Recommendation Engines

*Projektarbeit*

Lars Diestelhorst  
Am Centrumshaus 2  
21073 Hamburg  
+49 40 7654884  
diestelhorst@tu-harburg.de

Betreuung:

Prof. Dr. Florian Matthes,  
Arbeitsbereich Softwaresysteme  
TECHNISCHE UNIVERSITÄT HAMBURG-HARBURG

23. Juli 2001
Zusammenfassung

# Inhaltsverzeichnis

1 Einleitung

2 Personalisierung

2.1 Was ist Personalisierung? ........................................ 3
   2.1.1 Die individuelle Anpassung von Inhalten .................. 3
   2.1.2 Das Treffen der besten Entscheidung .................... 4
2.2 Nutzen der Personalisierung .................................. 5
   2.2.1 Nutzen für den Anwender .................................. 5
   2.2.2 Nutzen für den Anbieter .................................. 6
2.3 Datenschutz ..................................................... 6
2.4 Formen von Personalisierung .................................. 7

3 Der Weg zum Wissen

3.1 Web Mining .................................................... 9
   3.1.1 Web Content Mining ........................................ 9
   3.1.2 Web Structure Mining ...................................... 10
   3.1.3 Web Usage Mining ......................................... 10
3.2 Knowledge Discovery in Databases ........................... 11
   3.2.1 Definition von Knowledge Discovery in Databases ...... 12
   3.2.2 Prozessbeschreibung ....................................... 12
3.3 Ziele des KKD-Prozesses ..................................... 14
3.4 Interdisziplinarität ............................................. 15

4 Data Mining-Algorithmen ...................................... 17

4.1 Klassifizierende Algorithmen ................................. 17
   4.1.1 Nearest Neighbour ....................................... 18
   4.1.2 K-Nearest Neighbour ..................................... 18
4.2 Segmentierende Algorithmen ................................ 19
   4.2.1 Vector Quantization ....................................... 21
4.3 Abhängigkeitsentdeckende Algorithmen ..................... 22
   4.3.1 Assoziationsregeln ....................................... 22
## 5 Recommendation Systems

5.1 Aufgabe .................................................. 25
5.2 Grundannahmen - verwendete Heuristiken ......................... 26
5.3 Prädferenzen .............................................. 26
  5.3.1 Probleme in der Datendarstellung ................................ 28
  5.3.2 Neighbourhood formation .................................. 29
  5.3.3 Generierung der Empfehlung ................................ 30

## 6 Design und Implementierung

6.1 Aufgabe .................................................. 31
  6.1.1 Der infoAsset Broker ..................................... 31
  6.1.2 Das Recommendation System ................................ 33
6.2 Module des Recommendation System ................................. 34
  6.2.1 Hauptanwendungsfälle ..................................... 34
  6.2.2 Aktualisierung der Nutzernachbarschaft ....................... 36
  6.2.3 Pakete des Recommendation System .......................... 37
6.3 Identifizierung von Klassen ................................... 38
  6.3.1 Asset .................................................... 38
  6.3.2 AssetContainer .......................................... 39
  6.3.3 Person (Person) .......................................... 40
  6.3.4 Gruppe (Group) .......................................... 40
  6.3.5 Mitgliedschaft (Membership) ................................ 40
  6.3.6 Kategorie (Category) ..................................... 40
  6.3.7 Kategorisierung (Categorization) ............................ 41
  6.3.8 Klassifizierung (Classification) ............................. 41
  6.3.9 Klassifikator (Classifier) .................................. 42
  6.3.10 Bewertung (Rating) ....................................... 42
  6.3.11 Beziehung (Relationship) .................................. 42
  6.3.12 Recommendation Engine ................................... 43
6.4 Beziehungen der Klassen untereinander ............................. 44
6.5 Ereignisbehandlung im Recommendation System .................... 47

## 7 Schluss

## A Package de.infoasset.broker.interfaces

A.1 Interfaces ................................................. 54
  A.1.1 INTERFACE Membership .................................... 54
  A.1.2 INTERFACE MembershipEvent ............................... 56
  A.1.3 INTERFACE Memberships .................................. 57
  A.1.4 INTERFACE MembershipsListener ............................ 60
  A.1.5 INTERFACE Relationship .................................... 60
  A.1.6 INTERFACE RelationshipEvent ............................... 66
  A.1.7 INTERFACE Relationships .................................. 69
A.1.8 Interface RelationshipsListener ........................................... 79

B Package de.infoasset.broker.interfaces.recommendation 83
   B.1 Interfaces ........................................................................ 84
      B.1.1 Interface Rating ................................................... 84
      B.1.2 Interface RatingEvent ......................................... 87
      B.1.3 Interface Ratings ............................................... 89
      B.1.4 Interface RatingsListener ................................. 93
      B.1.5 Interface RecommendationEngine .................... 94
      B.1.6 Interface RecommendationEngines .................. 98

C Package de.infoasset.broker.interfaces.classification 103
   C.1 Interfaces .................................................................... 104
      C.1.1 Interface Categorization ................................. 104
      C.1.2 Interface Categorizations ............................. 105
      C.1.3 Interface Category ....................................... 107
      C.1.4 Interface Classification .................................. 108
      C.1.5 Interface Classifier ....................................... 112
      C.1.6 Interface Classifiers .................................... 116
INHALTSVERZEICHNIS
## Abbildungsverzeichnis

<table>
<thead>
<tr>
<th>Kapitel</th>
<th>Seite</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 KDD - Prozessmodell</td>
<td>11</td>
</tr>
<tr>
<td>3.2 Ziele des KDD-Prozesses</td>
<td>14</td>
</tr>
<tr>
<td>3.3 KDD als Interdisziplinäres Forschungsgebiet</td>
<td>15</td>
</tr>
<tr>
<td>4.1 Nearest Neighbour</td>
<td>19</td>
</tr>
<tr>
<td>4.2 K-Nearest Neighbour</td>
<td>20</td>
</tr>
<tr>
<td>4.3 Vector Quantization</td>
<td>20</td>
</tr>
<tr>
<td>5.1 Recommendation als individuelle Sortierung von Elementen</td>
<td>25</td>
</tr>
<tr>
<td>6.1 Die Client-Server Architektur des infoAsset Brokers</td>
<td>32</td>
</tr>
<tr>
<td>6.2 Die Komponenten des Application-Server</td>
<td>33</td>
</tr>
<tr>
<td>6.3 Die Use-Cases des Recommendation System</td>
<td>35</td>
</tr>
<tr>
<td>6.4 Die neuen Pakete des infoAsset Brokers</td>
<td>38</td>
</tr>
<tr>
<td>6.5 Persistenz- und Assetmanagement des infoAsset Brokers</td>
<td>39</td>
</tr>
<tr>
<td>6.6 Klassendiagramm der Recommendation Engines</td>
<td>45</td>
</tr>
<tr>
<td>6.7 Realisierung der Categorizations</td>
<td>46</td>
</tr>
<tr>
<td>6.8 Konzeptuelles Klassendiagramm: Zuordnung von Assets zu Kategorien und Kategorisierungen</td>
<td>47</td>
</tr>
<tr>
<td>6.9 Umsetzung der Zuordnung von Assets zu Kategorien und Kategorisierungen am Beispiel von Gruppen und Personen</td>
<td>48</td>
</tr>
<tr>
<td>6.10 Nachrichtensequenz beim Eventmechanismus</td>
<td>49</td>
</tr>
</tbody>
</table>
carpe diem
Kapitel 1

Einleitung


---

\(^1\)http://www.amazon.de
\(^2\)http://www.einkauf24.de
\(^3\)http://www.peopleunited.de
\(^4\)http://www.citeseer.com
\(^5\)http://www.welt.de
KAPITEL 1. EINLEITUNG


⁶Der infoAsset Broker ist ein Produkt der infoAsset AG, Harburger Schlossstr. 6-12, 21079 Hamburg
Kapitel 2

Personalisierung


2.1 Was ist Personalisierung?

2.1.1 Die individuelle Anpassung von Inhalten


\subsection*{2.1.2 Das Treffen der besten Entscheidung}

Werden die Inhalte einer Web-Site für jeden Nutzer individuell zusammengestellt, muss aus einer großen Zahl von Inhalten ausgewählt werden. Neben der Auswahl der einzelnen Inhalte muss zusätzlich die Anordnung festgelegt werden. Dabei sollte die für den Nutzer wichtigste Information an der ersten Stelle stehen. Das Software-System muss also bei jedem Request darüber entscheiden, welche Zusammenstellung für einen Nutzer die Richtige ist und wie die Elemente der Seite angeordnet werden sollen. Personalisierung kann man also auch als das Treffen der besten Entscheidung für den jeweiligen Nutzer sehen. Bei technischen Aspekten der Personalisierung ist das Treffen der Entscheidung recht einfach. Der Nutzer bekommt die Möglichkeit, die WWW-basierte Anwendung nach seinen Bedürfnissen zu konfigurieren. Diese Konfiguration wird gespeichert und steht bei jedem neuen Besuch des Kunden wieder zur Verfügung\footnote{Eine technische Umsetzung ist sowohl über Cookies, als auch über eine explizite Nutzeranmeldung möglich.}. Neben dieser manuellen Konfiguration können auch Informationen verwendet werden, die beim Request des Browsers übermittelt werden. Eine dieser Informationen ist zum Beispiel die Version und der Hersteller des Browsers, den der Kunde benutzt. Personalisierung ist aber schwierig, wenn man nicht genau weiß, was der Besucher möchte, und der Besucher aufgrund der Masse an Auswahlmöglichkeiten
2.2. NUTZEN DER PERSONALISIERUNG


2.2 Nutzen der Personalisierung


2.2.1 Nutzen für den Anwender


---

4siehe [Hagen et al. 98, Seiten 10,11]
5Eine Internetseite zum Thema Personalisierung, die von dem Softwareanbieter Netperceptions finanziert wird: http://www.personalization.com
6Als Maß für die Benutzerfreundlichkeit wird häufig die Zeit genommen, die man für die Erledigung einer Aufgabe braucht. http://www.uv-secure-it.com/pages/20010121.html

### 2.2.2 Nutzen für den Anbieter


### 2.3 Datenschutz

Wie bereits aus den bisherigen Abschnitten hervorgegangen ist, benötigt Personalisierung eine große Menge von personenbezogenen Daten. Inwiefern es der Nutzen für beide Seiten rechtfertigt, eine Datenbank anzulegen, wird im Rahmen dieser Arbeit nicht untersucht oder bewertet. Dennoch soll darauf aufmerksam gemacht werden, dass der Datenschutz nicht vernachlässigt werden darf. Allein schon aus dem Interesse des Web-Site-Betreibers,

\(^7\)siehe [Hagen et al. 98, Seite 9]
2.4 Formen von Personalisierung

Nutzer analysiert wird. Dabei versteht man unter dem Klickpfad den Navigationsweg, den ein Besucher auf einer Web-Site zurückgelegt hat, also die Sequenz der abgerufenen Seiten.
Kapitel 3

Der Weg zum Wissen

3.1 Web Mining


3.1.1 Web Content Mining


3.1.2 Web Structure Mining


3.1.3 Web Usage Mining

Web Usage Mining befasst sich hauptsächlich mit den Nutzern, die die verschiedenen Webseiten abrufen. Untersucht werden Fragestellungen wie „Welche Bereiche der Website werden stark frequentiert?“. Mit Hilfe des Web Usage Mining kann der Betreiber also erkennen, welche Rubriken seiner Homepage ausgebaut oder eingestellt werden müssen. Aber nicht nur die Frequenzierung der einzelnen Seiten ist von Belang, sondern auch wie die Nutzer dorthin gelangen. Anhand der häufigen Bewegungspfade kann der Betreiber erkennen, wie die Struktur der Website verbessert werden kann. So sollten häufig frequentierte Bereiche der Website einfach zu erreichen sein. Die Aufgaben und Ziele des Web Usage Mining sind
vergleichbar mit denen eines Supermarktleiters. Der Supermarktleiter untersucht ebenfalls die Wege, die die Kunden in seinem Markt nehmen. Er macht sich Gedanken darüber, ob die verschiedenen Produkte, die er anbietet, gefunden werden, ob Zweitplatzierungen einen Effekt haben und die Kunden sich möglichst lange in seinem Geschäft aufhalten. Ein weiteres wichtiges Aufgabengebiet des Web Usage Mining ist die Untersuchung der Nutzer der Web-Site. Je nach Nutzergruppe kann zum Beispiel der Content angepasst oder eine zielgruppenorientierte Werbung gezeigt werde.

