden, wird der Zugriff nur erlaubt, wenn alle verstanden wurden und verarbeitet werden konnten.
  In allen anderen Fällen wird der Zugriff verweigert.
- Base Strategy: siehe Kap. 4.3.8.1

Es ist dem Kunden überlassen, welche Strategie er implementieren möchte. In der Kernbibliothek wurde beispielhaft die Base-Strategie implementiert

4.3.8.1 BasePEPStrategy

Bei dieser Strategie wird grundsätzlich die Entscheidung des Policy Decision Point übernommen. Allerdings wird zusätzlich darauf beharrt, dass allfällig zurückgelieferte Obligationen durch die konfigurierten *ObligationHandler* verstanden werden konnten.

Laut XACML-Standard ist, falls der PDP als Entscheidung `NOT_APPLICABLE` oder `INDETERMINATE` angibt, das Verhalten des PEP bei dieser Strategie nicht definiert.

In Rücksprache mit den Betreuern wurde entschieden, in diesen Fällen den Zugriff zu verweigern und als Entscheidung "Deny" an Spring Security zurückzugeben.

4.3.9 ObligationHandler

Allfällige durch den Policy Decision Point mit der Zugriffsentscheidung zurückgelieferte Obligations müssen durch einen *ObligationHandler* behandelt werden. Bei einem Policy Enforcement Point können ein oder mehrere *ObligationHandler* eingesetzt werden.

Ein *ObligationHandler* kann eine Art von Obligations behandeln. Je nach der Art können sich die Strukturen von Obligationen stark voneinander unterscheiden. Darauf muss der *ObligationHandler*

zugeschnitten sein.

Um was für eine Obligation es sich handelt, wird über die obligationId ermittelt, die jede Obligation besitzt. Beispielsweise bezeichnet die obligationId "information" eine Obligation, die dafür verwendet wird, die Policy beschreibende Nachrichten zu halten.

Ein *ObligationHandler* implementiert unter anderem die Funktion

ObligationHandlingResult handle(ObligationType obligation)

Der Parameter vom Typ *ObligationType* wurde vom Policy Decision Point zusammen mit der Entscheidung in einem sogenannten *ResultType* zurückgegeben und enthält alle Informationen über eine Obligation.

Der Rückgabewert *ObligationHandlingResult* beinhaltet Informationen darüber, ob der *ObligationHandler* die Obligation korrekt verarbeiten konnte und Nachrichten und optional eine Fehlermeldung falls nicht. Das Verarbeiten dieser Rückgabe liegt in der Verantwortung der implementierten *PEPStrategy*. Auf keinen Fall darf die Fehlermeldung zur Laufzeit geworfen werden, da dies den Ablauf des Autorisierungsprozesses und somit des Programms unterbrechen würde.

5 Ausblick

Der im Rahmen dieser Arbeit implementierte PEP bietet eine gute Ausgangslage für weitere Funktionalität. Einige denkbare Ansätze wären:

- *SubContextHandler* für Annotations:

  Einen *SubContextHandler* erstellen, welcher anstelle der EL-Expressions mit Annotations arbeitet.

- Integration für EJB statt Spring

  Die Implementation ist in eine Spring-spezifische Bibliothek und eine Kernbibliothek, die HERAS^{AF} anbindet, aufgeteilt. Da die Kernbibliothek keinerlei Abhängigkeiten von Spring Security hat und lediglich Interfaces mit generischen Funktionsparametern für diese bereitstellt, liesse sich die Spring Security Anbindung leicht durch eine andere ersetzen, welche die Kernbibliothek nutzt. Denkbar wäre beispielsweise die Benutzung eines EJB-Services.

- Web Sicherheit

  In der vorliegenden Implementation wird die Absicherung von Methodenaufrufen unterstützt. Spring Security bietet jedoch auch Websicherheit an. Durch entsprechende *ContextHandler* und *SubContextHandler* liessen sich solche Aufrufe ebenfalls mittels HERAS^{AF} absichern.

- Attribute auf Klassen setzen

  Die Möglichkeit anbieten, Attribute, welche für jede Methode einer Klasse gleich sind, direkt auf der Klasse zu definieren.

- .NET-PEP

  Eine Implementation des PEPs für .NET.

6 Anhang

6.1 Projekt Management

6.1.1 Methodik

Dieses Projekt wird anhand der Prinzipien der agilen Softwareentwicklung (Agile Software Development ASD) verwirklicht.

Einige wesentliche Werte sind:

- leichtgewichtig: Im Gegensatz zu klassischen Vorgehensmodellen wie dem Rational Unified Process oder dem V-Modell wird durch den Fokus auf Ergebnisse der Prozess schlank gehalten.
- Offenheit für Änderungen: Oft ändern sich Anforderungen an das Produkt und das Verständnis des Problemfeldes während des Entwicklungsprozesses. Darauf kann dank ASD schnell reagiert werden.
- Stetige Abstimmung mit dem Kunden: Durch laufende konstruktive Kommunikation mit dem Kunden können Erwartungen und Ergebnisse schnell aufeinander abgestimmt werden.

6.1.2 Projektphasen

Grob wird der Projektablauf in vier Phasen eingeteilt: Analyse, Design & Prototyping, Ausarbeitung und Abschluss. Am Ende jeder Phase wird ein Meilenstein gesetzt.

6.1.2.1 Analyse

In der ersten Phase wird die Ausgangslage analysiert. Bestehende Produkte und Vorgaben werden betrachtet und in den Kontext der eigenen Arbeit gestellt. Zum Verständnis der Problemstellung werden Beispielapplikationen untersucht oder selbst programmiert.

6.1.2.2 Design & Prototyping

Nachdem die Ausgangslage bekannt ist, wird darauf aufbauend die Architektur des eigenen Produkts entworfen. Anhand dieses Designs wird ein erster lauffähiger Prototyp implementiert. Der Prototyp soll alle Aspekte des Designs wenigstens in grundlegenden Zügen implementieren und so einen Ansatz für die Ausarbeitung bilden.

6.1.2.3 Ausarbeitung

Steht ein Prototyp, der alle wesentlichen Funktionen implementiert, werden verschiedene Aspekte der Lösung erweitert. In welche Richtung diese Ausweitung geschieht, wird zusammen mit den betreuenden Personen entschieden. Am Ende der Phase soll das fertige Softwareprodukt stehen.

6.1.2.4 Abschluss

In der letzten Phase wird das Feintuning von Software und Dokumentation betont. Es sollen keine neuen Features mehr dazukommen. Zusätzliche abzugebende Artefakte wie Abstract, Poster und Erfahrungsberichte werden in dieser Phase erstellt.

6.1.2.5 Meilensteine

Anhand der Phasen wurden die folgenden Meilensteine definieren:

MS-Nr.	Inhalt	Arbeitsresultat	Lieferdatum Soll
MS 1	Abschluss der Voranalyse	Dokumentation der Analyse	08.10.10
MS 2	Lauffähiger Architekturprototyp	Vertikal vollständiger Softwareprototyp	22.10.10
MS 3	Lauffähige Version für Methodenaufrufe	Software & Dokumentation	26.11.10
MS 4	Endabgabe	Weitere Artefakte gemäss Vorgabe HSR (Abstract, Poster)	23.12.10

Tabelle 1: Meilensteine

6.1.3 Projektplanung

6.1.3.1 Meetings

Jeden Freitag um 15:00 wird ein Meeting mit den Betreuern durchgeführt. Dabei werden der aktuelle Stand der Arbeit besprochen und die Ziele für die Folgewoche definiert. Die Ergebnisse der Meetings sind in den Sitzungsprotokollen (siehe Kapitel 6.2) festgehalten.

6.1.3.2 Zeitplan

	Task Nr.	Taskname															
Woche			01		02		03		04		05		06		07		
Wochenstart			20.09.		27.09.		04.10.		11.10.		18.10.		25.10.		01.11.		
Phase			Analyse						Design & Prototyping				Ausarbeitung				
Teammitglied			db	sb	db	sb	db	sb	db	sb	db	sb	db	sb	db	sb	
		Dokumentation															
	101	Dokumentvorlagen			2	2				2							
	102	Dokumentation											6.5	10	12.5	2.5	
		Projektmanagement															
	201	Meetings	5	5	1.5	1.5	2.5	2.5	3.5	1.5	4	2	2.5	4	2	2.5	
	202	Infrastruktur	2.5	3.5	1	2		1	3	10.5	1						
		Voranalyse															
	301	Businessanalyse		3.5			6.5	6									
	302	Spring Analyse	4		10.5	7						2.5					
		Entwicklung															
	401	Prototyping							3.5	6.5	10.5	12	6			8	
	402	Design							1.5	3							
	403	Refactoring															
	404	Programmierung														8	
Total			11.5	12	15	12.5	9	9.5	11.5	23.5	15.5	16.5	15	14	14.5	21	
Total Team			23.5		27.5		18.5		35.0		32.0		29.0		35.5		
Total Phase			69.5						67				120				

Milestone				MS1		MS2		
Wochenziele		Infrastruktur	Analyse Spring Security	Analyse HERASAF, Voter Prototyp	Design PEP, Design ContextHandler	Vertikal vollständiger Prototyp	Dokumentation, Refactoring Prototyp	Dokumentation, Ausbau Software

	Task Nr.	Taskname														
Woche			08		09		10		11		12		13		14	
Wochenstart			08.11.		15.11.		22.11.		29.11.		06.12.		13.12.		20.12.	
Phase			Ausarbeitung						Abschluss							
Teammitglied			db	sb	db	sb	db	sb	db	sb	db	sb	db	sb	db	sb
		Dokumentation														
	101	Dokumentvorlagen														
	102	Dokumentation	4.5				2.5	9	3.5	16	19.5	18	20	20.5	21	19
		Projektmanagement														
	201	Meetings	2	2	4	2	3	2	2	1	1	1	2	2	1	1
	202	Infrastruktur														
		Voranalyse														
	301	Businessanalyse														
	302	Spring Analyse														
		Entwicklung														
	401	Prototyping														
	402	Design														
	403	Refactoring	2.5	2.5			6.5		2							
	404	Programmierung	5	12	15	10			3.5							
Total			14	16.5	19	12	12	11	11	17	20.5	19	22	22.5	22	20
Total Team			30.5		31.0		23.0		28.0		39.5		44.5		42.0	
Total Phase			120						154							

Milestone				MS3				MS4
Wochenziele		Dokumentation, Refactoring, Unit Testing	Ausbau Software, Unit Testing	Dokumentation, Refactoring	Dokumentation, Refactoring	Dokumentation	Dokumentation, Abstract	Dokumentation, Poster

6.1.3.3 Arbeitspakete

AP-Nr	Name	Inhalt/Arbeitsresultat	Ist (Std.)
100	Dokumentation		
101	Dokumentvorlagen	Erstellen und Verfeinern von Dokumentvorlagen	6
102	Ergebnisse dokumentieren	Dokumentieren der Resultate einzelner Iterationen	185
200	Projektmanagement		
201	Meetings	Vorbereitung, Durchführung und Nachbearbeitung (Protokollierung)	66
202	Infrastruktur	Einrichten der Entwicklungsumgebung	24.5
300	Voranalyse		
301	Business Analyse	Analyse der Funktionalität und Funktionsweise von HERAS[AF]	16
302	Spring Analyse	Analyse der Funktionalität und Funktionsweise von Spring	24
400	Entwicklung		
401	Prototyping	Erstellen von Prototypen verschiedener Varianten und Komponenten	46.5
402	Design	PEP Design	4.5
403	Refactoring	Code Refactoring	13.5
404	Programmierung	Erweiterung des Prototypen zu fertigem Produkt	53.5

Tabelle 2: Arbeitspakete

6.1.4 Risikomanagement

Risiko-Nr	Risiko	Auswirkung	Massnahme	Aufwand Massnahme	Maximaler Schaden	Eintritts-wahrscheinlichkeit	Gewichteter Schaden	Priorität
R01	Sourcecontrol steigt aus	Zeitverlust	Bei Eintreten: Umzug auf anderen Server	8h	8h	5%	0.4h	Klein
R02	Laptop steigt aus	Datenverlust	Regelmässiger Checkin	2h	5h	5%	0.25h	Klein
R03	Technologie zu komplex	Zeitaufwand für Umsetzung zu hoch	Ausführliche Analyse inklusive Prototypen	5h	30h	20%	6h	Mittel
R04	Anforderungen werden falsch interpretiert	Es werden Features anders als erwartet implementiert	Regelmässige und klare Kommunikation mit den Auftraggebern	10h	50h	30%	15h	Hoch
R05	Teammitglied fällt aus	Zeitplan kann nicht eingehalten werden	Funktionalität reduzieren	-	240h	10%	24h	Hoch
R06	Komplexität von XACML unterschätzt	Zeitaufwand für Implementation steigt	Ausführlich analysieren	5h	30h	20%	6h	Mittel

Tabelle 3: Risiken

6.1.5 Qualitätssicherung

6.1.5.1 Buildserver

Das Projekt verwendet den Buildserver von HERAS[AF]. Nach jedem Checkin wird automatisch ein Build gestartet. Im Buildprozess enthalten ist das Ausführen der Testfälle. Im Fehlerfall werden die Entwickler per E-Mail benachrichtigt.