3.2 Knowledge Discovery in Databases


3.2.1 Definition von Knowledge Discovery in Databases


3.2.2 Prozessbeschreibung

3.2. KNOWLEDGE DISCOVERY IN DATABASES

Datenauswahl

Bevor aus den Rohdaten die Zieldaten ausgewählt werden können, muss man sich über das Ziel dieses KDD-Prozesses im klaren sein. Um ein vernünftiges Ziel überhaupt definieren zu können, ist es notwendig, genügend Verständnis aus dem Anwendungsbereich, der dortigen Abläufe und dem bisherigen Wissen zu haben. Die eigentlichen Zieldaten, aus denen im weiteren das Wissen extrahiert werden soll, sind im allgemeinen eine Untermenge der zur Verfügung stehenden Daten.

Vorbearbeitung


Transformation

Um die Datenmenge für die laufzeitintensiven Data Mining-Algorithmen zu verkleinern, werden die Daten in ihrer Dimension reduziert. So bringen redundante Informationen für spätere Algorithmen keinen Vorteil. Auch Daten, die nur wenig zusätzliche Informationen enthalten, werden häufig fallengelassen. In der Mustererkennung nennt man diese Teilaufgabe Featureselection.

Data Mining


Interpretation und Auswertung

Die letzten beiden Schritte sind Auswertung und Interpretation des Ergebnisses sowie eine kritische Betrachtung und Überprüfung. Die aus dem Data Mining-Schritt entstandenen Daten werden entweder für einen Entscheidungsträger aufbereitet oder durch eine weitere

3.3 Ziele des KKD-Prozesses

3.4 Interdisziplinarität


Abbildung 3.3: KDD als Interdisziplinäres Forschungsgebiet
KAPITEL 3. DER WEG ZUM WISSEN
Kapitel 4

Data Mining-Algorithmen


4.1 Klassifizierende Algorithmen


Da während der Projektarbeit keine Klassifikatoren implementiert wurden, wird im wei-
teren auf die einfachen Nearest Neighbour- und K-Nearest Neighbour- Klassifikatoren eingegangen. Informationen zu den Bayes’schen Entscheidungsregeln und den darauf basierenden Klassifikatoren findet man in [Schürmann 96].

4.1.1 Nearest Neighbour


4.1.2 K-Nearest Neighbour

Der K-Nearest Neighbour-Klassifikator ist vom Prinzip her ähnlich aufgebaut wie der Nearest Neighbour-Klassifikator. Wieder wird die gesamte Trainingsmenge als Referenzmenge benutzt. In Abbildung 4.2 sieht man die Referenzpunkte von drei verschiedenen Klassen. Im Gegensatz zum Nearest Neighbour-Klassifikator ist jedoch nicht allein der nächste Nachbar ausschlaggebend dafür, zu welcher Klasse das Element zugeordnet wird, sondern die Mehrheit der k nächsten Nachbarn entscheidet. In dem zweidimensionalen Beispiel in Abbildung 4.2 ist k = 20 gewählt. Durch die Anzahl der Elemente aus den jeweiligen Klassen
4.2 Segmentierende Algorithmen

Häufig hat man Anwendungen, bei denen die verschiedenen Klassen noch nicht bekannt sind, oder keine Trainingsmenge zur Verfügung steht. Somit kann man also nicht eine klassifizierende Methode verwenden, um die Elemente den einzelnen Klassen zuzuordnen. Stattdessen versucht man die Menge in verschiedene Cluster einzuteilen. Dabei sollen die erzeugten Cluster folgende Eigenschaften möglichst gut erfüllen: Die Abstände der Elemente innerhalb eines Clusters sollen möglichst klein sein und die Abstände der Elemente zwischen zwei unterschiedlichen Clustern sollen möglichst groß sein. Die Metrik, die da-

Abbildung 4.1: Nearest Neighbour

KAPITEL 4. DATA MINING-ALGORITHMEN

Abbildung 4.2: K-Nearest Neighbour

Abbildung 4.3: Vector Quantization
bei verwendet wird, ist von der Anwendung abhängig und muss jeweils definiert werden. Insgesamt gibt es eine Vielzahl verschiedener Verfahren, um eine Menge von Objekten zu segmentieren. Um die prinzipielle Funktionsweise der Segmentierung zu verdeutlichen, wird im folgenden ein hierarchisches Verfahren zur Segmentierung vorgestellt.

4.2.1 Vector Quantization

Bei diesem Verfahren wird die Trainingsmenge in jedem Iterationsschritt in immer kleinere Cluster eingeteilt. Es handelt sich somit um ein divisives hierarchisches Verfahren. Dabei wird jeder Cluster durch einen Zentroid und einen Saatpunkt beschrieben. Der Saatpunkt wird dafür benutzt, die einzelnen Elemente der Trainingsmenge den jeweiligen Clustern zuzuordnen. Dabei wird jeweils der Cluster mit dem nächsten Saatpunkt gewählt. Sie haben somit die Rolle der Referenzpunkte des bereits vorgestellten Nearest Neighbour-Klassifikator. Die Zentroide sind die Mittelwerte der einzelnen Cluster. Das Verfahren funktioniert nach folgendem Algorithmus:

1. Falls aus der Anwendung heraus sinnvolle Startwerte für die Saatpunkte vorhanden sind, setze diese. Andernfalls wählte einen Saatpunkt zufällig aus.
5. Falls in dem vorherigen Schritt keine neuen Cluster erzeugt wurden, beende die Iteration, ansonsten mache weiter bei Schritt 2.

4.3 Abhängigkeitsentdeckende Algorithmen

4.3.1 Assoziationsregeln


- Es sei $I = \{i_1, \ldots, i_m\}$ eine Menge uninterpretierter, diskreter Größen, sogenannter Items. $t \subseteq I$ heißt Transaktion.

- Für eine Datenbasis $D = \{t_1, \ldots, t_N\}, t_i \subseteq I$ von Transaktionen und beliebige Itemmenge $X, Y \subset I$ heißt

  $$s(X) = \frac{|\{t \in D : X \subseteq t\}|}{|D|}$$

  Support von $X$ und

  $$c(X, Y) = \frac{|\{t \in D : (X \cup Y) \subseteq t\}|}{|\{t \in D : X \subseteq t\}|} = \frac{s(X \cup Y)}{s(X)}$$

  Konfidenz von $X$ und $Y$.

- Alle Itemmengen $X \subset I$, für die $s(X) \geq s_{\text{min}}$ für eine vorgegebene Supportschranke $s_{\text{min}}$, gilt, nennt man häufige Itemmenge.

- Das Paar $(X, Y)$ zusammen mit $s(X \cup Y)$ und $c(X, Y)$ heißt Assoziationsregel $X \rightarrow Y$ mit Support $s(X \cup Y)$ und Konfidenz $c(X, Y)$.

- Für eine Assoziationsregel $X \rightarrow Y$ heißen $X$ Regelrumpf und $Y$ Regelkopf.

Der Algorithmus hat die Aufgabe, alle Assoziationsregeln zu finden, bei denen der Support $s(X \cup Y) \geq s_{\text{min}}$ und die Konfidenz $c(X, Y) \geq c_{\text{min}}$ ist. Im allgemeinen suchen die Algorithmen zuerst häufige Itemmengen in der Datenbasis und konstruieren in einem zweiten Schritt daraus alle Assoziationsregeln mit einem Konfidenzwert größer oder gleich
der vorgegebenen unteren Schranke. Da die Zahl möglicher häufiger Itemmengen $2^m - 1$ schon bei kleinen Itemmengen $I$ recht groß wird, machen sich die Algorithmen folgende Eigenschaften zu Nutze, um die Zahl zu untersuchender Itemmengen zu reduzieren:

- Der Support einer Itemmenge $Y$ ist immer kleiner oder gleich dem Support ihrer Teilmengen.
  
  Seien $X, Y \subset I$ beliebige Teilmengen mit $X \subseteq Y$. Dann gilt:
  
  $$s(Y) \leq s(X)$$

- Der Support einer Itemmenge, die eine nicht häufige Teilmenge enthält, ist auch nicht häufig.

  Sei $X \subset I$ eine beliebige Itemmenge mit $s(X) < s_{\text{min}}$. Dann gilt:
  
  $$s(Y) < s_{\text{min}} \text{ für alle } Y \subset I, Y \supseteq X$$

- Jede Teilmenge einer häufigen Teilmenge ist auch häufig.

  Sei $X \subset I$ eine beliebige Itemmenge mit $s(X) \geq s_{\text{min}}$. Dann gilt:
  
  $$s(Y) \geq s_{\text{min}} \text{ für alle } Y \subset I, Y \subseteq X$$

Näheres zu einzelnen Assoziationsregeln konstruierenden Algorithmen stehen in [B. Mobasher et al. 97] (AIS-Algorithmus) und in [Agrawal et al. 93] (a priori Algorithmus).
Kapitel 5

Recommendation Systems

5.1 Aufgabe


Abbildung 5.1: Recommendation als individuelle Sortierung von Elementen
5.2 Grundannahmen - verwendete Heuristiken


5.3 Präferenzen

5.3. PRÄFERENZEN


5.3.1 Probleme in der Datendarstellung


\[1\]http://www.imdb.com (Internet Movie Database)
5.3. PRÄFERENZEN


5.3.2 Neighbourhood formation


$$\text{corr}_{ab} = \frac{\sum_i (r_{ai} - \tau_a)(r_{bi} - \tau_b)}{\sqrt{\sum_i (r_{ai} - \tau_a)^2 \sum_i (r_{bi} - \tau_b)^2}}$$

als auch der Cosinus

$$\cos(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\|_2 \cdot \|\vec{b}\|_2}$$


5.3.3 Generierung der Empfehlung

Kapitel 6

Design und Implemtierung

6.1 Aufgabe


6.1.1 Der infoAsset Broker


\footnote{infoAsset AG, Harburger Schloßstraße 6-12, 21079 Hamburg}
6.1.2 Das Recommendation System

KAPITEL 6. DESIGN UND IMPLEMENTIERUNG


6.2 Module des Recommendation System

6.2.1 Hauptanwendungsfälle

Während des eigentlichen Recommendation Betriebes lassen sich verschiedene Anwendungsfälle identifizieren. Menschliche Akteure in diesen Anwendungsfällen sind die Endbenutzer und der Administrator. Außerdem sind die Basis-Module des infoAsset Brokers als Akteure zu sehen, da sie bei verschiedenen Ereignissen auf das Recommendation System wirken. Die verschiedenen Anwendungsfälle sind in Abbildung 6.3, einem Use-Case-
6.2. MODULE DES RECOMMENDATION SYSTEM

Abbildung 6.3: Die Use-Cases des Recommendation System

Diagramm, zu sehen. Use-Case-Diagramme sind teil der UML\textsuperscript{2}-Notation, die vielfältige graphische Notationsmöglichkeiten bietet, um verschiedene Aspekte einer Software zu beschreiben. Im weiteren Verlauf der Projektarbeit sind andere Diagrammtypen der UML zu sehen. Informationen zu den Notationen der UML findet man in [Martin Fowler 99].