6.1.5.2 Code Reviews

Die Entwickler gehen regelmässig neuen Code gemeinsam durch. Wöchentlich wird neuer Code mit den Auftraggebern besprochen.

Erkenntnisse aus den Gesprächen mit den Auftraggebern werden in Protokollen (siehe Kap. 6.2) festgehalten.

Formale Codereviews gibt es keine.

6.1.5.3 Coverage

Es wurde eine Coverage von 80% über alle Projekte angestrebt und erreicht. Wichtige Klassen weisen eine höhere Abdeckung auf. Die Codeabdeckung wird lokal auf den Entwicklungsrechnern mithilfe des Eclipse Plugins EclEmma[6] überprüft.

6 http://www.eclemma.org/

6.2 Sitzungsprotokolle

Die Sitzungprotokolle folgen anschliessend an dieses Dokument.

6.3 Erfahrungsberichte

6.3.1 Daniel Bobst

Für mich war das Thema der Studienarbeit besonders interessant, da mir keine der zentral eingesetzten Technologien im Voraus bekannt war. Weder mit Spring Security, HERAS[AF] noch mit Maven hatte ich zuvor schon zu tun. Das war einerseits besonders zu Beginn eine Herausforderung, andererseits lernte ich so mit jedem Schritt etwas völlig Neues.

Da unser Produkt schlussendlich ein Modul von HERAS[AF] ist, waren die wöchentlichen Meetings mit den Auftraggebern für uns enorm wichtig. Wir lernten das Fachwissen sowie die unkomplizierte Art der Betreuer sehr zu schätzen.

Zugegebenermassen verlief die Arbeit zu Beginn etwas zögerlich. Der Grund lag darin, dass uns beiden die eingesetzten Frameworks neu waren und es sich zusätzlich um komplexe Anwendungen handelt. Dies machte die Analyse ein wenig schwerfällig. Ausserdem musste die Architektur sorgfältig und unter Rücksichtnahme der umliegenden Systeme entworfen werden.

Nicht zuletzt dank der Unterstützung des HERAS[AF] Teams konnte schliesslich doch noch rechtzeitig der Prototyp fertiggestellt werden. Nun konnte aufbauend auf dieser soliden Basis zügiger fortgeschritten werden.

Was mich auch motivierte, war, dass im Gegensatz zu vorherigen kleineren Projekten im Rahmen von Modulen wie User Interfaces oder Software Engineering das Endresultat nicht für sich allein steht und in Vergessenheit gerät. Es ist ein gutes Gefühl zu wissen, dass das entwickelte Produkt in der einen oder anderen Form vom Team HERAS[AF] weiterbenutzt oder weiterentwickelt wird.

6.3.2 Sandro Brändli

Das Thema der Studienarbeit hat mich vor allem aufgrund der involvierten technologien interessiert. Von HERAS[AF] hatte ich zuvor noch nichts gehört. Spring kannte ich nur vom Hörensagen und war mir der Mächtigkeit dieses Framweworks nicht bewusst. Es hat mich gereizt, mit modernen Enterprise-Technologien zu arbeiten.

Am Anfang des Projekts war ich etwas erschlagen von den vielen Frameworks und Technologien (Maven, Subversion, Spring, Spring Security, HERAS[AF]), mit denen ich noch wenig bis keine Erfahrung hatte. Glücklicherweise gab es gute Dokumentation.

In der vierten Woche bekamen wir eine Beispielapplikation, welche mittels HERAS[AF] abgesichert war. Dies hat unsere Arbeit stark erleichtert, da wir trotz der vorhandenen Dokumentation nicht in der Lage waren, einen PDP aufzusetzen. Wie sich herausstellte, ist dies nicht schwierig, aber es fehlte entsprechende Dokumentation.

Sehr geschätzt habe ich die wöchentlichen Statusmeetings, an welchen wir den aktuellen Stand besprachen und die Ziele für die nächste Woche definierten. Die Statusmeetings haben uns geholfen, iterativ zu arbeiten und Kommunikationsprobleme schnell zu erkennen. Die Anforderungen haben sich im Rahmen der Meetings weiterentwickelt.

Abschliessend lässt sich sagen, dass ich im Rahmen des Projekts viel gelernt habe und die Arbeit spannend war. Ich denke wir haben eine gute Basis geschaffen, die sich leicht erweitern lässt.

6.4 Referenzierte Dokumente

Kürzel	Ort des Dokuments	Bemerkungen
[AUFG]	doc/Auftrag/Aufgabenstellung.pdf	
[ANLEI]	doc/DokuAnleitungBA_DA_SA_100304.pdf	
[BEUR]	doc/Beurteilung_SA_empty.doc	
[XSPEC]	http://www.oasis-open.org/committees/download.php/26986/access_control-xacml-2.0-core-spec-os-errata.doc	letzter Zugriff: 06.12.2010
[SPREF]	http://static.springsource.org/spring-security/site/docs/3.0.x/reference/springsecurity.pdf	letzter Zugriff: 06.12.2010
[RFC3198]	http://www.ietf.org/rfc/rfc3198	letzter Zugriff: 06.12.2010
[HERAS]	http://www.herasaf.org/	letzter Zugriff: 08.12.2010

6.5 Erklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.

Ort, Datum: Unterschrift:

_____ _____

Sitzungsprotokoll

Projekt: **Heras-Spring Integration**
Woche: **1**
Datum: **Rapperswil, den 24.09.2010**
Dauer: **15:30-16:00**

Teilnehmer / Kürzel:
Daniel Bobst / db
Sandro Brändli / sb
Wolfgang Giersche / wg
René Eggenschwiler / re

Traktanden:

1. Benötigte Infrastruktur (SVN, JIRA, Wiki, Zeiterfassung, Projektmanagement)
2. Welche Termine gibt es
3. Welche Dokumente sind erforderlich
4. Was sind die Bewertungskriterien
5. Zielarchitektur festlegen
6. Beispielapplikationen
7. Mögliche Lösungen besprechen

Diskussion / Beschlüsse:

1. Infrastruktur
   SVN, JIRA, Wiki und ev. Bamboo werden uns zur Verfügung gestellt.

   SVN: http://svn.herasaf.org/education/herasaf-xacml-pep/
   JIRA/Wiki: http://wiki.herasaf.org/display/MSRPSPRINGSECURITYPEP

2. Termine
   Optional: Wöchentlich 13:00 mit René Eggenschwiler für technische Fragen

   Wöchentlich 15:00 mit Wolfgang Giersche: Planung, Zielerreichung, Administratorisches

   Weitere Termine: https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html

3. Dokumente
   Die notwendigen Dokumente sind im Dokument DokuAnleitungBA_DA_SA_100304.pdf beschrieben.

   Aus Sicht der Betreuer können die meisten Dokumente im technischen Bericht zusammengefasst werden.

Separat sein müssen: Abstract, Poster, Getting Started (am besten im Wiki)
Vorlagen gibt es hier: https://www.hsr.ch/Allgemeine-Infos-Diplom- Bach.4418.0.html
Zusätzlich werden uns Vorlagen von Heras zur Verfügung gestellt.

4. Bewertungskriterien
   Der Bewertungsbogen (Beurteilung_SA_empty.doc) dient als Referenz.

5. Zielarchitektur
   Die Zielarchitektur wird im Rahmen des Projekts erarbeitet. Teile davon sind in der Aufgabenstellung vorgegeben.

6. Beispielapplikationen
   http://svn.herasaf.org/playground/springsecurity-test/ ist eine Beispielapplikation für SpringSecurity.

7. Lösungsvarianten
   Die Lösungsvarianten sollen im technischen Bericht detailliert beschrieben werden.

Offene Punkte Verantwortung (erledigt vor nächster Sitzung):

Was	**Wer**
Infrastruktur zur Verfügung stellen	re
Zusammenspiel von SpringSecurity-Klassen zeigen (Doku, Debugger)	sb, db

Kommende Abwesenheiten

Wer	Von	Bis	Bemerkung

Nächster Termin:

Datum: Rapperswil, den 01.10.2010
Zeit: 15:00
Dauer:

Sitzungsprotokoll

Projekt: **Heras-Spring Integration**
Woche: **1**
Datum: **Rapperswil, den 24.09.2010**
Dauer: **13:00-15:30**

Teilnehmer / Kürzel:
Daniel Bobst / db
Sandro Brändli / sb
René Eggenschwiler / re

Traktanden:

Diskussion / Beschlüsse:

1. SpringSecurity: Zwei Kontexte für unsere Arbeit:
   MethodInvocation -> damit beginnen
   WebApplication Security
2. SVN: /education/herasaf-xacml-pep
   JIRA/Wiki: MSRPSPRINGSECURITYPEP
3. XACML Policy Bsp.
   s.a. /playground/syntax-exception-test/

   matchid=rfc-822nameMatch
   AttValue=hsr.ch
   SubjectAttributeDesignator
   AttributeId="urn:email:adress"

   Zusätzlich in „target":
   - resources
   - Actions
   - Environment

   Zusätzlich Policy
   - Rule

   - Target == true, falls matcht
   - If Rule == true -> evaluate to effect: permit OR deny OR indeterminate
   - Sind z.B. Subject und Action in Target definiert -> beide müssen matchen, damit Policy zutrifft

2. Request Bsp.
   Request
   - Subject
   - - Attribute id = „urn:email:adress"
   - - - AttributeValue = xyz@hsr.ch
   - - Attribute2

- - Attribute3
- - etc.
- Resource
- - Attribute id=urn:java:method:...
- - - value=getRecord

3. Problem: ActionValue „write“ zu Java Methode „savePerson“ matchen
   1. Eine Möglichkeit:
      @PreAuthorize(hasPermission(#p, „write“)

      MyPermissionEvaluator implements PermissionEvaluator
      hasPermission(...)
      subject: principal.getName() o.ä.
      action: urn:hsr:actiontype=write

      Problem: woher kennen wir urn:hsr:actiontype ? Hier noch nicht bekannt
      Mögl. Lösung: in generischer Library -> Dictionary (muss zentral gepflegt werden)
      Mapping AppLanguage – XACML language
      „write“ - urn:hsr:actiontype (AttId) – write (AttValue)

      Problem: subject & action rausholen machbar, aber resource? Methode dafür fehlt (Bei Interceptor machbar, Evaluator nicht)

   2. Weitere Möglichkeit:
      AccessDecisionManager
      Voters
      Evaluators

   3. Weitere Möglichkeit:
      Eigene ExpressionLanguage: z.B.
      @PreAuthorize(herasafsecured('resource=#p', 'actiontype=save',...))
      statt
      @PreAuthorize(hasRole=“SuperUser“)

4. Technischer Bericht Struktur
   1. Kontext:
      1. Warum diese SA?
   2. Business Analyse
      1. Was braucht XACML?
   3. Spring Analyse
      1. Was kann Spring?
      2. Was wäre zusätzlich schön?
   4. Architektur
      1. Mögl. Varianten -> inkl. Entscheidungen und Begründung
      2. Design
   5. Realisierung
      1. Code/Test
   6. Ausblick
      WICHTIG: Ausprobieren, aber auch jeweils dokumentieren

5. HERAS Request Struktur
   RequestType
   - List<SubjectType>
   - - List<AttributeType>
   - - - List<AttributeValueType>
   -> bildet RequestCtx
   => PDP.evaluate(RequestCtx)
   SimplePDPFactory.getSimplePDP(...)

   RequestCtx: mühsame Struktur
   könnte sich lohnen, für CtxHandler eine Art Convenience API zu entwickeln -> Request.addSubject(...), Request.addAttrib(...)

6. Abzugebende Doku:
   1. Technischer Bericht
      1. Erklärung über eigenständige Arbeit
      2. Danksagung
      3. Aufgabenstellung
      4. Inhalt Doku
         1. Inkl. Lit.verzeichnis/Glossar
      5. Erfahrungsbericht
   2. Poster
   3. Abstract
   4. Wiki: Getting Started Guide erstellen (Installation/Setup)
7. TODO:
   Spring Analyse: Diagramm & Sample debuggen
   -> AccessDecisionManager
   -> MethodInvocation
   Varianten analysieren und eine auswählen

Offene Punkte Verantwortung (erledigt vor nächster Sitzung):

Was	**Wer**
Infrastruktur zur Verfügung stellen	re
Zusammenspiel von SpringSecurity-Klassen zeigen (Doku, Debugger)	sb, db

Kommende Abwesenheiten

Wer	Von	Bis	Bemerkung

Nächster Termin:

Datum: Rapperswil, den 01.10.2010
Zeit: 15:00
Dauer:

Sitzungsprotokoll

Projekt: **Heras-Spring Integration**
Woche: **2**
Datum: **Rapperswil, den 01.10.2010**
Dauer: **14:50-16:30**

Teilnehmer / Kürzel:
Daniel Bobst / db
Sandro Brändli / sb
Wolfgang Giersche / wg

Traktanden:

1. Mögliche Lösungen für Heras-Integration besprechen

Diskussion / Beschlüsse:

1. Lösungsvarianten
   Es gibt folgende Lösungsvarianten:
   1. Eigenen Interceptor implementieren
   2. Eigenen DecisionManager implementieren
   3. Eigenen Voter implementieren
   4. Neue EL-Expression einführen
   5. PermissionEvaluator implementieren

   Folgende Daten sollen Heras geliefert werden:
   - Subject: Der Benutzer
   - Resource: Die Resource, auf welche zugegriffen werden soll
   - Action: Die Aktion, die auf der Resource ausgeführt werden soll
   - Environment: Kann alles mögliche enthalten

   Wir haben uns vorläufig für Variante 3 (Eigener Voter) entschieden, da ein Voter noch über alle zu liefernden Daten verfügt. Ausserdem kann im Gegensatz zu den Varianten 1 und 2 ein grösserer Teil der Infrastruktur von Spring Security verwendet werden.
   Das Konzept der Voter soll beibehalten werden. Der Benutzer muss aber darauf hingewiesen werden, dass es nicht sinnvoll ist eigene Voter zu verwenden, wenn mit Heras gearbeitet wird.

   Es dürfen auf keinen Fall technische Namen im PDP verwendet werden. Dazu muss ein ContextHandler mit möglichst abstraktem Interface erstellt werden. Dieser übersetzt die technischen Namen in formale Ausdrücke, die im PDP verarbeitet werden können.

   Als nächster Schritt soll Heras besser verstanden werden. Dazu soll die RequestFactory analysiert werden.
   Ausserdem soll ein Prototyp eines Voters erstellt und getestet werden.

Offene Punkte Verantwortung (erledigt vor nächster Sitzung):

Was	**Wer**
RequestContextFactory analysieren	sb, db
XACMLVoter Prototyp erstellen und testen	sb, db

Kommende Abwesenheiten

Wer	Von	Bis	Bemerkung

Nächster Termin:

Datum: Rapperswil, den 08.10.2010
Zeit: 15:00
Dauer:

Sitzungsprotokoll

Projekt:	**Heras-Spring Integration**
Woche:	**3**
Datum:	**Rapperswil, den 08.10.2010**
Dauer:	**15:00-16:45**

Teilnehmer / Kürzel:
Daniel Bobst / db
Sandro Brändli / sb
Wolfgang Giersche / wg

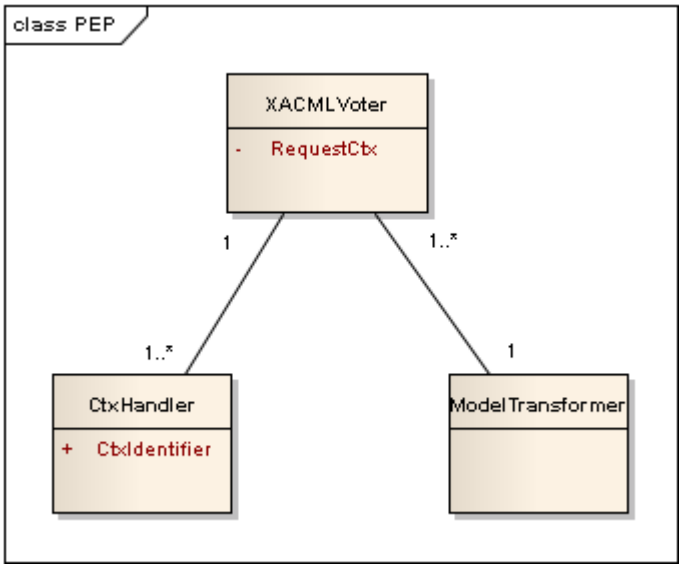
Traktanden:

1. XACMLVoter besprechen
2. RequestContextFactory besprechen
3. Welche Datentypen unterstützt Heras (z.B. als Environment-Parameter), wie funktioniert die Konvertierung?
4. Wo gibt es Code zur aktuellen PEP-Implementation?
5. Wir benötigen einen PDP, an den wir Requests senden können

Diskussion / Beschlüsse:

1. Review XACMLVoter:
   - Im XACMLVoter werden noch Casts verwendet (ReflectiveMethodInvocation) → besser: Wissen zentralisieren → ContextHandler soll sich darum kümmern, was für ein Objekt den Aufruf anforderte
   - Informationen weiterreichen an richtigen ContextHandler → brauchen Handler für jede mögliche Objektvariante
   - Empfehlung: *Strategy Pattern* benutzen
   - Von Spring übergebene Rollen weitergeben, obwohl vielleicht nicht benötigt
   - Methodenname ev. für Mapping benutzen → aus Methodenname auf URN schliessen, die Resource bezeichnet
   - Klassenname könnte Action bezeichnen
   - Antwort des PDP kommt in Form eines ResponseCtx (Positive, Negative, NotApplicable(d.h. keine zutreffende Policy gefunden), Indeterminate(Fehlerfall)) → diese Antwort mappt der Voter auf die entsprechende Antwort an Spring (Access Granted, Deny, Abstain)
   - Architektur mit Rücksicht auf Obligations entwerfen→ kann sein, dass in der Response neben einer Entscheidung auch eine Obligation zurückkommt → diese muss der Voter verarbeiten können (z.B. eine Fehlermeldung an den Benutzer zurückgeben)
2. Review RequestCtxFactory:
   - Eigene RequestCtxFactory wird nötig sein (die im Sample ist nur eine Demo, die nicht viel macht)
   - Empfehlung: Mapbasierte RequestCtxFactory
3. Datentypen werden erst später in der Entwicklung zum Thema → zurückgestellt

4. Bisherige Implementationen von einem PEP existieren bis auf einige Proof-of-Concept Prototypen keine
5. Einen rudimentären PDP wird wg zur Verfügung stellen.
6. Aufgaben für W04:
   - Varianten von Aufrufen entwerfen
     - action/subject/resource aus Name, Parameter, Annotations & EL-Expr., …
     - Dem Entwickler Möglichkeiten offenlassen, wie Aufruf aussieht → Was nicht aus Meth.name & Param. extrahiert werden kann → Annotations
     - Wie finde ich den korrekten CtxHandler für den aufgetretenen Aufruf?
   - Design des PEP (s. Graphik) verfeinern



Offene Punkte Verantwortung (erledigt vor nächster Sitzung):

Was	**Wer**
PDP zur Verfügung stellen	wg
Versch. CtxHandler entwerfen	sb, db
Design des PEP	sb, db

Kommende Abwesenheiten

Wer	Von	Bis	Bemerkung

Nächster Termin:
Datum: Rapperswil, den 11.10.2010
Zeit: 15:00

Sitzungsprotokoll

Projekt: **Heras-Spring Integration**
Woche: **4**
Datum: **Rapperswil, den 15.10.2010**
Dauer: **15:00-16:30**

Teilnehmer / Kürzel:
Daniel Bobst / db
Sandro Brändli / sb
Wolfgang Giersche / wg
René Eggenschwiler / re
Florian Huonder / fh

Traktanden:

1. Design des PEP besprechen
2. Implementation besprechen
3. Tasks W05

Diskussion / Beschlüsse:

1. Design des PEP
    1. Das Design haben sb und db vor dem Meeting von 13 – 14h bereits mit re und fh besprochen und erweitert.
    2. wg weist darauf hin, dass der Interceptor den weiteren Programmfluss steuert. Der Voter hat keine Kenntnis über den Programmablauf, erhält aber durch die vom PDP zurückgelieferte Obligations Anweisungen, die den Ablauf kontrollieren.
    → ObligationHandler so nicht machbar
    → Voter so nicht brauchbar
    3. Vorschlag: Eigenen DecisionManager oder Bridge zwischen AffirmativeBased DecisionManager und XACMLVoter implementieren
        - AffirmativeBased DecisionManager von Spring wirft AccessDeniedException, wenn gewisser int-Wert vom Voter zurückkommt
    4. ModelTransformer bezieht Daten aus Dictionary um technische Ausdrücke in formale umzuwandeln.

2. Implementation
    1. ModelTransformer vorläufig erst als Dummy belassen
    2. Mapping in XML verdrahten ist ok
    3. wg schlägt Benutzung von Annotations vor:
        1. Vorteil: Sind typesafe
        2. @Read, @Write, @SecureObject als Basis zur Verfügung stellen
        3. Expressions für Weiteres: @SecureObject → resourceid=..., resourcevalue=...
        4. MethodInvocation nach Interface fragen → dann Annotations etc. (Kontextinfos) vom Interface erfragen

4. Rest des Teams bevorzugt Lösung per EL-Expressions und @PreAuthorize → keine eigene Annotations definieren
5. Subject, Resourceclass, ResourceId, Action in XACML Request
6. ObligationHandler am DecisionManager ansetzen

3. Tasks nächste Woche (W05)
   - Wohlgeformten XACML Request an PDP schicken
     - Aufwandschätzung: 8h
   - Dummy Transformer erstellen
     - Aufwandschätzung: 0.5h
   - Annotations rausnehmen
     - Aufwandschätzung: 0.5h
   - Modell um DecisionManager erweitern (inkl. Exceptions)
     - Aufwandschätzung: 4h
   - EL Grundstruktur aufbauen
     - Aufwandschätzung: 3h
   - Doku schreiben (niedere Priorität)
     - Aufwandschätzung: 8h

Offene Punkte Verantwortung (erledigt vor nächster Sitzung):

Was	**Wer**
XACML Request an PDP schicken	sb, db
Dummy Transformer	sb, db
Annotations rausnehmen	sb, db
Modell um DecisionManager erweitern	sb, db
EL Grundstruktur einbauen	sb, db
Dokumentation	sb, db

Kommende Abwesenheiten

Wer	Von	Bis	Bemerkung

Nächster Termin:

Datum: Rapperswil, den 22.10.2010
Zeit: 15:00

Sitzungsprotokoll

Projekt: **Heras-Spring Integration**
Woche: **5**
Datum: **Rapperswil, den 22.10.2010**
Dauer: **15:00-17:00**

Teilnehmer / Kürzel:
Daniel Bobst / db
Sandro Brändli / sb
Wolfgang Giersche / wg
René Eggenschwiler / re

Traktanden:

1. Prototyp vorführen / besprechen
2. Datentypenproblematik besprechen
3. Weiteres Vorgehen

Diskussion / Beschlüsse:

1. **Protoyp**
   Das Ziel wurde erreicht. Der Prototyp entspricht dem definierten Design und hat den gewünschten Funktionsumfang.