**Ausgabe eines Empfehlungswertes**


**Ausgabe einer sortierten Liste**

Häufig hat der Anwender mit Listen von Dokumenten zu tun. Diese Listen treten zum Beispiel als Ergebnis von Suchen auf. Der Anwender hat mit Hilfe des \textit{Recommendation}
System die Möglichkeit, die Trefferliste seiner Suche nach den für ihn berechneten Empfehlungen zu sortieren. Dabei werden die Werte der von ihm ausgewählten Bewertungsregel genutzt.

**Auszgabe einer Liste interessanter Dokumente**


**Eingabe von Bewertungen**


**Auswahl einer Bewertungsregel**


**Administration der Bewertungsregeln**


**6.2.2 Aktualisierung der Nutzernachbarschaft**

Im normalen Betrieb des infoAsset Brokers werden immer wieder Personen einer Gruppe zugeordnet oder aus ihr entfernt. Da es eine Recommendation Engine gibt, die die
Nachbarschaft über die Guppenmitgliedschaft definiert, ist es notwendig, dass die Recommendation Engine von den anderen Services des infoAsset Brokers informiert wird, falls sich Mitgliedschaften ändern.

### Einordnung neuer Nutzer


### 6.2.3 Pakete des Recommendation System


---

3 Um die Lesbarkeit und die Übersichtlichkeit des Textes zu erhöhen werden bei den folgen Package Angaben die Superpackages de.infoasset.broker durch * abgekürzt. Die vollständige Schreibweise wäre in diesem Fall also `de.interfaces.broker.services.classification.amf`
6.3 Identifizierung von Klassen


6.3.1 Asset

Innerhalb des infoAsset Brokers werden alle Klassen, die für den Endbenutzer Informationsgehalt haben, als Asset bezeichnet. Dazu zählen auch Klassen, die keine eigentlichen Inhalte (Content) repräsentieren, wie die Klassen Person, Group, Membership sowie die

Abbildung 6.4: Die neuen Pakete des infoAsset Brokers
6.3. IDENTIFIZIERUNG VON KLASSEN


6.3.2 **AssetContainer**

6.3.3 Person (\textit{Person})

Die Klasse 	extit{Person} stellt eine natürliche Person dar. Diese Klasse ist nicht neu zu dem infoAsset Broker hinzugekommen, sondern existiert seit der ersten Version. Da es sich bei einem 	extit{Recommendation System} um einen personalisierten Service handelt, ist die Klasse hier mit aufgeführt.

6.3.4 Gruppe (\textit{Group})


6.3.5 Mitgliedschaft (\textit{Membership})


6.3.6 Kategorie (\textit{Category})

Der „Kategorie“-Begriff steht in einem starken Zusammenhang mit der Klassifikation und der Segmentierung. Innerhalb der Segmentierung bezeichnet man die einzelnen erzeugten
6.3. IDENTIFIZIERUNG VON KLASSEN


6.3.7 Kategorisierung (Categorization)


6.3.8 Klassifizierung (Classification)

wie die Membership Klasse eine Beziehungsklasse und wird daher durch eine verborgene Relationship realisiert.

6.3.9 Klassifikator (Classifier)


6.3.10 Bewertung (Rating)


6.3.11 Beziehung (Relationship)

Die Klasse Relationship stellt generelle Beziehungen zwischen zwei Assets dar. Die Attribute der allgemeinen Beziehungsklasse sind: 1. der Typ der Beziehung, der relativ grob die Art der Beziehung zwischen den beiden Assets beschreibt, 2. eine Rolle, die die Möglichkeit zu einer näheren Beschreibung bietet5, und 3. einem Zahlenwert, der die Stärke der Beziehung

5 vgl. mit Membership, Abschnitt 6.3.5

6.3.12 Recommendation Engine


---

6 vgl. mit Classification und Rating, Abschnitt 6.3.8 und 6.3.10

\[ \text{recom}_{PD}(p_x, d_y) = \frac{\sum_{i} \text{int}_{PP}(p_x, p_i) \cdot \text{rat}_{PD}(p_i, d_y)}{\sum_{i} \left|\text{int}_{PP}(p_x, p_i)\right|} \text{ mit } i \in I(p_x, d_y) \]

Dabei ist I die Indexmenge aller von \( p_x \) bewerteten Personen \( p_i \), die selber das Dokument \( d_y \) bewertet haben. Die Funktion \( \text{int}_{PP}(p_x, p_i) \) gibt die „Interessenähnlichkeit“ der Person \( p_x \) zu der Person \( p_i \) an. Die Bewertung eines Dokuments wird durch die Funktion \( \text{rat}_{PD}(p_i, d_y) \) ausgedrückt. Die Variable \( p \) steht also immer für eine Person und \( d \) entsprechend für ein Dokument. Die Indizes werden benötigt um die einzelnen Personen und Dokumente von einander unterscheiden zu können. Die zweite Art von Recommendation Engines fasst die Ergebnisse anderer Engines zusammen. In Abbildung 6.6 ist eine implementierungsnahe Klassendiagramm der Recommendation Engines zu sehen.

### 6.4 Beziehungen der Klassen untereinander

In dem vorherigen Abschnitt wurden die Haupklassen des Recommendation Systems und der angrenzenden Bereiche vorgestellt. In diesem Abschnitt werden die Beziehungen der Klassen untereinander hervorgehoben. Der infoAsset Broker hat eine zentrale Klasse (Services), bei der die verschiedenen Dienste des Brokers angefragt werden können. Bei den
6.4. BEZIEHUNGEN DER KLASSEN UNTEREINANDER

Abbildung 6.6: Klassendiagramm der Recommendation Engines

Die Abbildung 6.9 zeigt die Implementierung von Classification, Categorization und Category im Rahmen der Personen- und Gruppen-Klasse. Es ist jedoch analog mit den Klas-
6.5 Ereignisbehandlung im Recommendation System


Abbildung 6.8: Konzeptuelles Klassendiagramm: Zuordnung von Assets zu Kategorien und Kategorisierungen.
Abbildung 6.10: Nachrichtensequenz beim Eventmechanismus
zugehörigen AssetContainer, der dann die Nachricht an die angemeldeten Listener weiterleitet. Diese Veränderung ist notwendig, um nicht jedem einzelnen Asset einen Listener hinzufügen zu müssen. Dies wäre besonders Aufwändig, da die Benachrichtigung auch bei Assets möglich sein soll, die noch nicht von dem AssetContainer initialisiert wurden.
Kapitel 7
Schluss


\(^1\) Asset Mining Framework
## Anhang A

### Package de.infoasset.broker.interfaces

<table>
<thead>
<tr>
<th>Package Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces</td>
<td></td>
</tr>
<tr>
<td>Membership</td>
<td>54</td>
</tr>
<tr>
<td>A membership describes the role of a Person in a Group.</td>
<td></td>
</tr>
<tr>
<td>MembershipEvent</td>
<td>56</td>
</tr>
<tr>
<td>This interface describes a membershipevent.</td>
<td></td>
</tr>
<tr>
<td>Memberships</td>
<td>57</td>
</tr>
<tr>
<td>A persistent collection of Membership objects (for all groups and all persons).</td>
<td></td>
</tr>
<tr>
<td>MembershipsListener</td>
<td>60</td>
</tr>
<tr>
<td>The interface is implemented by IMPMembershipsListener to use as a superclass for an inner anonymous class.</td>
<td></td>
</tr>
<tr>
<td>Relationship</td>
<td>60</td>
</tr>
<tr>
<td>A relationship describes general relations between 2 different assets.</td>
<td></td>
</tr>
<tr>
<td>RelationshipEvent</td>
<td>66</td>
</tr>
<tr>
<td>This interface describes a RelationshipEvent.</td>
<td></td>
</tr>
<tr>
<td>Relationships</td>
<td>69</td>
</tr>
<tr>
<td>The interface Relationships describes the relation between asset a and asset b.</td>
<td></td>
</tr>
<tr>
<td>RelationshipsListener</td>
<td>79</td>
</tr>
<tr>
<td>The interface is implemented by IMPRelationshipsListener to use as a superclass for an inner anonymous class.</td>
<td></td>
</tr>
</tbody>
</table>
A.1 Interfaces

A.1.1 INTERFACE Membership

A membership describes the role of a Person in a Group.

Declaration

```java
public interface Membership
    implements Asset
```

Methods

- **getAuthorId**
  ```java
  public String getAuthorId()
  ```
  - **Usage**
    * Return the id of this membership’s author.
  - **Returns** - the id of the author

- **getAuthorKind**
  ```java
  public String getAuthorKind()
  ```
  - **Usage**
    * Return the kind of this membership’s author.
  - **Returns** - the kind of the author

- **getComment**
  ```java
  public String getComment()
  ```
  - **Usage**
    * Return the comment describing the role of the person (initially null).
  - **Returns** - the comment

- **getGroupId**
  ```java
  public String getGroupId()
  ```
  - **Usage**
    * Return the id of the group.
  - **Returns** - the id of the group
• `getPersonId`
  public String getPersonId( )

  - **Usage**
    * Return the id of the person.
  - **Returns** - the id of the person

• `getRoleId`
  public String getRoleId( )

  - **Usage**
    * Return the id of a domain value (in the domain "GroupRole") classifying the role of the person in the group.
  - **Returns** - the id of the Role

• `setAuthorId`
  public void setAuthorId( java.lang.String id )

  - **Usage**
    * Update the id of this membership’s author.
  - **Parameters**
    * id - the new id of the author

• `setAuthorKind`
  public void setAuthorKind( java.lang.String kind )

  - **Usage**
    * Update the kind of this membership’s author.
  - **Parameters**
    * kind - the new kind of the author

• `setComment`
  public void setComment( java.lang.String comment )

  - **Usage**
    * Update the comment describing the role of the person in the group.
  - **Parameters**
    * comment - the new comment

• `setGroupId`
  public void setGroupId( java.lang.String id )
A.1.2 INTERFACE MembershipEvent

This interface describes a membership event. The event is fired when a membership is added or removed from the memberships asset container or the relationship is changed. You can get the former values of the changed attributes via the get methods. The get Methods of non changed attributes return null. Additionally needed method calls, fired on certain events, should be added on demand.

DECLARATION

```java
public interface MembershipEvent
    implements AssetEvent
```

METHODS

- **getOldPersonId**
  ```java
  public String getOldPersonId()
  ```
Usage
* Returns the former id of the person being part of this membership

- Returns - the former id or null if the id of the person is not changed.

A.1.3 INTERFACE Memberships

A persistent collection of Membership objects (for all groups and all persons).

Declaration

```
public interface Memberships
  implements AssetContainer
```

Methods

- `addMembershipsListener`

  ```
  public void addMembershipsListener(
      de.infoasset.broker.interfaces.MembershipsListener msl )
  ```

  - Usage
    * Attaches a membershipslistener to this memberships assetcontainer. Methods of this listener are invoked by the memberships implementation on certain events. The Observer - Pattern, described by Gamma et. al. in "Design Patterns" is an equivalent pattern to this listener.

  - Parameters
    * msl - the additional MembershipListener

- `countMembershipsOfGroup`

  ```
  public int countMembershipsOfGroup( java.lang.String groupId )
  ```

  - Usage
    * Return the number of memberships assigned with this group. If any person has two different roles in a group the number of Persons in group differ from the number of Memberships.

  - Parameters
    * groupId - the id of the group

  - Returns - the number of Memberships
• **createMembership**
  ```java
  public Membership createMembership()
  ```

  - **Usage**
    * Create a new membership. Person, Group and Role still need to be defined.
  - **Returns** - Description of the Returned Value

• **getAllMembershipsOfPerson**
  ```java
  public Iterator getAllMembershipsOfPerson(java.lang.String personId)
  ```

  - **Usage**
    * Return an iterator over all membership objects for this person.
  - **Parameters**
    * `personId` - Description of Parameter
  - **Returns** - iterator over memberships

• **getMembership**
  ```java
  public Membership getMembership(java.lang.String id)
  ```

  - **Usage**
    * Return the membership with this id or `null` if no such membership exists.
  - **Parameters**
    * `id` - Description of Parameter
  - **Returns** - the membership with the id or `null` if no such membership exists

• **getMembershipsOfGroup**
  ```java
  public Iterator getMembershipsOfGroup(java.lang.String groupId)
  ```

  - **Usage**
    * Return an iterator over all membership objects for this group.
  - **Parameters**
    * `groupId` - Description of Parameter
  - **Returns** - iterator over memberships

• **getMembershipsOfPerson**
  ```java
  public Iterator getMembershipsOfPerson(java.lang.String personId)
  ```

  - **Usage**
* Return an iterator over all normalmembership objects for this person. Normal means that memberships of groups created by a classifier are not mentioned.

- **Parameters**
  * personId - Description of Parameter
- **Returns** - iterator over memberships

*getMembershipsOfPersonWithAuthor*

```java
public Iterator getMembershipsOfPersonWithAuthor(
    java.lang.String personId, java.lang.String authorKind,
    java.lang.String authorId)
```

- **Usage**
  * Return an iterator over all membership objects for this person with the given author.

- **Parameters**
  * personId - the id of a person
  * authorKind - the kind of the author
  * authorId - the id of the author
- **Returns** - iterator over memberships

*ids*

```java
public Iterator ids()
```

- **Usage**
  * Return an iterator over the ids of all memberships
- **Returns** - iterator over the ids of all memberships

*iterator*

```java
public Iterator iterator()
```

- **Usage**
  * Return an iterator over all membership objects for all groups and all persons.
- **Returns** - iterator over all memberships

*remove*

```java
public boolean remove(java.lang.String id)
```

- **Usage**
  * Remove the membership with this id. Return true if the operation succeeded, false otherwise.

- **Parameters**
• id - the id of the membership to remove
  - **Returns** - true, if the operation succeeded

• **removeMembershipsListener**
  public void removeMembershipsListener(
    de.infoasset.broker.interfaces.MembershipsListener msl )

  - **Usage**
    * Detaches a listener from this memberships assetcontainer.
  - **Parameters**
    * msl - the membershiplistener to remove

### A.1.4 **INTERFACE MembershipsListener**

The interface is implemented by IMPMembershipsListener to use as a superclass for an inner anonymous class.

**Declaration**

public interface MembershipsListener
implements AssetContainerListener

**Methods**

• **personIdChanged**
  public void personIdChanged(
    de.infoasset.broker.interfaces.MembershipEvent e )

  - **Usage**
    * The method is invoked whenever the id of the person is changed and persistency is granted.
  - **Parameters**
    * e - a membershipevent, provides informations about the event occured

### A.1.5 **INTERFACE Relationship**

A relationship describes general relations between 2 different assets. The participating assets in this relationship are called A and B (a- and b-part of a relationship). Furthermore the
relationship gives information about the type of relationship. Currently there are several relationship types in use e.g. person_rat_person" (a person rates another person), person_rat_document" (a person rates a document), person_mbr_group" (a person is a member of a group) ... But often the relationship type doesn’t describe the relationship sufficient. Therefore the relationship gives role information of one asset in the relationship, e.g. a member of a group might be a group leader or a research assistant Additionally the relationship consists of some other attributes. The owner of a relationship identifies the creator of the relationship. The weight of a relationship might be used for the confidence value in a classification or the rating value in a rating relationship. The state of a relationship is either suggested or confirmed in a classification relationship. But may be used in another relationship context in a different way.

**Declaration**

```java
public interface Relationship
    implements Asset
```

**Methods**

- `getAId`
  ```java
  public String getAId( )
  ```
  
  - **Usage**
  * Returns the id of the asset taking the A - part of a relationship
  - **Returns** - the asset id of part a

- `getAKind`
  ```java
  public String getAKind( )
  ```
  
  - **Usage**
  * Returns the kind of the asset taking the A - part of a relationship
  - **Returns** - the asset kind of part a

- `getAuthorId`
  ```java
  public String getAuthorId( )
  ```
  
  - **Usage**
  * Returns the id of the author of this relationship. The author might be a person, a classifier or any other asset
  - **Returns** - the id of the author
• getAuthorKind
  public String getAuthorKind( )
  – Usage
    * Returns the kind of the author of this relationship.
  – Returns - the kind of the author

• getBId
  public String getBId( )
  – Usage
    * Returns the id of the asset taking the B - part of a relationship
  – Returns - the asset id of part b

• getBKind
  public String getBKind( )
  – Usage
    * Returns the kind of the asset taking the B - part of a relationship
  – Returns - the asset kind of part b

• getComment
  public String getComment( )
  – Usage
    * Returns a comment for this relationship.
  – Returns - the comment

• getCreationDate
  public Date getCreationDate( )
  – Usage
    * Returns the creationdate of this relationship. During the hole lifetime of a relationship the creation date of relationsjip should never change.
  – Returns - the date of creation

• getRelationshipType
  public String getRelationshipType( )
  – Usage
    * Returns the relationship type of a relationship. E. g. person_mbr_group”.
  – Returns - the type of the relationship
• **getRoleId**
  public String getRoleId( )
  
  - **Usage**
    * Returns the role of an asset in this relationship. E.g. a person might be a group leader in a membership relationship. The returned value is a domain value.
  
  - **Returns** - The RoleId value

• **getState**
  public String getState( )
  
  - **Usage**
    * Returns the current state of the relationship. The state changes during the lifetime of the relationship. The returned value is a domain value.
  
  - **Returns** - The State value

• **getWeight**
  public double getWeight( )
  
  - **Usage**
    * Returns the weight of a relationship. In each specific relationship the weight is interpreted in a different way.
  
  - **Returns** - the weight

• **setAId**
  public void setAId( java.lang.String id )
  
  - **Usage**
    * Updates the id of the asset taking the A - part of a relationship
  
  - **Parameters**
    * id - the new id of asset a

• **setAKind**
  public void setAKind( java.lang.String kind )
  
  - **Usage**
    * Updates the kind of the asset taking the A - part of a relationship
  
  - **Parameters**
    * kind - the new kind of asset a

• **setAuthorId**
  public void setAuthorId( java.lang.String id )

• **setBId**
  public void setBId( java.lang.String id )
  
  - **Usage**
    * Updates the id of the asset taking the B - part of a relationship
  
  - **Parameters**
    * id - the new id of asset b

• **setBKind**
  public void setBKind( java.lang.String kind )
  
  - **Usage**
    * Updates the kind of the asset taking the B - part of a relationship
  
  - **Parameters**
    * kind - the new kind of asset b

• **setComment**
  public void setComment( java.lang.String comment )
  
  - **Usage**
    * Stores a comment about the relationship
  
  - **Parameters**
    * comment - the comment of the relationship

• **setDate**
  public void setDate( java.util.Date date )
  
  - **Usage**
    * Stores the date of the relationship
  
  - **Parameters**
    * date - the date of the relationship
- **Usage**
  * Updates the id of the author of this relationship

- **Parameters**
  * id - the new id

---

- **setAuthorKind**
  `public void setAuthorKind( java.lang.String kind )`

  - **Usage**
    * Updates the kind of the author of this relationship.
  - **Parameters**
    * kind - the new kind of the author

---

- **setBId**
  `public void setBId( java.lang.String id )`

  - **Usage**
    * Updates the id of the asset taking the B - part of a relationship
  - **Parameters**
    * id - the new id of asset b

---

- **setBKind**
  `public void setBKind( java.lang.String kind )`

  - **Usage**
    * Updates the kind of the asset taking the B - part of a relationship
  - **Parameters**
    * kind - the new kind of asset b

---

- **setComment**
  `public void setComment( java.lang.String comment )`

  - **Usage**
    * Updates the comment for this relationship
  - **Parameters**
    * comment - the new comment

---

- **setCreationDate**
  `public void setCreationDate( java.util.Date date )`

  - **Usage**
* Updates the creation date of a relationship. The creation date should never be changed during the entire lifetime of a relationship. Use the Asset.getLastModificationDate() method instead.

- **Parameters**
  - **date** - the new creation date

- **setRelationshipType**
  ```java
  public void setRelationshipType( java.lang.String id )
  ```

  - **Usage**
    - Updates the type of a relationship.

  - **Parameters**
    - **id** - the new type

- **setRoleId**
  ```java
  public void setRoleId( java.lang.String id )
  ```

  - **Usage**
    - Updates the role of an asset in a relationship.

  - **Parameters**
    - **id** - the new role (a domain value)

- **setState**
  ```java
  public void setState( java.lang.String state )
  ```

  - **Usage**
    - Updates the state of the relationship.

  - **Parameters**
    - **state** - the new state

- **setWeight**
  ```java
  public void setWeight( double weight )
  ```

  - **Usage**
    - Updates the weight of a relationship. The implementing class might use a different number format than double.

  - **Parameters**
    - **weight** - the new weight
A.1.6 **INTERFACE RelationshipEvent**

This interface describes a RelationshipEvent. The event is fired when a relationship is added or removed from the relationships assetcontainer or the relationship is changed. You can get the old values of the changed attributes via the get methods. The get Methods of non changed attributes return null values.

**DECLARATION**

```java
public interface RelationshipEvent
 implements AssetEvent
```

**METHODS**

- **getOldAId**
  ```java
  public String getOldAId()
  ```
  - **Usage**
    * Returns the former id of the asset assigned to the A part of the relationship.
  - **Returns** - the former id or null if the id of the asset is not changed

- **getOldAuthorId**
  ```java
  public String getOldAuthorId()
  ```
  - **Usage**
    * Returns the former id of the asset assigned to the relationship as the author.
  - **Returns** - the former id or null if the id of the asset is not changed

- **getOldAuthorKind**
  ```java
  public String getOldAuthorKind()
  ```
  - **Usage**
    * Returns the former kind of the asset assigned to the relationship as the author.
  - **Returns** - the former kind or null if the id of the asset is not changed

- **getOldBId**
  ```java
  public String getOldBId()
  ```
- **Usage**
  * Returns the former id of the asset assigned to the B part of the relationship.
- **Returns** - the former id or null if the id of the asset is not changed

- **getOldRoleId**
  ```java
  public String getOldRoleId() {
  }
  ```
  - **Usage**
    * Returns the role of the asset assigned to the A or B part of the relationship in this relationship. The role is the id of a `DomainValue`.
  - **Returns** - the former role or null if the role in this relationship is not changed.