   Es sollten noch einige Anpassungen gemacht werden:
   - Die Initializers (JAXBInitializer, etc.) sollen nicht direkt in unserem Code, sondern im Laufzeitkontext (z.B. Testcase) aufgerufen werden.
   - Es muss immer gegen Interfaces entwickelt werden.
   - Abhängigkeiten müssen injected werden. Allenfalls dürfen teilweise Default-Implementationen verwendet werden, falls nichts injected wird. Die DefaultImplementation des Transformers soll nur die Eingabe zurückliefern, da eine eigentliche Implementation des Transformers businessspezifisch ist.
   - Die Daten sollen transformiert werden, bevor sie zu den Transformables kommen. Dies soll in der HerasRequestFactory geschehen.
   - Die Klasse RequestCtxFactory von Heras ist in der heutigen Form obsolet und wird angepasst. Künftig soll sie einen RequestType bekommen und aus diesem den Request erzeugen.
   - Es soll ein ObligationHandler am DecisionManager angehängt werden. Fälle, in denen der aufrufende Code die Obligations behandeln muss, müssen noch genauer analysiert werden (Reentrant-Problematik).
   - Die Expressions sollen nach Möglichkeit in folgender Form angegeben werden können:

```
@PreAuthorize("{resources(" +
      "{'patientRecord', {#o.id.toString(), '5328'} }, " +
      "{'medicamentationList', {#b.id}})" +
      "}")
```

2. Datentypenproblematik

- Die Datentyp-URNs werden im Dictionary gespeichert.
- Der URNToDataTypeConverter kann daraus die Datentypen erstellen.
- Die Attribute müssen entsprechend in den RequestType abgefüllt werden. Der RequestContext erlaubt danach das Marshalling.

3. Weiteres Vorgehen

In W06 soll hauptsächlich die bisherige Arbeit dokumentiert werden. Dazu einige Tipps:

- Wichtig ist vor allem der Inhalt, nicht die Struktur.
- Kapitel sollen erst erstellt werden, wenn sie benötigt werden.
- Die Einleitung wird am besten erst am Schluss geschrieben.
- Bei Internetreferenzen muss der letzte Besuch vermerkt werden.

Es muss dokumentiert werden, wie sich die Spring-Annotations bei Vererbung und Interfaces verhalten.

4. Buildserver

- Um mit dem Buildserver zu arbeiten, müssen die Projekte im Filesystem hierarchisch strukturiert sein und die Abhängigkeiten über Module geregelt werden.

Offene Punkte Verantwortung (erledigt vor nächster Sitzung):

Was	**Wer**
Dokumentation	sb, db
RequestContextFactory anpassen	re, fh
Prototyp anpassen	sb, db

Kommende Abwesenheiten

Wer	Von	Bis	Bemerkung

Nächster Termin:

Datum: Rapperswil, den 29.10.2010
Zeit: 15:00

Sitzungsprotokoll

Projekt: **Heras-Spring Integration**
Woche: **6**
Datum: **Rapperswil, den 29.10.2010**
Dauer: **15:00-17:45**

Teilnehmer / Kürzel:
Daniel Bobst / db
Sandro Brändli / sb
Wolfgang Giersche / wg
René Eggenschwiler / re
Florian Huonder / fh

Traktanden:

1. Dokumentation besprechen
2. Buildserver Einrichtung
3. Anpassung ModelTransformer
4. Verwendung RequestType

Diskussion / Beschlüsse:

1. **Dokumentation besprechen**
   - Das Projektmanagement soll in den Anhang
   - Heras -> HERAS^{AF} (Autokorrektur)
   - Bei Themen jeweils mit einer Einleitung (Grobübersicht) beginnen. Die für die Arbeit wichtigen Themen vertiefen.
   - Abbildungs- und Tabellenverzeichnis gehören hinter das Inhaltsverzeichnis und sollen darin nicht aufgeführt werden.
   - Logos von HSR und HERAS^{AF} in den Header -> siehe Vorlage
   - Bei der Beschreibung von Begriffen wie PAP etc. Referenz auf IEEE-Definitionen einfügen.
   - Beim Request zuerst beschreiben, wofür er gebraucht wird.
     - Den Aufbau beschreiben
     - Die Semantik beschreiben
     - Im Design ins Detail gehen
   - Die Analyse beschreibt das „Big Picture“
   - Im Design sind genauere technische Details beschrieben.
   - Bei der Beschreibung des DecisionType nicht auf die Klasse, sondern auf die Spezifikation verweisen.
   - Einleitung schreiben, welche die Aufgabenstellung enthält.
   - Spring Analyse
     - -> Spring Security Analyse
     - Voting unter Autorisierung aufführen
     - Expression Handling ist wichtiger als die Bausteine zuvor

- Architektur
  - Einleitung mit „Big Picture“
  - Überlegungen zur Modulaufteilung
  - Nicht Spring beschreiben
  - Stil: Abkürzungen gelegentlich ausschreiben
  - 5.4: Erster Satz ist gleich ein Verweis auf anderes Kapitel
  - 5.4: Erklären wieso unsauber
  - Voter und DecisionManager dürfen erwähnt werden, jedoch keine konkreten Klassen.
- Im Design soll die Lösung im Detail beschrieben werden.
- Das Dokument soll auf einen Leser ausgerichtet werden, der nur Java kennt, also keine genauen Spring oder HERAS^{AF}-Kenntnisse hat.
- Sitzungsprotokolle in den Anhang
- Tipps zur Beschreibung des Projektmanagements
  - Agil
    - weil modern & gut
    - eignet sich für Forschung
  - Planungsphasen
    - Grössere Abschnitte mit Grobzielen beschreiben
    - Milestones definieren
  - Detailplanung
    - Task-Fokussiert
    - Jede Woche lauffähige Software & Doku als Ziel
- Allgemeine Struktur
  - Analyse (Problem & Technik)
    - Was ist …? Was tut es?
    - Wie funktionierts grundsätzlich
    - Wo ist es interessant für uns und warum?
    - Was ist zu berücksichtigen?
  - Architektur (Lösung)
    - Wie im grösserem Überblick?
    - Module & Zweck
    - Wie nutzen wir … um uns einzuklinken? (konzeptionell)
    - Entscheidungen Design
  - Design
    - Wie im Detail?
    - Packages, Klassen etc.
    - UML (Klassen-, Sequenzdiag.)

3. Buildserver

Das Projekt muss folgende Struktur haben (im Trunk):
herasaf-xacml-pep (parent: herasaf-xacml)
    herasaf-xacml-pep-core
    herasaf-xacml-pep-springsecextension
    herasaf-xacml-pep-integrationtests

4. **ModelTransformer**
   Zusätzlich zu den bisherigen Attributen soll ein Attribut für den Context hinzukommen. Der Context ist an den ContextHandler gebunden. Beispiele für den Context sind jsfContext oder methodContext.
   Zusätzlich zur DataType-URN muss auch der Issuer abgespeichert werden.

5. **Requesterzeugung**
   Die Requests sollen künftig folgendermassen zusammengestellt werden können:

```
RequestType req = new RequestType();
req.addSubjectAttribute(
    AttributeFactory.getAttribute(
        „issuer“,
        „att-id“,
        „type-urn“,
        value
    )
);
```

   value ist dabei vom Typ Object. Die Klasse von Value muss mit der Datatype-URN übereinstimmen, ansonsten wird eine Exception geworfen.

Offene Punkte Verantwortung (erledigt vor nächster Sitzung):

Was	**Wer**
Dokumentation gemäss Besprechung anpassen	sb, db
Mail an HERAS^{AF} Team welche Verschachtelung der Request-Attribute wir unterstützen	sb
Mail mit Beschreibung unseres Projekts an Mike Wiesner (CC re)	sb, db
Projektstruktur für Buildserver anpassen und Mail mit svn-Pfad an fh	sb
Software gemäss Besprechung anpassen (Context, Issuer)	sb, db

Kommende Abwesenheiten

Wer	Von	Bis	Bemerkung

Nächster Termin:
Datum: Rapperswil, den 5.11.2010
Zeit: 15:00

Sitzungsprotokoll

Projekt:	**Heras-Spring Integration**
Woche:	**7**
Datum:	**Rapperswil, den 5.11.2010**
Dauer:	**15:00-17:00**

Teilnehmer / Kürzel:
Daniel Bobst / db
Sandro Brändli / sb
Wolfgang Giersche / wg
René Eggenschwiler / re
Florian Huonder / fh

Traktanden:

1. Stand nach W06
2. Einrichtung Buildserver
3. Vorgehen W07

Diskussion / Beschlüsse:

1. **Stand nach W06**
   - Die Ziele für W06 konnten nicht erreicht werden.
     Das Refactoring des Projekts nahm erheblich mehr Zeit in Anspruch als vorgängig geschätzt. Dazu zählt einerseits die Anpassung an M3-SNAPSHOT und andererseits die Bereitmachung des Projekts für Continuous Integration.
2. **Einrichtung Buildserver**
   - Im Laufe der Sitzung konnte das Projekt erfolgreich auf dem Buildserver von HERAS[AF] eingerichtet werden.
3. **Vorgehen W07**
   - Die in W06 noch nicht erledigten Aufgaben bleiben bestehen für W07.
   - Zusätzlich soll der Fokus auf Qualitätssicherung gerichtet werden (Unit Tests / Coverage, Code Reviews)

Offene Punkte Verantwortung (erledigt vor nächster Sitzung):

Was	**Wer**
Dokumentation gemäss Besprechung W06 anpassen	sb, db
Software gemäss Besprechung anpassen (Context, Issuer)	sb, db
Qualitätssicherung (Unit Tests etc.)	sb, db

Kommende Abwesenheiten

Wer	Von	Bis	Bemerkung

Nächster Termin:

Datum: Rapperswil, den 12.11.2010
Zeit: 15:00

Sitzungsprotokoll

Projekt: **Heras-Spring Integration**
Woche: **8**
Datum: **Rapperswil, den 12.11.2010**
Dauer: **15:00-17:10**

Teilnehmer / Kürzel:
Daniel Bobst / db
Sandro Brändli / sb
Wolfgang Giersche / wg
Florian Huonder / fh

Traktanden:

1. Review Software
2. Struktur eines Requests
3. ObligationHandler
4. Testing

Diskussion / Beschlüsse:

1. **Review Software**
    1. HerasAuthorizationHandler: Benutzung über Interface entkoppeln
    2. DefaultSpringSecurityContextHandler als Beispiel implementieren
    3. Mit dem „aufrufenden Code“ ist der InvocationHandler gemeint.

    4. Ein bestimmter Aufrufkontext besitzt nur einen ContextHandler
        1. In verschiedenen Fällen eingesetzte AuthorizationHandlers werden jeweils mit dem geeigneten ContextHandler verdrahtet (Konfiguration durch Admin)
        2. Ein ContextHandler hat eine ContextId fest verdrahtet und weiss, wie der Aufrufkontext aussieht und wo sich welche Informationen befinden
            1. ContextId z.B.: App1_Method, App1_Web
        3. Behandlung von verschiedenen Teilen des Contexts (Subject Handling, Annotationen etc.) durch SubContextHandler (haben supports()-Methode o.ä.)
    5. Unterschiedliche Contexte werden von unterschiedlichen Instanzen des PEP bearbeitet.

    6. Transformer != Dictionary
    7. App.A und App.B können denselben Methodenname verwenden, aber das Dictionary muss verschiedene Sets zurückliefern -> brauche ContextId zur Identifizierung
    8. Transformer muss nicht nur Keys, sondern auch Values übersetzen können
    9. Transformer soll Set von AttributeTypes zurückliefern

2. **Struktur Request/Reply**
    1. Theoretisch sind mehrere Resource-Blöcke erlaubt
    2. Soll implementieren: 1 Resource mit mehreren Attribute-Blöcken
    3. Pro Attribut ein Value

4. Ausblick für nächste Arbeit: Mehrere Resource-Blöcke -> Mehrere Results
5. Es muss nur ein Result angeschaut werden

3. ObligationHandler

1. Interfaces in generischer Library, Implementation bei Kunde
2. PEP-Strategie allgemein definierbar → eine Strategie umsetzen
    1. Laut XACML gibt es Strategien: deny-biased, permit-biased, base
    2. Hat CannotUnderstand Exception und verschiedene CannotFulfillXYException
3. ObligationHandler an Strategie anhängen
4. Empfehlung: Base-Strategie implementieren

4. Testing

1. TestCase schreiben, der Inhalt von AccessDeniedException zeigt (soll Info aus Obligation beinhalten) → PEP-Strategie muss Strings in Exception packen
2. Testcases zu EL schreiben (subject, resource, action, env)
3. DB-Script in target unterbringen

Offene Punkte Verantwortung (erledigt vor nächster Sitzung):

Was	**Wer**
W09: ContextHandler/SubContextHandler implementieren	sb, db
W09: PEP-Strategie/Obligations implementieren	sb, db
W09: TestCases	sb, db
W10: Dokumentation nachführen	sb, db

Kommende Abwesenheiten

Wer	Von	Bis	Bemerkung

Nächster Termin:

Datum: Rapperswil, den 19.11.2010
Zeit: 15:00

Sitzungsprotokoll

Projekt: **Heras-Spring Integration**
Woche: **9**
Datum: **Rapperswil, den 19.11.2010**
Dauer: **15:00-17:00**

Teilnehmer / Kürzel:
Daniel Bobst / db
Sandro Brändli / sb
Wolfgang Giersche / wg
René Eggenschwiler / re
Florian Huonder / fh

Traktanden:

1. Review W09
2. Review Obligation Handling

Diskussion / Beschlüsse:

1. **Review W09**
   1. Das ContextHandling wurde gemäss Besprechung von letzter Woche implementiert.
   2. Der Transformer wurde eingeführt.
   3. DB-Scripts wurden in target-Ordner verschoben.
   4. Es wurde ein Test für die Expression Language eingeführt, welcher die bisher eingebauten Funktionen testet.
   5. Es wurden die Interfaces RequestFactory und AuthorizationHandler eingeführt.
   6. ObligationHandler wurden eingeführt.
   7. Der Transformer soll statt einem Set von AttributeTypes (Protokoll W08) einen RequestType zurückliefern.
      - Die Hilfsklasse TransformerResponse wird damit nicht mehr benötigt.
   8. Aufgabe W10: Sicherheit der implementierten EL untersuchen und Erkenntnisse dokumentieren.
   9. HerasRequestFactory soll die Erzeugung des RequestType an Transformer delegieren.
   10. Die Entscheidung, dass nur der erste vom PDP zurückgelieferte ResultType weiterverarbeitet wird, muss dokumentiert werden.
   11. Die Methode checkAccessAllowed des AuthorizationHandlers darf unter keinen Umständen eine Exception werfen. Diese Bedingung muss dokumentiert und durchgesetzt werden.
   12. In der Dokumentation sollen Behauptungen begründet oder als noch zu begründen markiert werden.

2. **Review Obligation Handling**
   1. PEPStrategy muss threadsafe implementiert werden, d.h. muss stateless sein.
   2. Die Listen cannotfulfillObligations und cannotunderstandObligations sollen der Methode treatObligations als final übergeben werden.

3. ObligationHandler müssen mehr als nur ein Boolean über Erfolg/Misserfolg zurückgeben können. Ein ObligationHandler soll ein ObligationHandlingResult zurückliefern.
   - Dieses Objekt beinhaltet: die ObligationId, Status (ok, cannotfulfill, cannotunderstand), eine Message und optional eine Exception als Begründung für einen allfälligen Misserfolg.
4. Die unübliche Implementation von PEPStrategyImpl und PEPStrategyBase soll angepasst werden. Momentan wird PEPStrategyImpl von PEPStrategyBase erweitert und Base verändert Membervariablen von Impl. Dies macht den Datenfluss schwer überschaubar.
5. PEPResponse soll eine Liste von Messages enthalten, zu der von verschiedenen ObligationHandlern beigetragen werden kann. Im DecisionManager sollen diese Messages in einer protected Methode zu einer einzigen kombiniert werden, die an Spring zur Anzeige beim Benutzer übergeben wird.
6. Methode applyStrategy soll umbenannt werden.
7. PEPStrategyBase umbenennen in BasePEPStrategy
8. PEPStrategyImpl umbenennen in AbstractPEPStrategy
9. Laut XACML-Spezifikation ist bei der Base-Strategie das Verhalten bei den Decisions NOT_APPLICABLE und INDETERMINATE undefiniert. In unserer Implementation soll in diesen Fällen die Decision DENY zurückgeschickt werden. Diese Entscheidung muss dokumentiert werden.
10. Es müssen keine weiteren konkreten ObligationHandler implementiert werden.

Offene Punkte Verantwortung (erledigt vor nächster Sitzung):

Was	**Wer**
Obligation Handling anpassen	sb, db
Dokumentation nachführen	sb, db
EL-Sicherheit analysieren	sb, db

Kommende Abwesenheiten

Wer	Von	Bis	Bemerkung
René Eggenschwiler	22.11.10	10.12.10	WK

Nächster Termin:

Datum: Rapperswil, den 26.11.2010
Zeit: 15:00

Sitzungsprotokoll

Projekt: **Heras-Spring Integration**
Woche: **10**
Datum: **Rapperswil, den 26.11.2010**
Dauer: **15:00-15:50**

Teilnehmer / Kürzel:
Daniel Bobst / db
Sandro Brändli / sb
Wolfgang Giersche / wg
Florian Huonder / fh

Traktanden:

1. Dokumentation besprechen
2. ObligationHandler besprechen

Diskussion / Beschlüsse:

1. Dokumentation
    1. Mehr Dokumentation
    2. Getting Started sollte in technischen Bericht aufgenommen werden.
    3. Getting Started soll ein HelloWorld-Projekt sein.
    4. Falls einfach möglich, Lesbarkeit von Diagrammen erhöhen.
    5. Wo sinnvoll Kommunikationsdiagramme benutzen
    6. Die Doku soll so geschrieben sein, dass jemand der Heras[AF] und Spring nicht kennt sie versteht.
    7. Javadoc für Public & Protected Klassen und Methoden schreiben.

2. ObligationHandler
    1. Message der Obligation selbst muss in AccessDeniedException verpackt werden.
    2. Tests für Obligation Handling einführen.

3. Allgemeine Punkte
    1. Es soll ein Showcase erstellt werden.
    2. Testfälle sollen realistischer gemacht werden (realistische Methodennamen/Parameter).
    3. Tests sollen kommentiert werden.
    4. Es müssen auch Fehlerfälle getestet werden.
    5. Für Fehler in der EL sollen sinnvolle Fehlermeldungen ausgegeben werden.
    6. Es muss auf korrekte Exceptions geprüft werden.

Offene Punkte Verantwortung (erledigt vor nächster Sitzung):

Was	**Wer**
Obligation Handling anpassen	sb, db
Dokumentation weiterführen	sb, db

Kommende Abwesenheiten

Wer	Von	Bis	Bemerkung

Nächster Termin:

Datum: Rapperswil, den 03.12.2010
Zeit: 15:00

Sitzungsprotokoll

Projekt: **Heras-Spring Integration**
Woche: **11**
Datum: **Rapperswil, den 03.12.2010**
Dauer: **15:00-16:00**

Teilnehmer / Kürzel:
Daniel Bobst / db
Sandro Brändli / sb
Wolfgang Giersche / wg
Florian Huonder / fh
René Eggenschwiler / re

Traktanden:

1. Review Tasks W10
2. Besprechung Dokumentation
3. Diverses

Diskussion / Beschlüsse:

1. Review Tasks W10
    1. Im Rahmen des „Getting Started“ Guide hat Sandro eine Beispielapplikation namens UserManager entwickelt.
    2. Schöner wäre Implementierung von UserManager gegen Interface, ist aber als Sample so ok.
    3. Prinzipiell wäre gedacht: subject -> user, action -> read, resource -> readAllUsers. Als Beispiel ist die jetzige Form aber ok.
    4. Implementierter ObligationHandler muss genau wissen, was er behandeln kann -> InfoObligationHandler
    5. Code und XML dokumentieren -> Javadoc
2. Besprechung Dokumentation
    1. Designrelevante Methoden müssen auch in der Papierdoku erklärt werden.
    2. In den Kapiteln Architektur und Design soll unsere Implementation erläutert werden. Es kann aber auf Erklärungen und Begründungen aus dem Kapitel Analyse verwiesen werden.
    3. High Level Übersicht fehlt
        1. Grob die Absicht der Arbeit erklären (Applikationen absichern, Zugriffe abfangen, z.B. durch AOP, was sind die Optionen dafür -> wir wollen eine bestehende Applikation instrumentieren)
        2. Fokus danach auf Interaktion PEP <-> PDP
        3. Beschreibung Module: Was geschieht wo? (PEP instrumentiert Applikation -> wie? AOP)
        4. Systemgrenzen in Architekturdiagramm einzeichnen
    4. Business Cases -> Neben Methoden auch Filter erwähnen (und erklären, dass aus Zeitgründen nur Methodensicherheit implementiert wurde)

5. Nach Erklärung der Umgebung (Heras, XACML, Spring) nochmal den Auftrag beschreiben -> Scope der Arbeit wird klarer
6. Code-Beispiele in Bericht nehmen -> Dokumentation wird verständlicher
7. Architektur -> Logische Architektur mit Diagrammm
8. Design -> Physikalische Architektur (Projekte)
9. Designentscheidungen erklären: z.B. Voter: Methodensignatur lässt Weitergabe von Informationen nicht zu -> Begründet, dass wir höher ansetzen müssen
10. Spezielle Anforderungen als eigenes Thema aus Architekturbeschreibung hervorheben
    1. Konkret: Transformer und Obligation Handling -> warum ein Transformer nötig ist und was er macht (Mapping)
11. Erklären wofür Obligations eigentlich gebraucht werden und dass wir sie für Messaging „missbrauchen“
12. „AuthorizationHandler ist zentrale Klasse des PEP“ -> erklären was damit gemeint ist
13. Tip für Dokureview: Gemeinsames Brainstorming, gegenseitig Zusammenhänge erklären -> ist die Doku verständlich

3. Diverses
    1. Sind Interfaces auch absicherbar? -> untersuchen und dokumentieren
    2. AttributeFactory kommt erst im M3 Release im Zug der besprochenen Convenience-API.