- **getOldState**
  ```java
  public String getOldState() {
  }
  ```
  - **Usage**
    * Returns the former state of the relationship. The state is a domainvalue.
  - **Returns** - the former state of the relationship or null if the state is not changed

- **getOldWeight**
  ```java
  public double getOldWeight() {
  }
  ```
  - **Usage**
    * Returns the former weight of the relationship.
  - **Returns** - the former weight or null if the weight of the relationship is not changed

- **getRelationship**
  ```java
  public Relationship getRelationship() {
  }
  ```
  - **Usage**
    * Returns the relationship, that is the origin of this event.
  - **Returns** - the relationship, that causes this event

- **setOldAId**
  ```java
  public void setOldAId(java.lang.String oldAId) {
  }
  ```
  - **Usage**
    * Sets the former id of the asset assigned to the A part of the relationship.
  - **Parameters**
oldAId - the former id of the asset

- setOldAuthorId
  public void setOldAuthorId( java.lang.String oldAuthorId )
  
  - Usage
    * Sets the former id of the asset assigned to the relationship as the author.
  
  - Parameters
    * oldAuthorId - the former id of the asset

- setOldAuthorKind
  public void setOldAuthorKind( java.lang.String oldAuthorKind )
  
  - Usage
    * Sets the former kind of the asset assigned to the relationship as the author.
  
  - Parameters
    * oldAuthorKind - the former kind of the asset

- setOldBId
  public void setOldBId( java.lang.String oldBId )
  
  - Usage
    * Sets the former id of the asset assigned to the B part of the relationship.
  
  - Parameters
    * oldBId - the former id of the asset

- setOldRoleId
  public void setOldRoleId( java.lang.String oldRoleId )
  
  - Usage
    * Sets the former role of the A or B part of the relationship.
  
  - Parameters
    * oldRoleId - the former role

- setOldState
  public void setOldState( java.lang.String oldState )
  
  - Usage
    * Sets the former state of the relationship
  
  - Parameters
    * oldState - the former state
• `setOldValues`  
  ```java
  public void setOldValues( de.infoasset.broker.interfaces.Relationship oldRelationship )
  ```  
  **Usage**  
  * This method saves the values of an relationship in this event message.  
  **Parameters**  
  * `oldRelationship` - the relationship, that attribute values to transfer in this message.

• `setOldWeight`  
  ```java
  public void setOldWeight( double oldWeight )
  ```  
  **Usage**  
  * Sets the former weight of the relationship  
  **Parameters**  
  * `oldWeight` - the former weight

### A.1.7 INTERFACE `Relationships`

The interface `Relationships` describes the relation between asset a and asset b. The assets are identified by their kind and their id. Each relationship has a relationshipType which identifies the kind of the relation (e.g. Classification Concept-Document). There are other attributes to identify the author, the creation date and the weight of the relationship.

Example: (Pseudocode)  
```java
aRelationship.aId = 123;

aRelationship.aKind=Person"

aRelationship.bId= 33;

aRelationship.bKind= "Group"

aRelationship.type= "g1"; // the id of a distinct groupcategory

aRelationship.role= anotherDomainvalue; // representing "Guest"

aRelationship.weight= something;
```
The pseudocode has the meaning: The person with the id 123 is a guest of the group with the id 33. The weight has no meaning in this relationship. There should be conventions about what happens if the relationship is a part-of relation proposal: - AssetA consists of AssetBs - AssetA is always the superAsset, AssetB always the subAsset;

**Declaration**

```java
public interface Relationships
    implements AssetContainer
```

**Methods**

- addRelationshipsListener
  ```java
  public void addRelationshipsListener( de.infoasset.broker.interfaces.RelationshipsListener rl )
  ```
  - **Usage**
    - * Attaches a relationship listener to this relationships asset container. The listener is equivalent to the observer-pattern described by Gamma et. al. in "Design Patterns", but provides more methods and information about the occurred event.
  - **Parameters**
    - * rl - the relationships listener to attach
  
- countRelationshipsOfA
  ```java
  public int countRelationshipsOfA( java.lang.String aKind, java.lang.String aId, java.lang.String relationshipTypeId )
  ```
  - **Usage**
    - * Counts the number of relationships of an asset taking the A part of a relationship
  - **Parameters**
    - * aKind - the kind of asset a
    - * aId - the id of asset a
    - * relationshipTypeId - the type of the relationship
  - **Returns** - the number of relationships

- countRelationshipsOfAAndAuthor
  ```java
  public int countRelationshipsOfAAndAuthor( java.lang.String aKind, java.lang.String aId, java.lang.String authorKind, java.lang.String authorId, java.lang.String relationshipTypeId )
  ```
- Usage
  * Counts the number of relationships of an asset taking the A part of a relationship an a given author

- Parameters
  * aKind - the kind of asset a
  * aId - the id of asset a
  * authorKind - the kind of the author
  * authorId - the id of the author
  * relationshipTypeId - the type of the relationship

- Returns - the number of relationships

- countRelationshipsOfB
  public int countRelationshipsOfB(java.lang.String bKind, java.lang.String bId, java.lang.String relationshipTypeId)

  - Usage
    * Counts the number of relationships of an asset taking the B part of a relationship

  - Parameters
    * bKind - the kind of asset b
    * bId - the id of asset b
    * relationshipTypeId - the type of the relationship

  - Returns - the number of relationships

- countRelationshipsOfBAndState
  public int countRelationshipsOfBAndState(java.lang.String bKind, java.lang.String bId, java.lang.String state, java.lang.String relationshipTypeId)

  - Usage
    * Counts the number of relationships of an asset taking the B part of a relationship an a given author

  - Parameters
    * bKind - the kind of asset b
    * bId - the id of asset b
    * relationshipTypeId - the type of the relationship
    * state - Description of Parameter

  - Returns - the number of relationships

- createRelationship
  public Relationship createRelationship()
* Creates a new relationship.
  - **Returns** - the new created `Relationship`

- **createRelationship**
  public `Relationship` createRelationship( `java.lang.String` id )
  - **Usage**
    * Creates a new relationship with the relationshipType with the Id id.
  - **Parameters**
    * `id` - the id of relationshipType
  - **Returns** - the new `relationship`

- **getRelationship**
  public `Relationship` getRelationship( `java.lang.String` id )
  - **Usage**
    * Returns the relationship with this id.
  - **Parameters**
    * `id` - the id of the relationship
  - **Returns** - the relationship

- **getRelationshipOfAAndB**
  public `Relationship` getRelationshipOfAAndB( `java.lang.String` aKind, `java.lang.String` aId, `java.lang.String` bKind, `java.lang.String` bId, `java.lang.String` relationshipTypeId )
  - **Usage**
    * Returns a relationship between asset a and asset b with the given relationshiptype. If there are more relationships that fulfill the conditions no certain relationship is returned
  - **Parameters**
    * `aKind` - the kind of asset a
    * `aId` - the id of asset a
    * `bKind` - the kind of asset b
    * `bId` - the id of asset b
    * `relationshipTypeId` - the type of the relationship between asset a and b
  - **Returns** - the relationship between Asset a and Asset b

- **getRelationshipOfAAndBId**
  public `String` getRelationshipOfAAndBId( `java.lang.String` aKind, `java.lang.String` aId, `java.lang.String` bKind, `java.lang.String` bId, `java.lang.String` relationshipTypeId )
Usage

* Returns the id of a relationship between asset a and asset b with the given relationship type. If there are more relationships that fulfill the conditions no certain relationship is returned

Parameters

* aKind - the kind of asset a
* aId - the id of asset a
* bKind - the kind of asset b
* bId - the id of asset b
* relationshipTypeId - the type of the relationship between asset a and b

Returns - the id of the relationship between Asset a and Asset b

getRelationshipsOfA

```java
public Iterator getRelationshipsOfA( java.lang.String aKind, java.lang.String aId, java.lang.String relationshipTypeId )
```

Usage

* Return an iterator over relationships identified by the assetKind, assetId and the RelationshipType.

Parameters

* aKind - the kind of the asset A, that participates in the relationship
* aId - the id of the asset A, that participates in the relationship
* relationshipTypeId - Id of the RelationshipType Asset A participates in

Returns - iterator over relationships

getRelationshipsOfAAndAuthor

```java
public Iterator getRelationshipsOfAAndAuthor( java.lang.String aKind, java.lang.String aId, java.lang.String authorKind, java.lang.String authorId, java.lang.String relationshipTypeId )
```

Usage

* Returns all relationships of an asset taking the A part in the relationship with a given author and relationship type

Parameters

* aKind - the kind of asset a
* aId - the id of asset a
* authorKind - the kind of the author
* authorId - the id of the author
* relationshipTypeId - the type of the relationship

Returns - iterator over relationships
• getRelationshipsOfAAndAuthorIds
  public Iterator getRelationshipsOfAAndAuthorIds( java.lang.String aKind, java.lang.String aId, java.lang.String authorKind, java.lang.String authorId, java.lang.String relationshipTypeId )

  – Usage
  * Returns all relationship ids of an asset taking the A part in the relationship with a given author and relationship type

  – Parameters
  * aKind - the kind of asset a
  * aId - the id of asset a
  * authorKind - the kind of the author
  * authorId - the id of the author
  * relationshipTypeId - the type of the relationship

  – Returns - iterator over ids of relationships

• getRelationshipsOfAAndB
  public Iterator getRelationshipsOfAAndB( java.lang.String aKind, java.lang.String aId, java.lang.String bKind, java.lang.String bId )

  – Usage
  * Returns all relationships between asset a and asset b. The returned relationships may be of a different relationship type

  – Parameters
  * aKind - the kind of asset a
  * aId - the id of asset a
  * bKind - the kind of asset b
  * bId - the id of asset b

  – Returns - an iterator over relationships between Asset a and Asset b

• getRelationshipsOfAAndBIds
  public Iterator getRelationshipsOfAAndBIds( java.lang.String aKind, java.lang.String aId, java.lang.String bKind, java.lang.String bId )

  – Usage
  * Returns all relationship ids between asset a and asset b. The returned relationships may be of a different relationship type

  – Parameters
  * aKind - the kind of asset a
  * aId - the id of asset a
  * bKind - the kind of asset b
- **bId** - the id of asset b
  - **Returns** - an iterator over ids of relationships between Asset a and Asset b

- **getRelationshipsOfAAndState**
  public Iterator getRelationshipsOfAAndState( java.lang.String aKind, java.lang.String aId, java.lang.String state, java.lang.String relationshipTypeId )
  - **Usage**
    * Returns all relationships of an asset taking the A part in the relationship with a given state and relationship type
  - **Parameters**
    * aKind - the kind of asset a
    * aId - the id of asset a
    * state - the current state of the relationship
    * relationshipTypeId - the type of the relationship
  - **Returns** - iterator over relationships

- **getRelationshipsOfAAndStateIds**
  public Iterator getRelationshipsOfAAndStateIds( java.lang.String aKind, java.lang.String aId, java.lang.String state, java.lang.String relationshipTypeId )
  - **Usage**
    * Returns all relationship ids of an asset taking the A part in the relationship with a given state and relationship type
  - **Parameters**
    * aKind - the kind of asset a
    * aId - the id of asset a
    * state - the current state of the relationship
    * relationshipTypeId - the type of the relationship
  - **Returns** - iterator over ids of relationships

- **getRelationshipsOfAIds**
  public Iterator getRelationshipsOfAIds( java.lang.String aKind, java.lang.String aId, java.lang.String relationshipTypeId )
  - **Usage**
    * Return an iterator over relationship ids identified by the assetKind, assetId and the RelationshipType.
* relationshipTypeId - Id of the RelationshipType Asset A participates in
  - **Returns** - iterator over ids of relationships

---

- **getRelationshipsOfAuthor**
  public Iterator getRelationshipsOfAuthor(java.lang.String id,
                                           java.lang.String kind,
                                           java.lang.String relationshipTypeId)

  - **Usage**
    * Returns an Iterator of Relationships that have this author(kind and id)
      and are of this relationshipType.
  