Offene Punkte Verantwortung (erledigt vor nächster Sitzung):

Was	**Wer**
Code kommentieren	sb, db
Papierdokumentation	sb, db

Kommende Abwesenheiten

Wer	Von	Bis	Bemerkung

Nächster Termin:

Datum: Rapperswil, den 10.12.2010
Zeit: 15:00

Sitzungsprotokoll

Projekt: **Heras-Spring Integration**
Woche: **12**
Datum: **Rapperswil, den 10.12.2010**
Dauer: **15:00-16:00**

Teilnehmer / Kürzel:
Daniel Bobst / db
Sandro Brändli / sb
Wolfgang Giersche / wg
Florian Huonder / fh
René Eggenschwiler / re

Traktanden:

1. Besprechung Dokumentation
2. Diverses

Diskussion / Beschlüsse:

1. Besprechung Dokumentation
    1. Der Technische Bericht wurde gemeinsam reviewt und an einigen Stellen verbessert. Insbesondere:
        1. Kap.2.2. Darauf hinweisen: hasAuthority postuliert bereits ein Securitymodell Kap.2.3.6: Nochmal erwähnen: hasPermission zwingt ein Securitymodell auf, muss nicht sein, dass ein Kunde dieses so umsetzen will
    2. Javadoc wurde stichprobenartig angeschaut und abgenommen. Einige Anmerkungen:
        1. HERAS[AF] Copyright am Anfang jedes Files
        2. Autoren in Javadoc schreiben
        3. Wo möglich Link auf XACML-Spez. (analog zu HERAS[AF])
2. Diverses
    1. Anfang W13: Abstract schreiben und an wg und re schicken zur Korrektur
    2. XacmlDecisionManager: Ev. @Autowired entfernen

Offene Punkte Verantwortung (erledigt vor nächster Sitzung):

Was	**Wer**
Doku fertigstellen	sb, db
Abstract schreiben	sb, db

Kommende Abwesenheiten

Wer	Von	Bis	Bemerkung

Nächster Termin:

Datum: Rapperswil, den 17.12.2010
Zeit: 15:00

Sitzungsprotokoll

Projekt:	**Heras-Spring Integration**
Woche:	**13**
Datum:	**Rapperswil, den 17.12.2010**
Dauer:	**13:00-13:50**

Teilnehmer / Kürzel:
Daniel Bobst / db
Sandro Brändli / sb
Wolfgang Giersche / wg

Traktanden:

1. Besprechung Dokumentation
2. Abgabe

Diskussion / Beschlüsse:

1. Besprechung Dokumentation
    1. Getting Started in gleiches Pdf wie Rest der Doku, darauf hinweisen dass Sprache wechselt
    2. Text des Abstracts am Anfang des Dokuments einfügen
    3. SubContextHandler: erklären wofür er gebraucht wird
    4. Transformer: Begriff Modelltransformer verwenden
    5. PEPStrategy: Strategien (Permit, Deny, Base) wenigstens kurz beschreiben
    6. Kap.3.2: Nicht Funktion erklären (schon an anderer Stelle erledigt) sondern warum aufgrund von Abhängigkeiten in separaten Projekten
    7. Wo nötig und sinnvoll: Entscheidungen
    8. Ausblick auseinandernehmen in „Offene Punkte“ (was aus Zeitgründen weggelassen wurde)
    9. Code in Fliesstext in anderem Font
    10. „Material für Präsentation“ aus Aufgabenstellung rausnehmen
2. Abgabe
    1. Für Thomas Letsch: 1 CD mit 1 Zip des Codes und 1 Pdf
    2. 3x Dokument inkl Getting Started -> Ausdruck in SW
    3. 1x CD an wg
    4. 2x CD an re & fh ?
    5. 1x CD für HSR
    6. Letztes Meeting zur Übergabe: 23.12.10 16:00h

Offene Punkte Verantwortung (erledigt vor nächster Sitzung):

Was	**Wer**
Doku fertigstellen	sb, db
Poster gestalten	sb, db

Kommende Abwesenheiten

Wer	Von	Bis	Bemerkung

Nächster Termin:

Datum: Rapperswil, den 23.12.2010
Zeit: 16:00

Getting Started

Inhaltsverzeichnis

Getting Started

With this guide you will create a simple application secured with a Spring Security Policy Enforcement Point (PEP) and a HERAS^{AF} Policy Decision Point (PDP). It will demonstrate how you can use policies to secure method calls.

It is recommended you also download the sample application. It has many comments that should help you to understand how to configure the PEP to meet your needs. To keep this document simple and readable, the comments have been removed from the source listings.

If you need more in depth explanations please refer to the documentation.

Requirements

To follow this guide, you should have the following software installed:

- Eclipse IDE for Java EE Developers (http://www.eclipse.org/downloads/moreinfo/jee.php)
- M2Eclipse Core (http://m2eclipse.sonatype.org/installing-m2eclipse.html)
- SpringSource Tool Suite 2.5 (Update Site: http://dist.springsource.com/milestone/TOOLS/update/e3.6)

Other configurations might work but are untested.

POM

1. Create a new simple Maven project.
2. Add the HERAS^{AF} and JBoss Repositories to the pom.xml:

```
<repository>
	<id>HERASAF Mirror</id>
	<url>http://maven.herasaf.org/repo/</url>
	<snapshots>
		<enabled>true</enabled>
	</snapshots>
</repository>
<repository>
	<id>JBoss</id>
	<name>JBoss Repository</name>
	<url>
	https://repository.jboss.org/nexus/content/repositories/public/
	</url>
</repository>
```

3. Add the needed dependencies to the pom.xml:

   For the PEP:

```
<dependency>
	<groupId>org.herasaf.xacml.pep.springsecextension</groupId>
	<artifactId>herasaf-xacml-pep-springsecextension</artifactId>
	<version>[0.0.0,]</version>
</dependency>
```

   For the database:

```
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-orm</artifactId>
```

```
		<version>[0.0.0,]</version>
	</dependency>
	<dependency>
		<groupId>org.hibernate</groupId>
		<artifactId>hibernate-entitymanager</artifactId>
		<version>3.6.0.Final</version>
	</dependency>
	<dependency>
		<groupId>org.hsqldb</groupId>
		<artifactId>hsqldb</artifactId>
		<version>[0.0.0,]</version>
	</dependency>
```

The database is needed for the transformer component of the PEP. The transformer translates technical names into formal ones, e.g. role -> urn:herasaf:user:role.

Source

1. Create a domain class:

```
package org.sample.secure;

public class User {
	private static int ID = 0;
	private String firstName;
	private String lastName;
	private int id = ID++;

	public User(String firstName, String lastName) {
		this.firstName = firstName;
		this.lastName = lastName;
	}

	public int getId() {
		return id;
	}

	public String getFirstName() {
		return firstName;
	}

	public String getLastName() {
		return lastName;
	}

	@Override
	public String toString() {
		return "User [firstName=" + firstName + ", lastName=" +
			lastName + "]";
	}
}
```

2. Add a secured business class:

   This class is used to store the information entered into the GUI we will create later. For an explanation on the usage of the expression please refer to the documentation.

```
package org.sample.secure;

public class UserManager {
	private Map<Integer, User> userMap = new HashMap<Integer, User>();

	@PreAuthorize("{actions({'type', {'read'}}), " +
```

```
            "resources({'user', {#id}})}")
      public User getUser(int id) {
            return userMap.get(id);
      }

      @PreAuthorize("{actions({'type', {'write'}}), " +
            "resources({'user', {#id}})}")
      public void updateUser(int id, User user) {
            userMap.put(id, user);
      }
}
```

3. Add a utility class:

   This class adds some entries to the transformer database and deploys policies. We will create the policies below.

```
package org.sample.secure;

public class Utilities {
      @Autowired
      private Dictionary transformer;
      @Autowired
      private PDP pdp;
      @Autowired
      private AuthenticationManager authenticationManager;
      private List<EvaluatableID> depPolicies = new ArrayList<EvaluatableID>();

      public void deployPolicy(String policyPath) {
            ((UnorderedPolicyRepository) pdp
                        .getPolicyRepository())
                        .undeploy(depPolicies);
            depPolicies.clear();
            InputStream policyFile = ClassLoader
                        .getSystemResourceAsStream(policyPath);
            Evaluatable policy = null;
            try {
                  policy = PolicyMarshaller.unmarshal(policyFile);
            } catch (SyntaxException e) {
            }
            depPolicies.add(policy.getId());
            UnorderedPolicyRepository repository = (UnorderedPolicyRepository)
                        pdp.getPolicyRepository();
            repository.deploy(policy);
      }

      public void setup() {
            Context context = new Context("method");
            transformer.addContext(context);
            transformer.addEntry(new Entry(context, "user",
                        "urn:herasaf:subject:username",
                        "User name",
                        "http://www.w3.org/2001/XMLSchema#string",
                        "spring-pep-group"));
            transformer.addEntry(new Entry(context,
                        "patientRecord",
                        "urn:herasaf:resource:patientRecord",
                        "Patient Record",
                        "http://www.w3.org/2001/XMLSchema#string",
                        "spring-pep-group"));
            transformer.addEntry(new Entry(context, "write",
                        "urn:herasaf:action:write", "Write",
                        "http://www.w3.org/2001/XMLSchema#string",
```

```
                        "spring-pep-group"));
                transformer.addEntry(new Entry(context, "envId",
                        "urn:herasaf:environment", "Environment",
                        "http://www.w3.org/2001/XMLSchema#string",
                        "spring-pep-group"));
                transformer.addEntry(new Entry(context, "role",
                        "urn:herasaf:user:role", "Environment",
                        "http://www.w3.org/2001/XMLSchema#string",
                        "spring-pep-group"));
                transformer.addEntry(new Entry(context, "method",
                        "urn:herasaf:method", "Method",
                        "http://www.w3.org/2001/XMLSchema#string",
                        "spring-pep-group"));
                transformer.addEntry(new Entry(context, "type",
                        "urn:herasaf:type", "Type",
                        "http://www.w3.org/2001/XMLSchema#string",
                        "spring-pep-group"));

                Authentication request = new UsernamePasswordAuthenticationToken(
                        "test", "test");
                final Authentication response = authenticationManager
                        .authenticate(request);
                SwingUtilities.invokeLater(new Runnable() {
                      public void run() {
                            SecurityContextHolder.getContext()
                                        .setAuthentication(response);
                      }
                });
                SecurityContextHolder.getContext()
                        .setAuthentication(response);
                deployPolicy("policies/PermitEverything.xml");
        }
}
```

4. Add a GUI-class:

```
package org.sample.secure;

public class Application extends JPanel {
      @Autowired
      private UserManager userManager;
      @Autowired
      private Utilities utilities;
      private JTable table = new JTable();
      private JTextField firstName = new JTextField(15);
      private JTextField lastName = new JTextField(15);

      public static void main(String[] args) throws Exception {
            UIManager.setLookAndFeel(UIManager
                        .getSystemLookAndFeelClassName());
            ClassPathXmlApplicationContext bf =
                  new ClassPathXmlApplicationContext("AppCtx.xml");
            JFrame mainWindow = new JFrame("Application");
            Application application = (Application) bf.getBean("app");
            application.init();
            mainWindow.setContentPane(application);
            mainWindow.setSize(600, 300);
            mainWindow.setVisible(true);
            mainWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }

      public void init() {
            utilities.setup();
            initGui();
```

```
	}

	private void initGui() {
		setLayout(new BorderLayout());
		userManager.addUser(new User("John", "Doe"));
		userManager.addUser(new User("Jane", "Doe"));
		updateTable();
		JScrollPane panel = new JScrollPane(table);
		add(panel, BorderLayout.CENTER);
		JPanel panel1 = new JPanel(new FlowLayout(
				FlowLayout.LEFT, 2, 2));
		JComboBox comboBox = new JComboBox(new String[] {
				"PermitEverything", "DenyEverything",
				"DenyWithObligations" });
		comboBox.addItemListener(new ItemListener() {
			public void itemStateChanged(ItemEvent e) {
				if (e.getStateChange() == ItemEvent.SELECTED) {
					utilities.deployPolicy("policies/"
							+ e.getItem() + ".xml");
				}
			}
		});
		panel1.add(new JLabel("Policy:"));
		panel1.add(comboBox);
		panel1.add(new JLabel("First name:"));
		panel1.add(firstName);
		panel1.add(new JLabel("Last name:"));
		panel1.add(lastName);
		JButton addButton = new JButton("Add");
		addButton.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				try {
					userManager.addUser(new User(firstName
							.getText(), lastName.getText()));
					updateTable();
				} catch (AccessDeniedException ex) {
					String message = ex.getMessage();
					JOptionPane.showMessageDialog(
							Application.this,
							"".equals(message) ? "Access Denied"
									: message);
				}
			}
		});
		panel1.add(addButton);
		add(panel1, BorderLayout.NORTH);
	}

	private void updateTable() {
		Vector<Vector<String>> rows = new Vector<Vector<String>>();
		for (User user : userManager.getUsers().values()) {
			Vector<String> cols = new Vector<String>();
			cols.add("" + user.getId());
			cols.add(user.getFirstName());
			cols.add(user.getLastName());
			rows.add(cols);
		}
		List<String> header = Arrays.asList(new String[] {
				"id", "First name", "Last name" });
		table.setModel(new DefaultTableModel(rows,
				new Vector<String>(header)));
	}
}
```

5. Add an obligation handler:

   Every policy may have obligations. An obligation handler's purpose is to fulfill these obligations. This obligation handler understands obligations of the type "urn:sample:basic:obligations:information".

```
package org.sample.secure;

public class SampleObligationHandler implements ObligationHandler {
	public boolean canHandle(ObligationType obligation) {
		return "urn:sample:basic:obligations:information"
				.equals(obligation.getObligationId());
	}

	public ObligationHandlingResult handle(ObligationType obligation) {
		String attributeValue = obligation.getAttributeAssignments()
				.get(0).getContent().get(0).toString();
		attributeValue += "\n" + obligation.getAttributeAssignments()
				.get(1).getContent().get(0).toString();
		ObligationHandlingResult result = new ObligationHandlingResult(
				obligation.getObligationId(), ObligationStatus.OK,
attributeValue);
		return result;
	}
}
```

Spring Configuration

1. Configure application security using Spring (src/main/resources/SecCtx.xml):

   This file contains all the configuration needed for the PEP to work.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:sec="http://www.springframework.org/schema/security"
xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:util="http://www.springframework.org/schema/util"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
            http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
            http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
            http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security.xsd
            http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
            http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.0.xsd">


  <context:annotation-config />

  <bean id="entityManagerFactory"
    class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
    <property name="persistenceUnitName" value="sample" />
  </bean>

  <bean id="dictionary" class="org.herasaf.transformer.JpaDictionary" />
```

```
  <tx:annotation-driven transaction-manager="transactionManager" />

  <bean id="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory" />
  </bean>

  <bean id="accessDecisionManager"
    class="org.herasaf.pep.springsecextension.XacmlDecisionManager" />

  <sec:global-method-security
    access-decision-manager-ref="accessDecisionManager"
    pre-post-annotations="enabled" secured-annotations="enabled" />

  <sec:authentication-manager>
    <sec:authentication-provider>
      <sec:user-service>
        <sec:user name="test" password="test" authorities="ROLE_USER" />
        <sec:user name="supervisor" password="supervisor"
          authorities="ROLE_SUPERVISOR" />
      </sec:user-service>
    </sec:authentication-provider>
  </sec:authentication-manager>


  <bean id="sampleObligationHandler"
class="org.sample.secure.SampleObligationHandler" />
  <bean id="pepstrategy"
class="org.herasaf.xacml.pep.core.strategy.BasePEPStrategy">
    <property name="obligationHandlers">
      <list>
        <ref bean="sampleObligationHandler" />
      </list>
    </property>
  </bean>

  <bean id="authHandler"
class="org.herasaf.xacml.pep.core.HerasAuthorizationHandler">
    <property name="contextHandler">
      <ref bean="ctxHandler" />
    </property>
    <property name="strategy">
      <ref bean="pepstrategy" />
    </property>
    <property name="pdp" ref="pdp" />
    <property name="requestFactory" ref="herasRequestFactory" />
  </bean>

  <bean id="ctxHandler"
    class="org.herasaf.pep.springsecextension.SpringMethodContextHandler">
    <property name="methodInvocationHandlers">
      <bean class="org.herasaf.pep.springsecextension.MethodInvocationHandler"
/>
    </property>
    <property name="configAttributesHandlers">
      <bean class="org.herasaf.pep.springsecextension.SpelHandler" />
    </property>
    <property name="contextId" value="method" />
  </bean>

  <bean id="herasRequestFactory"
class="org.herasaf.xacml.pep.core.HerasRequestFactory">
    <property name="transformer" ref="transformerService" />
  </bean>
```

```
  <bean id="transformerService"
class="org.herasaf.transformer.ResolveAttributesTransformer">
    <property name="dictionary" ref="dictionary" />
  </bean>

  <bean id="pdp" class="org.herasaf.xacml.core.simplePDP.SimplePDPFactory"
    factory-method="getSimplePDP" />

</beans>
```

2. Configure application (src/main/resources/AppCtx.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:sec="http://www.springframework.org/schema/security"
xmlns:tx="http://www.springframework.org/schema/tx"
      xmlns:util="http://www.springframework.org/schema/util"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
            http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
            http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
            http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security.xsd
            http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
            http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.0.xsd">

      <context:annotation-config />

      <import resource="SecCtx.xml" />

      <bean id="app" class="org.sample.secure.Application" />

      <bean class="org.sample.secure.Utilities" />

      <bean class="org.sample.secure.UserManager" />
</beans>
```

Database

1. Configure database (src/main/resources/META-INF/persistence.xml):

   The database is used by the transformer to store its entries.

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
  <persistence-unit name="sample">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <class>org.herasaf.transformer.Entry</class>
    <class>org.herasaf.transformer.Context</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.jdbc.driver"
        value="org.hsqldb.jdbcDriver" />
```

```
      <property name="javax.persistence.jdbc.user"
        value="sa" />
      <property name="javax.persistence.jdbc.password"
        value="" />
      <property name="javax.persistence.jdbc.url"
        value="jdbc:hsqldb:file:target/test;shutdown=true" />
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.HSQLDialect" />
      <property name="hibernate.max_fetch_depth"
        value="3" />
      <property name="hibernate.dialect"
        value="org.hibernate.dialect.HSQLDialect" />
      <property name="hibernate.hbm2ddl.auto"
        value="create" />
      <property name="hibernate.show_sql" value="false" />
    </properties>
  </persistence-unit>
</persistence>
```

Logging

Add logging configuration (src/main/resources/logback.xml):

```
<configuration>
  <appender name="STDOUT"
    class="ch.qos.logback.core.ConsoleAppender">
    <layout class="ch.qos.logback.classic.PatternLayout">
      <Pattern>%d{HH:mm:ss.SSS} [%thread] %-5level
        %logger{36} - %msg%n
                  </Pattern>
    </layout>
  </appender>

  <root level="INFO">
    <appender-ref ref="STDOUT" />
  </root>

  <logger name="org.hibernate" level="ERROR" />
</configuration>
```

Policies

Add some policies:

src/main/resources/policies/DenyEverything.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:permit-overrides"
  PolicyId="DenyEverything" Version="1.0"
  xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
           http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-
schema-os.xsd">
  <Description>This policy denies access generally. It is to be overridden by
exceptions</Description>
  <Target />
  <Rule Effect="Deny" RuleId="DenyByDefault" />
</Policy>
```

src/main/resources/policies/DenyWithObligations.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:permit-overrides"
  PolicyId="AllowWithObligations" Version="1.0"
  xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
           http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-
schema-os.xsd">
  <Description>This policy denies access and additionally contains
Obligations.</Description>
  <Target />
  <Rule Effect="Deny" RuleId="DenyWithObligations" />
  <Obligations>
    <Obligation ObligationId="urn:sample:basic:obligations:information"
      FulfillOn="Deny">
      <AttributeAssignment
        AttributeId="urn:sample:basic:obligations:policyId"
        DataType="http://www.w3.org/2001/XMLSchema#string">DenyWithObligations</
AttributeAssignment>
      <AttributeAssignment
        AttributeId="urn:sample:basic:obligations:InfoText"
        DataType="http://www.w3.org/2001/XMLSchema#string">You are not allowed
to do this.
      </AttributeAssignment>
    </Obligation>
  </Obligations>
</Policy>
```

src/main/resources/policies/PermitEverything.xml:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:permit-overrides"
      PolicyId="PermitEverything" Version="1.0"
      xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:policy:schema:os
           http://docs.oasis-open.org/xacml/access_control-xacml-2.0-policy-
schema-os.xsd">
           <Description>This policy grants access generally. It is to be
overridden by exceptions</Description>
      <Target />
      <Rule Effect="Permit" RuleId="PermitByDefault"/>
</Policy>
```

Expression Language

Secured Methods are annotated with Expressions:

```
@PreAuthorize("{actions({'type', {'read', #amount}})}")
public void deposit(Integer amount);
```

The language used for the expressions is the Spring Expression Language[1]. If you are not familiar with it, please refer to its Documentation, as we don't explain it here.

The Expression is evaluated with a root object of type *org.herasaf.pep.springsecextension.AuthorizationContextElAdapter*. The *Authentication* is set as a variable with the name *authentication*. You can also access the method parameters as variables with their parameter name preceded by a `#` .

1 http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/expressions.html

The Expression represents basically a list of calls to methods of the root object:

```
{actions(...), environment(...), resources(...), subjects(...)}
```

The methods expects a list as parameter. The list should contain the attribute name, followed by a list of attribute values. The list can contain multiple attributes and their values:

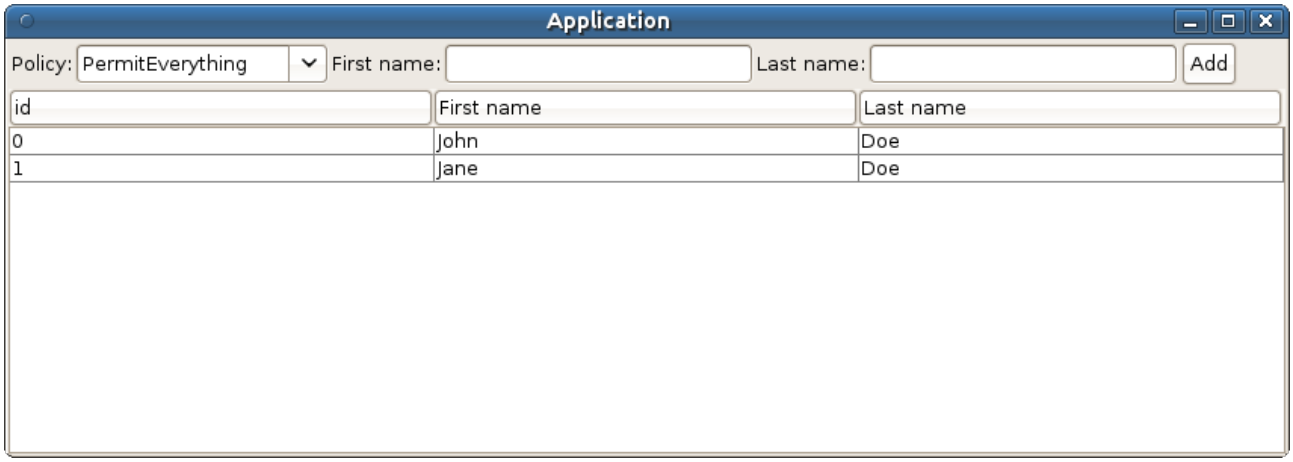
```
actions({'type', {'read', 'write'}, 'category', {'update'}})
```

After transformation the action attribute in the request could look like the following:

```
<Action>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:herasaf:sample:type">
      <AttributeValue>read</AttributeValue>
    </Attribute>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:herasaf:sample:type">
      <AttributeValue>write</AttributeValue>
    </Attribute>
    <Attribute DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="urn:herasaf:sample:category">
      <AttributeValue>update</AttributeValue>
    </Attribute>
</Action>
```

Demonstration

When you start the application, you'll see the following window:

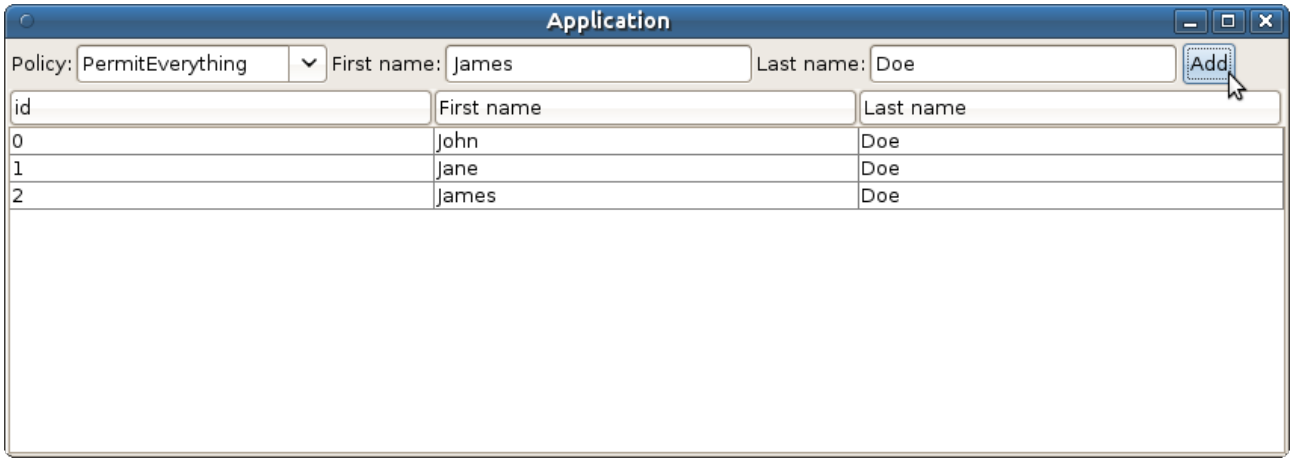


This sample Application only supports to add entries, not to delete nor to edit them. The method to add an entry and the method to get a list of all entries are secured.