  - **Parameters**
    * id - the id of the author asset
    * kind - of the asset (e.g. Person", "Classifier")
    * relationshipTypeId - Description of Parameter
  
  - **Returns** - iterator of relationships

---

- **getRelationshipsOfAuthorIds**
  public Iterator getRelationshipsOfAuthorIds(java.lang.String id,
                                             java.lang.String kind,
                                             java.lang.String type)

  - **Usage**
    * Returns an Iterator of relationship ids that have this author(kind and id)
      and are of this relationshipType.
  
  - **Parameters**
    * id - the id of the author asset
    * kind - of the asset (e.g. Person", "Classifier")
    * type - the type of the relationhip
  
  - **Returns** - iterator over ids of relationships

---

- **getRelationshipsOfB**
  public Iterator getRelationshipsOfB(java.lang.String bKind,
                                     java.lang.String bId,
                                     java.lang.String relationshipTypeId)

  - **Usage**
    * Return an iterator over relationships identified by the assetKind, assetId
      and the RelationshipType.
  
  - **Parameters**
    * bKind - the kind of the asset B, that participates in the relationship
    * bId - the id of the asset B, that participates in the relationship
    * relationshipTypeId - Id of the RelationshipType Asset B participates in
  
  - **Returns** - iterator over relationships
• *getRelationshipsOfBAndAuthor*

  public Iterator getRelationshipsOfBAndAuthor( java.lang.String bKind, java.lang.String bId, java.lang.String authorKind, java.lang.String authorId, java.lang.String relationshipTypeId )

  - Usage
    * Returns all relationships of an asset taking the B part in the relationship with a given author and relationship type

  - Parameters
    * bKind - the kind of asset b
    * bId - the id of asset b
    * authorKind - the kind of the author
    * authorId - the id of the author
    * relationshipTypeId - the type of the relationship

  - Returns - iterator over relationships

• *getRelationshipsOfBAndAuthorIds*

  public Iterator getRelationshipsOfBAndAuthorIds( java.lang.String bKind, java.lang.String bId, java.lang.String authorKind, java.lang.String authorId, java.lang.String relationshipTypeId )

  - Usage
    * Returns all relationship ids of an asset taking the B part in the relationship with a given author and relationship type

  - Parameters
    * bKind - the kind of asset b
    * bId - the id of asset b
    * authorKind - the kind of the author
    * authorId - the id of the author
    * relationshipTypeId - the type of the relationship

  - Returns - iterator over ids of relationships

• *getRelationshipsOfBAndState*

  public Iterator getRelationshipsOfBAndState( java.lang.String bKind, java.lang.String bId, java.lang.String state, java.lang.String relationshipTypeId )

  - Usage
    * Returns all relationships of an asset taking the B part in the relationship with a given state and relationship type

  - Parameters
    * bKind - the kind of asset b
    * bId - the id of asset b
• getState - the current state of the relationship
  • relationshipTypeId - the type of the relationship
  
  - Returns - iterator over relationships

- getRelationshipsOfBAndStateIds
public Iterator getRelationshipsOfBAndStateIds(java.lang.String bKind, java.lang.String bId, java.lang.String state, java.lang.String relationshipTypeId)

- Usage
  * Returns all relationship ids of an asset taking the B part in the relationship with a given state and relationship type.

- Parameters
  * bKind - the kind of asset B
  * bId - the id of asset B
  * state - the current state of the relationship
  * relationshipTypeId - the type of the relationship

- Returns - iterator over ids of relationships

- getRelationshipsOfBIds
public Iterator getRelationshipsOfBIds(java.lang.String bKind, java.lang.String bId, java.lang.String relationshipTypeId)

- Usage
  * Returns an iterator over relationship ids identified by the assetKind, assetId and the RelationshipType.

- Parameters
  * bKind - the kind of the asset B, that participates in the relationship
  * bId - the id of the asset B, that participates in the relationship
  * relationshipTypeId - Id of the RelationshipType Asset B participates in

- Returns - iterator over ids of relationships

- getRelationshipsOfIds
public Iterator getRelationshipsOfIds(java.lang.String id, java.lang.String kind)

- Usage
  * Returns an Iterator of relationship ids that are associated with this asset in any kind. That means, that either the A part or the B part or the author is this asset.

- Parameters
  * id - the id of asset
  * kind - of the asset (e.G. Person", Classifier")
Returns - iterator over ids of relationships

- getRelationshipsOfType
  public Iterator getRelationshipsOfType(java.lang.String type)
  - Usage
    * Returns an iterator over relationships of this relationship type.
  - Parameters
    * type - the id of the relationship type
  - Returns - iterator over relationships

- getRelationshipsOfTypeIds
  public Iterator getRelationshipsOfTypeIds(java.lang.String relationshipTypeId)
  - Usage
    * Returns an iterator over relationship ids of this relationship type.
  - Parameters
    * relationshipTypeId - Description of Parameter
  - Returns - iterator over ids of relationships

- remove
  public boolean remove(java.lang.String id)
  - Usage
    * Removes the relationship with this id.
  - Parameters
    * id - the id of the relationship to remove
  - Returns - true if the operation succeeded else false

- removeRelationshipsListener
  public void removeRelationshipsListener(de.infoasset.broker.interfaces.RelationshipsListener rl)
  - Usage
    * Detaches a relationships listener from this relationships asset container.
  - Parameters
    * rl - the relationships listener to detach

A.1.8 Interface RelationshipsListener

The interface is implemented by IMPRelationshipsListener to use as a superclass for an inner anonymous class.
Declaration

```java
public interface RelationshipsListener
    implements AssetContainerListener
```

Methods

- **aIdChanged**
  ```java
  public void aIdChanged(
      de.infoasset.broker.interfaces.RelationshipEvent e )
  ```
  - **Usage**
    - * This method is invoked whenever the id of the asset that takes part as A in this Relationship changes.
  - **Parameters**
    - * e - provides information about the occurred event

- **authorIdChanged**
  ```java
  public void authorIdChanged(
      de.infoasset.broker.interfaces.RelationshipEvent e )
  ```
  - **Usage**
    - * This method is invoked whenever the id of the author of this relationship changes.
  - **Parameters**
    - * e - provides information about the occurred event

- **authorKindChanged**
  ```java
  public void authorKindChanged(
      de.infoasset.broker.interfaces.RelationshipEvent e )
  ```
  - **Usage**
    - * This method is invoked whenever the kind of the author of this relationship changes.
  - **Parameters**
    - * e - provides information about the occurred event

- **bIdChanged**
  ```java
  public void bIdChanged(
      de.infoasset.broker.interfaces.RelationshipEvent e )
  ```
- **Usage**
  - *This method is invoked whenever the id of the asset that takes part as B in this Relationship changes.*

- **Parameters**
  - * `e` - provides informations about the occured event

---

- **roleIdChanged**
  ```java
  public void roleIdChanged(
  de.infoasset.broker.interfaces.RelationshipEvent e)
  ```

  - **Usage**
    - *This method is invoked whenever the role of the assets taking part in this relationship changes.*

  - **Parameters**
    - * `e` - provides information about the occured event

---

- **stateChanged**
  ```java
  public void stateChanged(
  de.infoasset.broker.interfaces.RelationshipEvent e)
  ```

  - **Usage**
    - *This method is invoked whenever the state of the relationship changes.*

  - **Parameters**
    - * `e` - provides informations about the occured event

---

- **weightChanged**
  ```java
  public void weightChanged(
  de.infoasset.broker.interfaces.RelationshipEvent e)
  ```

  - **Usage**
    - *This method is invoked whenever the weight of the relationship changes.*

  - **Parameters**
    - * `e` - Description of Parameter
de.infoasset.broker.interfaces.recommendation – RelationshipsListener
# Anhang B

## Package
de.infoasset.broker.interfaces.recommendation

<table>
<thead>
<tr>
<th>Package Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interfaces</strong></td>
<td></td>
</tr>
<tr>
<td>Rating</td>
<td>84</td>
</tr>
<tr>
<td>A rating is a (discrete) score assigned by aAsset to a Asset at a specific point in time.</td>
<td></td>
</tr>
<tr>
<td>RatingEvent</td>
<td>87</td>
</tr>
<tr>
<td>This interface describes a ratingevent.</td>
<td></td>
</tr>
<tr>
<td>Ratings</td>
<td>89</td>
</tr>
<tr>
<td>A collection of Rating s managed by the server.</td>
<td></td>
</tr>
<tr>
<td>RatingsListener</td>
<td>93</td>
</tr>
<tr>
<td>This interface describes the methods of an listener for a ratings asset-container.</td>
<td></td>
</tr>
<tr>
<td>RecommendationEngine</td>
<td>94</td>
</tr>
<tr>
<td>This interface describes the methods of a recommendation engine.</td>
<td></td>
</tr>
<tr>
<td>RecommendationEngines</td>
<td>98</td>
</tr>
<tr>
<td>A collection of RecommendationEngine s managed by the server.</td>
<td></td>
</tr>
</tbody>
</table>
B.1 Interfaces

B.1.1 INTERFACE Rating

A rating is a (discrete) score assigned by an asset to an asset at a specific point in time. Usually the assigner is person or group and the rated asset is either another person or an document.

Declaration

```java
public interface Rating
    implements de.infoasset.broker.interfaces.Asset
```

Methods

- `getAssignerId`
  ```java
  public String getAssignerId()
  ```
  - **Usage**
    - * Return the id of the asset that assigned this rating.
  - **Returns** - the id of the assigner

- `getAssignerKind`
  ```java
  public String getAssignerKind()
  ```
  - **Usage**
    - * Return the kind of the asset that assigned this rating.
  - **Returns** - the kind of the assigner

- `getAuthorId`
  ```java
  public String getAuthorId()
  ```
  - **Usage**
    - * Return the authorId of this Rating.
  - **Returns** - the id of the author

- `getAuthorKind`
  ```java
  public String getAuthorKind()
  ```
  - **Usage**
    - * Return the authorKind of this Rating. (e.g. a recommendationEngine for group-document rating)
- **Returns**: kind of the author

- **getComment**

  public String getComment()

  - **Usage**
    * Returns the comment for this rating
  - **Returns**: the comment

- **getCreationDate**

  public Date getCreationDate()

  - **Usage**
    * Return the date when the assigner created this rating. To get the date when the rating was assigned use the getLastModification() method.
  - **Returns**: the date of creation

- **getFloatRating**

  public double getFloatRating()

  - **Usage**
    * Return the rating value.
  - **Returns**: the rating which is guaranteed to be different from NO_RATING.

- **getRatedId**

  public String getRatedId()

  - **Usage**
    * Return the id of the asset that is rated by this rating.
  - **Returns**: the id of the rated asset

- **getRatedKind**

  public String getRatedKind()

  - **Usage**
    * Return the kind of the asset that is rated by this rating.
  - **Returns**: kind of the rated asset

- **getRating**

  public int getRating()

  - **Usage**
    * Return the rating value.
  - **Returns**: the rating which is guaranteed to be different from NO_RATING.
• **setAssignerId**
  public void setAssignerId( java.lang.String assignerId )

  - **Usage**
    * Update the id of the person that assigned this rating.
  - **Parameters**
    * assignerId - the new id of assigner

• **setAssignerKind**
  public void setAssignerKind( java.lang.String assignerKind )

  - **Usage**
    * Update the kind of the asset that assigned this rating.
  - **Parameters**
    * assignerKind - the new kind of the assigner

• **setAuthorId**
  public void setAuthorId( java.lang.String id )

  - **Usage**
    * Update the authorId
  - **Parameters**
    * id - the new id of author

• **setAuthorKind**
  public void setAuthorKind( java.lang.String kind )

  - **Usage**
    * Update the authorKind (e.g. Person or RecommendationEngine”
  - **Parameters**
    * kind - the new kind of the author

• **setComment**
  public void setComment( java.lang.String comment )

  - **Usage**
    * Update the comment.
  - **Parameters**
    * comment - the new comment for this rating

• **setFloatRating**
  public void setFloatRating( double rating )
de.infoasset.broker.interfaces.recommendation– RatingEvent

- **Usage**
  * Update the rating value to a value different from NO_RATING.

- **Parameters**
  * `rating` - the new rating value

- **setRatedId**
  
  public void `setRatedId` ( java.lang.String `assetId` )

  - **Usage**
    * Update the id of the asset that rated by this rating.
  
  - **Parameters**
    * `assetId` - the new id of the rated asset

- **setRatedKind**
  
  public void `setRatedKind` ( java.lang.String `ratedKind` )

  - **Usage**
    * Update the kind of the asset that rated by this rating.
  
  - **Parameters**
    * `ratedKind` - the new kind of the rated asset

- **setRating**
  
  public void `setRating` ( int `rating` )

  - **Usage**
    * Update the rating value to a value different from NO_RATING.
  
  - **Parameters**
    * `rating` - the new rating value

### B.1.2 INTERFACE RatingEvent

This interface describes a rating event. The event is fired when a rating is added or removed from the ratings asset container or the relationship is changed. You can get the former values of the changed attributes via the get methods. The get Methods of non changed attributes return `null`.

**Declaration**

```java
public interface RatingEvent
    implements de.infoasset.broker.interfaces.AssetEvent
```
Methods

- `getOldAssignerId`
  
  ```java
  public String getOldAssignerId()
  ```
  
  - **Usage**
    * Returns the former id of the assigner of the rating.
  - **Returns** - the former id or `null` if the id of the assigner is not changed.

- `getOldAuthorId`
  
  ```java
  public String getOldAuthorId()
  ```
  
  - **Usage**
    * Returns the former id of the asset assigned to the rating as the author.
  - **Returns** - the former id or `null` if the id of the asset is not changed.

- `getOldAuthorKind`
  
  ```java
  public String getOldAuthorKind()
  ```
  
  - **Usage**
    * Returns the former kind of the asset assigned to the rating as the author.
  - **Returns** - the former kind or `null` if the id of the asset is not changed.

- `getOldFloatRating`
  
  ```java
  public double getOldFloatRating()
  ```
  
  - **Usage**
    * Returns the former rating value of the rating.
  - **Returns** - the former rating value or `null` if the rating value of the rating is not changed.

- `getOldRatedId`
  
  ```java
  public String getOldRatedId()
  ```
  
  - **Usage**
    * Returns the former id of the asset rated by the rating.
  - **Returns** - the former id or `null` if the id is not changed.

- `getOldRating`
  
  ```java
  public int getOldRating()
  ```
  
  - **Usage**
    * Returns the former rating value of the rating.
– **Returns** - the former rating value or null if the rating value of the rating is not changed

• `getOldState`
  
  ```java
  public String getOldState()
  ```

  – **Usage**
  * Returns the former state of the rating. The state is a domainvalue.
  
  – **Returns** - the former state of the rating or null if the state is not changed

### B.1.3 INTERFACE Ratings

A collection of Ratings managed by the server. The ratings assetcontainer manages ratings of one type. That means that the ratings in the container are assigned by assets of a certain kind and rate a asset of the same or another certain kind. This leads to the result that there might be several ratings assetcontainer in the broker. (Person rates Person; Person rates Document; Group rates Document)

#### Declaration

```java
public interface Ratings
implements de.infoasset.broker.interfaces.AssetContainer
```

#### Fields

- **public static final int NO_RATING**
  
  – This rating is returned if no rating is available.

- **public static final int DEFAULT_RATING**
  
  – This rating is the default value used to initialize ratings.

- **public static final int BEST_RATING**
  
  – This is the best possible rating.

- **public static final int WORST_RATING**
  
  – This is the worst rating possible.
Methods

- **addRatingsListener**
  
  ```java
  public void addRatingsListener(
      de.infoasset.broker.interfaces.recommendation.RatingsListener rl )
  ```

  **Usage**

  * Attaches a ratingslistener to this ratings assetcontainer. The methods of the ratingslistener are invoked on certain events. The ratingslistener is equivalent to the observer pattern described by Gamma et. al. in "Design Patterns" but provides the observer more information about the occurred event.

  **Parameters**

  * rl - the ratingslistener to attach

- **createRating**

  ```java
  public Rating createRating( java.lang.String assignerId, java.lang.String ratedId, int rating )
  ```

  **Usage**

  * Return a newly created rating of this person/group for this person/document with this score and with the current time stamp. This rating itself has a unique assetId. It is possible to create multiple ratings of the same person/group for the same person/document.

  **Parameters**

  * assignerId - the id of the asset, that assigns the rating
  * ratedId - the id of the asset, that is rated
  * rating - the rating value, must be between Ratings.WORST_RATING and Ratings.BEST_RATING

  **Returns** - the newly created rating

- **getRating**

  ```java
  public Rating getRating( java.lang.String assetId )
  ```

  **Usage**

  * Return the rating with this assetId

  **Parameters**

  * assetId - the assetId to look for

  **Returns** - the rating with the assetId or null if no such rating exists.

- **getRatingOfAssignerForRated**

  ```java
  public Rating getRatingOfAssignerForRated( java.lang.String assignerId, java.lang.String ratedId )
  ```
• **getRatingOfAssignerForRatedId**
  public String getRatingOfAssignerForRatedId(java.lang.String assignerId, java.lang.String ratedId)

  - **Usage**
    * Returns the id of the rating that has the given assigner and rated asset.
  - **Parameters**
    * assignerId - the id of the assigner
    * ratedId - the id of the rated asset
  - **Returns** - the id, or null if no such rating exists.

• **getRatingsFor**
  public Iterator getRatingsFor(java.lang.String ratedId)

  - **Usage**
    * Returns an iterator of ratings that rate a asset with this ratedId.
  - **Parameters**
    * ratedId - look for assets, that rate assets with this id
  - **Returns** - an iterator over rating(s)

• **getRatingsForWithAuthor**
  public Iterator getRatingsForWithAuthor(java.lang.String ratedId, java.lang.String authorKind, java.lang.String authorId)

  - **Usage**
    * Returns an iterator of ratings that rate a asset with this ratedId and the given author.
  - **Parameters**
    * ratedId - look for assets, that rate assets with this id
    * authorKind - the kind of the author
    * authorId - the id of the author
  - **Returns** - an iterator over rating(s)

• **getRatingsOf**
  public Iterator getRatingsOf(java.lang.String assignerId)
- **Usage**
  
  * Returns an iterator of rating(s), that have the assigner with this assignerId.

- **Parameters**
  
  * assignerId - look for ratings with this id

- **Returns** - an iterator over rating(s)

---

**getRatingsOfWithAuthor**

```java
public Iterator getRatingsOfWithAuthor(java.lang.String assignerId, java.lang.String authorKind, java.lang.String authorId)
```

- **Usage**
  
  * Returns an iterator of rating(s), that have the assigner with this assignerId and the given author.

- **Parameters**
  
  * assignerId - look for ratings with this id
  * authorKind - the kind of the author
  * authorId - the id of the author

- **Returns** - an iterator over rating(s)

---

**iterator**

```java
public Iterator iterator()
```

- **Usage**
  
  * Returns all ratings in this assetcontainer

---

**remove**

```java
public boolean remove(java.lang.String assetId)
```

- **Usage**
  
  * Remove the rating with this assetId. Return false if no such rating exists.

- **Parameters**
  
  * assetId - the id of the asset to be removed

- **Returns** - true if the operation succeeded, else false

---

**removeRatingsListener**

```java
public void removeRatingsListener(de.infoasset.broker.interfaces.recommendation.RatingsListener rl)
```

- **Usage**
  
  * Detaches the ratingslistener from this assetcontainer.

- **Parameters**
  
  * rl - the ratingslistener to detach
B.1.4 INTERFACE RatingsListener

This interface describes the methods of an listener for a ratings assetcontainer. The listener is equivalent to the observer design pattern described by Gamma et. al. in "Design Patterns"

DECLARATION

```java
public interface RatingsListener
    implements de.infoasset.broker.interfaces.AssetContainerListener
```

METHODS

- `assignerIdChanged`
  ```java
  public void assignerIdChanged(
      de.infoasset.broker.interfaces.recommendation.RatingEvent e )
  ```
  - Usage
    * The method is invoked whenever the id of the assigner is changed and persistency is granted.
  - Parameters
    * e - a ratingevent, provides informations about the event occured

- `authorIdChanged`
  ```java
  public void authorIdChanged(
      de.infoasset.broker.interfaces.recommendation.RatingEvent e )
  ```
  - Usage
    * The method is invoked whenever the id of the author is changed and persistency is granted.
  - Parameters
    * e - a ratingevent, provides informations about the event occured

- `authorKindChanged`
  ```java
  public void authorKindChanged(
      de.infoasset.broker.interfaces.recommendation.RatingEvent e )
  ```
  - Usage
    * The method is invoked whenever the kind of the author is changed and persistency is granted.
  - Parameters

B.1.5 INTERFACE RecommendationEngine

This interface describes the methods of a recommendation engine.