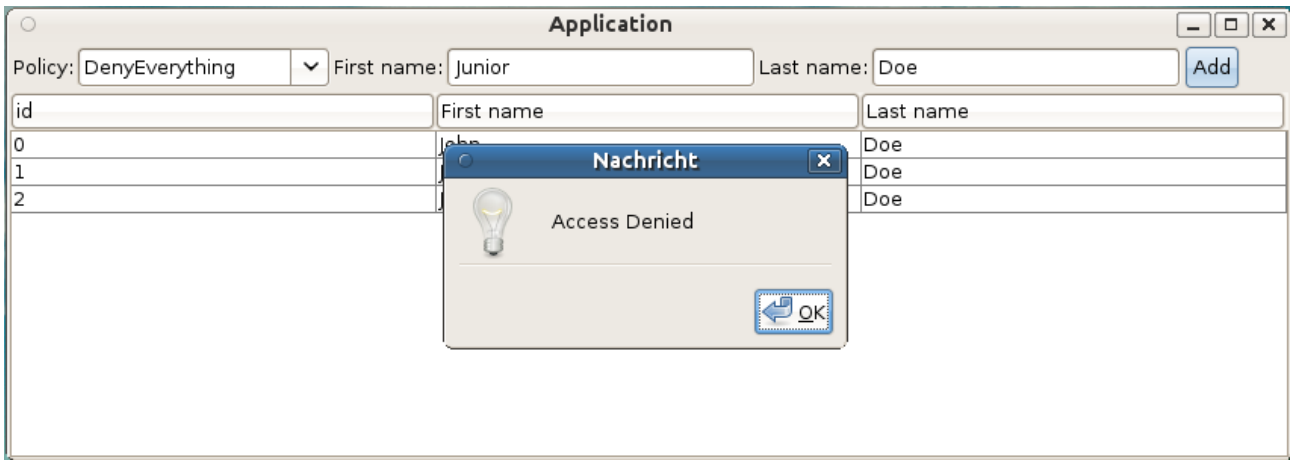
Note that the PermitEverything policy is selected by default in the dropdown list in the top left corner.

Try adding a new entry to the list by entering a first and last name and clicking the "Add" button.

As expected, the name is added to the list:

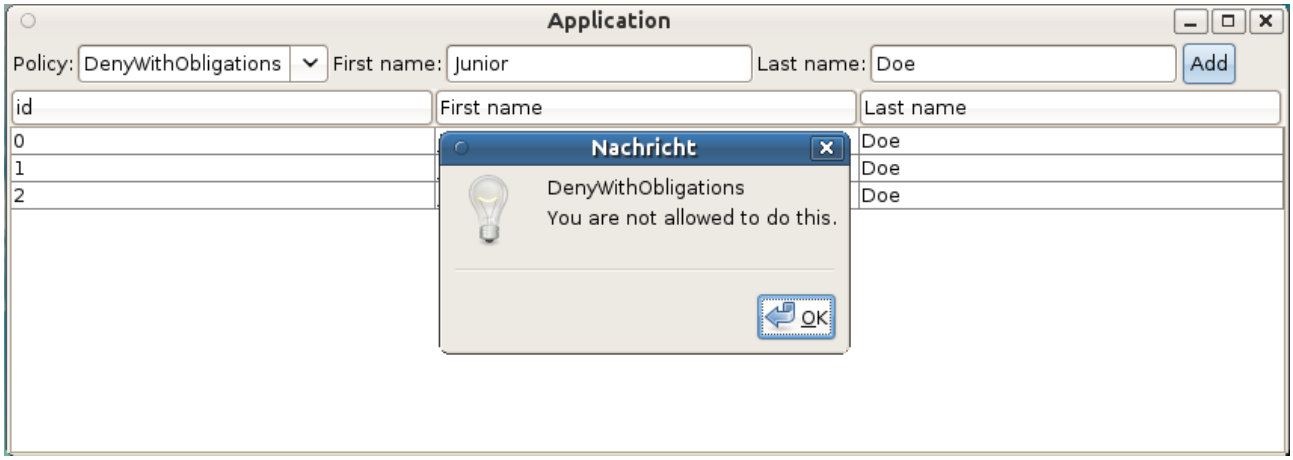


Change the active policy to DenyEverything and try adding another name:



As you can see, the policy denies execution of the Add function.

Finally, change the active policy to DenyWithObligations and try adding the name:



Not only is using the Add button denied, but the error message contains the info text we defined in the policy. This demonstrates how information contained in the obligation can be displayed to the user.

1. Auftraggeber

Verantwortlicher Dozent:

- Wolfgang Giersche, Dozent für Informatik HSR

Betreuer:

- René Eggenschwiler

2. Ausgangslage, Problembeschreibung

Im Zuge des Open Source Projekt HERAS[AF] der Hochschule Rapperswil ist in den letzten Jahren eine Softwarekomponente "Policy Decision Point" (PDP) entwickelt worden, die es erlaubt, Zugriffsentscheidungen für praktisch beliebige Softwareapplikationen anhand einer beliebigen Menge von formalen Richtlinien basierend auf dem OASIS Standard XACML 2.0 zu treffen. Applikationen, die diesen Service nutzen wollen, müssen Zugriffe auf schützenswerte Ressourcen (Methoden, URLs, Domain-Objekte) in einer für die eingesetzte Technologie angemessenen Weise abfangen, sicherheitsrelevante Attribute wie Ressourcenbezeichner, Benutzer-ID, Rollen und sonstige Daten aus dem Kontext ermitteln und eine formale Anfrage (XACML-Request) an den PDP schicken. Innerhalb der Applikation kann dann auf negative Entscheidungen spezifisch reagiert werden

In der Entwicklung mit Java hat sich in den vergangenen Jahren das Springframework (heute in Version 3.0) als vollwertige Alternative zum Java Enterprise Standard etabliert. Das Springframework bietet eine grosse Zahl von hilfreichen Bibliotheken zur Unterstützung der verschiedensten Architekturen und Komponenten. Insbesondere stellt die Springframework Security Bibliothek Mittel zur programmatischen oder deklarativen Absicherung technischer Ressourcen zur Verfügung. Dabei werden die Regeln aber an vielen Stellen im Quellcodebestand selbst niedergelegt oder aber mindestens gewisse Vorentscheidungen getroffen. Durch die konsequente Delegation aller Zugriffsentscheidungen sollte es möglich sein, Spring 3.0 - basierte Applikationen so abzusichern, dass die gültigen Zugriffsregeln während der Applikationslaufzeit und nicht während der Entwicklung bestimmt werden können.

3. Aufgabenstellung

Ziel der Arbeit ist die Integration der HERAS[AF] XACML Engine in das Spring Security Framework. Dazu müssen zwei Bibliotheken (JAR-Files) entwickelt werden, von denen die eigentliche Integrationsbibliothek sowohl von HERAS[AF] als auch von Spring Security abhängig sein darf und die zweite "generische" Bibliothek all den Code enthält, der nicht von Spring Security abhängig ist. So kann die zweite Bibliothek für den Einsatz in anderen Technologien (etwa Java

Enterprise, EJB) wiederverwendet werden. Die gewünschte Zielarchitektur wird in der ersten Besprechung grob festgelegt und in den folgenden Sitzungen (gegebenenfalls aufgrund neuer Erkenntnisse) immer weiter verfeinert.

Spring Integration Bibliothek

In dieser Komponente werden hauptsächlich solche Klassen entwickelt, die sich in die Springframework Extension Points integrieren. Aus dem Spring-spezifischen Kontext soll eine Technologie-agnostische Darstellung des Zugriffsversuches als abstrakte Requests extrahiert werden. Diese Darstellung wird dann an ein wohldefiniertes Interface aus der generischen Bibliothek übergeben. Die zurückerhaltene generische Zugriffsentscheidung wird dann hier jeweils kontextspezifisch zu interpretieren sein.

Generische Bibliothek

In der generischen Komponente sollen die aus der Integrationsschicht kommenden Requests übersetzt werden in formale XACML Requests und als solche an den PDP geschickt werden. Die XACML Antwort des PDP wird in eine abstrakte Form überführt und an den Aufrufenden zurückgegeben. Für diese Arbeit soll die Übersetzung jeweils gegen Interfaces entwickelt werden, zu denen mindestens eine simple Default-Implementierung zu erstellen ist.

4. Zur Durchführung

Die Bearbeitung der Aufgabe erfordert eine Einarbeitung das Springframework und die Grundlagen des XACML Standards.

Solide Programmierkenntnisse in Java und die Bereitschaft, sich rasch neues Domänenwissen anzueignen, werden vorausgesetzt.

Wegen der variablen Breite der Aufgabe eignet sich das Thema gut für eine Bearbeitung durch zwei Studierende.

Mit dem Betreuer finden in der Regel wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen.

Alle Besprechungen sind von den Studenten mit einer Traktandenliste vorzubereiten und die Ergebnisse sind in einem Protokoll zu dokumentieren, das dem Betreuer per E-Mail zugestellt wird.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhalten die Studierenden ein vorläufiges Feedback. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Implementierung und Dokumentation.

5. Dokumentation und Abgabe

Die Studierenden dürfen als Sprache der technischen Dokumentation Englisch oder Deutsch wählen.

Die Dokumentation zur Projektplanung und –verfolgung ist gemäss den Richtlinien der Abteilung Informatik in deutscher Sprache anzufertigen. Die Detailanforderungen an die Dokumentation der Recherche- und Entwicklungsergebnisse werden innerhalb des konkreten Arbeitsplanes festgelegt.

Die Dokumentation ist vollständig auf CD in drei Exemplaren abzugeben.

Neben der Dokumentation sind abzugeben:

- ein Poster zur Präsentation der Arbeit
- alle zum Nachvollziehen der Arbeit notwendigen Ergebnisse und Daten (Quellcode, Buildskripte, Testcode, Testdaten usw.)
- Material für eine Abschlusspräsentation (ca. 20’)

6. Termine

Die wichtigsten Termine werden mit den Studierenden zu Beginn der Arbeit vereinbart.
Siehe auch Terminplan auf https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html

7. Literaturhinweise

8. Beurteilung

Eine erfolgreiche Studienarbeit erhält 8 ECTS-Punkten (1 ECTS Punkt entspricht einer Arbeitsleistung von ca. 25 bis 30 Stunden). Für die Modulbeschreibung der Studienarbeit siehe
https://unterricht.hsr.ch/staticWeb/allModules/10938_M_SAI.html

Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/5
2. Berichte (Abstract, Mgmt Summary, techn. u. persönliche Berichte) sowie Gliederung, Darstellung, Sprache der gesamten Dokumentation	1/5
3. Inhalt*)	3/5

*) Die Unterteilung und Gewichtung von des Inhaltes wird im Laufe dieser Arbeit mit dem Studenten vereinbart

Im Übrigen gelten die Bestimmungen der Abt. Informatik zur Durchführung von Studienarbeiten.

Rapperswil, der 6. August 2010

Der verantwortliche Dozent

Wolfgang Giersche
Dozent für Informatik, HSR