Declaration

```java
public interface RecommendationEngine
    implements de.infoasset.broker.interfaces.Asset
```
Methods

- **getActions**
  public Iterator getActions()

  - **Usage**
    * Returns an iterator over Strings. These strings are domainValue Ids and describe actions, that the engine can perform.
  - **Returns** - iterator over domain values of possible actions

- **getClassifiers**
  public Iterator getClassifiers()

  - **Usage**
    * Return the associated classifiers of this engine.
  - **Returns** - iterator over classifiers

- **getClassifiersIds**
  public Iterator getClassifiersIds()

  - **Usage**
    * Return the associated classifiers of this engine.
  - **Returns** - iterator over the id's of the classifiers

- **getDefaultWeights**
  public Iterator getDefaultWeights()

  - **Usage**
    * Return an iterator over relationships that describe the default weights of the referenced recommendation engine in the calculation of this engine.
  - **Returns** - iterator over Relationships

- **getDescription**
  public String getDescription()

  - **Usage**
    * Return a textual description for this recommendation engine.
  - **Returns** - the description

- **getPersonalWeights**
  public Iterator getPersonalWeights(java.lang.String personId)

  - **Usage**
* Return an iterator over relationships that describe the weights of the referenced recommendation engine in the calculation of this engine.

- **Parameters**
  * personId - the weights for this person
- **Returns** - iterator over Relationships

---

**getRecommendation**

```java
public double getRecommendation(String personId, de.infoasset.broker.interfaces.Asset asset)
```

- **Usage**
  * Returns a recommendation value for this the person and this asset.
- **Parameters**
  * personId - the id of the person
  * asset - the id of the asset
- **Returns** - the recommendation value

---

**getRecommendation**

```java
public double getRecommendation(String personId, String assetId)
```

- **Usage**
  * Returns a recommendation value for this the person and this asset.
- **Parameters**
  * personId - the id of the person
  * assetId - the id of the asset
- **Returns** - the recommendation value

---

**getTopRecommendationIds**

```java
public Iterator getTopRecommendationIds(String personId)
```

- **Usage**
  * Return an iterator over assetIds, that have a positive recommendation. The iterator is sorted by the recommendation values.
- **Parameters**
  * personId - use recommendation values calculated for this person
- **Returns** - the sorted top assets

---

**isDefaultEngine**

```java
public boolean isDefaultEngine()
```

- **Usage**
• isTopRecommendationEngine
  public boolean isTopRecommendationEngine()

  - Usage
    * Return whether this recommendation engine makes use of
      recommendation values calculated by other recommendation engines.
  - Returns - true, if this engine makes use of other engines.

• performAction
  public void performAction( java.lang.String  action )

  - Usage
    * The engine performs the action.
  - Parameters
    * action - the action to perform

• setDescription
  public void setDescription( java.lang.String  description )

  - Usage
    * Update the description for this recommendation engine.
  - Parameters
    * description - the new description

• sortAssetIdsByRecommendation
  public Iterator sortAssetIdsByRecommendation( java.lang.String  personId, java.util.Iterator  assetIds )

  - Usage
    * Return a iterator over assetIds sorted by the asset’s recommendation
      value. The most recommendet Asset is returned first
  - Parameters
    * personId - use the recommendation values calculated for this person
    * assetIds - Description of Parameter
  - Returns - the sorted iterator over assetIds
• sortAssetsByRecommendation
  public Iterator sortAssetsByRecommendation( java.lang.String personId, java.util.Iterator assets )

  – Usage
  * Return a iterator over assets sorted by the asset’s recommendation value.
    The most recommendet Asset is returned first
  – Parameters
  * personId - use the recommendation values calculated for this person
  * assets - Description of Parameter
  – Returns - the sorted iterator over assets

B.1.6 INTERFACE RecommendationEngines

A collection of RecommendationEngine s managed by the server. The recommendation engines
assetconrainer manages the recommendation engines of the system. There are maximum 6
different recommendation engines available in the current implementation.

DECLARATION

public interface RecommendationEngines
  implements de.infoasset.broker.interfaces.AssetContainer

FIELDS

• public static final String PERSON_RECOMMENDER
  – The id of the selected person recommender (a Recommendation Engine).
• public static final String GROUP_RECOMMENDER
  – The id of the selected group recommender (a Recommendation Engine).
• public static final String PERSONS_RECOMMENDER
  – The id of the favourite persons recommender (a Recommendation Engine).
• public static final String GROUPS_RECOMMENDER
  – The id of the membership recommender (a Recommendation Engine).
• public static final String CLUSTER_RECOMMENDER
• The id of a cluster recommender (a Recommendation Engine).

• public static final String SUM_RECOMMENDER
  – The id of the summarizing recommender (a Recommendation Engine).

METHODS

• createRecommendationEngine
  public RecommendationEngine createRecommendationEngine(
      java.lang.String id, java.lang.String className )

  – Usage
    * Description of the Method
  – Parameters
    * id - Description of Parameter
    * className - Description of Parameter
  – Returns - Description of the Returned Value

• getDefault
  public RecommendationEngine getDefault( )

  – Usage
    * Returns the default recommendation engine. The default engine should be
      used when a person has not selected a favourite one.
  – Returns - the default recommendation engine of the system

• getDefaultId
  public String getDefaultId( )

  – Usage
    * Return the id of the default recommendation engine
  – Returns - the id of the recommendation engine

• getFavouriteEngine
  public RecommendationEngine getFavouriteEngine( java.lang.String personId )

  – Usage
    * Returns the the selected Recommendation Engine for this user. If the user
      has no favourite engine the default engine is returned.
  – Parameters
    * personId - the id of the person
- **Returns** - the recommendation engine to use as the default

**getRecommendationEngine**

```java
public RecommendationEngine getRecommendationEngine(
        java.lang.String id)
```

- **Usage**
  - Return the recommendation engine with the given id.
- **Parameters**
  - id - the id of the engine
- **Returns** - the recommendation engine

**getRecommendationEngines**

```java
public Iterator getRecommendationEngines()
```

- **Usage**
  - Return an iterator over all recommendation engines of this container. The recommendation Engines are sorted by its' ids.
- **Returns** - sorted iterator over recommendation engines

**getRecommendationEnginesIds**

```java
public Iterator getRecommendationEnginesIds()
```

- **Usage**
  - Return an iterator over the ids of all recommendation engines of this container. The recommendation Engines are sorted by its' ids.
- **Returns** - sorted iterator over ids of recommendation engines

**setDefault**

```java
public void setDefault(
        de.infoasset.broker.interfaces.recommendation.RecommendationEngine engine)
```

- **Usage**
  - Update the id of the default recommendation engine.
- **Parameters**
  - engine - the new default recommendation engine

**setFavouriteRecommendationEngine**

```java
public void setFavouriteRecommendationEngine(
        java.lang.String personId, java.lang.String id)
```

- **Usage**
* Updates the favourite Recommendation Engine for this person and this engine.

- **Parameters**
  * **personId** - the id of the Person
  * **id** - the id of the recommendation engine
Anhang C

Package
de.infoasset.broker.interfaces.classification

Package Contents

<table>
<thead>
<tr>
<th>Interface</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Categorization</td>
<td>104</td>
</tr>
<tr>
<td>Categorizations</td>
<td>105</td>
</tr>
<tr>
<td>Category</td>
<td>107</td>
</tr>
<tr>
<td>Classification</td>
<td>108</td>
</tr>
<tr>
<td>Classifier</td>
<td>112</td>
</tr>
<tr>
<td>Classifiers</td>
<td>116</td>
</tr>
</tbody>
</table>

- Categorization: A categorization is a collection of categories.
- Categorizations: The interface Categorizations represents a collection of Categorizations.
- Category: The interface Category represents one category.
- Classification: A classification represents the assignment of an asset to a category.
- Classifier: This interface describes the methods of a classifier.
- Classifiers: The interface Classifiers represents a collection of Classifiers.
C.1 Interfaces

C.1.1 Interface Categorization

A categorization is a collection of categories.

Declaration

```java
public interface Categorization
    implements de.infoasset.broker.interfaces.Asset
```

Methods

- `createCategory`
  ```java
  public Category createCategory( )
  ```
  - **Usage**
    * Creates a new category.
  - **Returns** - the new created category

- `getCategory`
  ```java
  public Category getCategory( java.lang.String id )
  ```
  - **Usage**
    * Returns the category with this id or `null` if there is no corresponding category
  - **Parameters**
    * `id` - the id of the category
  - **Returns** - the category or `null` if there isn’t such a category

- `getDescription`
  ```java
  public String getDescription( )
  ```
  - **Usage**
    * Return the description of this categorization.
  - **Returns** - the textual description for this category

- `iterator`
  ```java
  public Iterator iterator( )
  ```
• **Usage**
  * Returns an iterator of all categories in this categorization.

• **Returns** - iterator over category

• **iteratorIds**
  public Iterator iteratorIds()

  • **Usage**
    * returns an iterator of all categoryIds in this categorization.

  • **Returns** - iterator over the ids (String) of the categories

• **removeCategory**
  public boolean removeCategory( java.lang.String id )

  • **Usage**
    * Removes the category with this id from this categorization. The category must be empty.

  • **Parameters**
    * id - the id of the category

  • **Returns** - false if the operation failed, else true

• **setDescription**
  public void setDescription( java.lang.String desc )

  • **Usage**
    * Update the description of this categorization.

  • **Parameters**
    * desc - the new description for this category

• **setServices**
  public void setServices( de.infoasset.broker.interfaces.Services services )

  • **Usage**
    * the implementations need an instance of Services ## to be removed !!
      (All assets know the (Singelton) Services Class)

  • **Parameters**
    * services - the services Singelton

**C.1.2 INTERFACE Categorizations**

The interface Categorizations represents a collection of Categorizations.
### Declaration

```java
public interface Categorizations
    implements de.infoasset.broker.interfaces.AssetContainer
```

#### Methods

- **clear**
  ```java
  public void clear()
  ```
  - **Usage**
    * Removes all categorizations from the container.

- **createCategorization**
  ```java
  public Categorization createCategorization(java.lang.String id,
                                               java.lang.String className)
  ```
  - **Usage**
    * Creates a new categorization which has the type className and the Id id
  - **Parameters**
    * `id` - the id of the categorization
    * `className` - the name of the class, that the categorization is a instance of
  - **Returns** - new created categorization

- **getCategorization**
  ```java
  public Categorization getCategorization(java.lang.String id)
  ```
  - **Usage**
    * Returns the categorization with the given asset id. If the categorization does not exist, return `null`.
  - **Parameters**
    * `id` - the id of the categorization
  - **Returns** - the categorization

- **getCategorizationIds**
  ```java
  public Iterator getCategorizationIds()
  ```
  - **Usage**
    * Returns an iterator over all existing categorization ids. The order of the ids is undetermined.
  - **Returns** - iterator over categorization ids
C.1.3  INTERFACE Category

The interface Category represents one category

DECLARATION

```java
public interface Category
    implements de.infoasset.broker.interfaces.Asset
```

METHODS

- **getAssetIds**
  ```java
  public Iterator getAssetIds()
  ```
  - **Usage**
    - * Returns an iterator of all AssetIds, that possess a Classification to this Category
  - **Returns** - iterator over the ids of assets

- **getCategorization**
  ```java
  public String getCategorization()
  ```
  - **Usage**
    - * Returns the categorization this category belongs to
  - **Returns** - the id of the categorization

- **getClassificationIds**
  ```java
  public Iterator getClassificationIds()
  ```
  - **Usage**
    - * Returns an iterator of all ClassificationIds, that reference this Category
  - **Returns** - iterator over the ids of classifications

- **getClassifications**
  ```java
  public Iterator getClassifications()
  ```
  - **Usage**
    - * Returns an iterator of all Classifications, that reference this Category
  - **Returns** - iterator over classifications
• *getComment*
  public String getComment() {
  
  – *Usage*
  * Gets the comment of this Category
  
  – *Returns* - the comment

• *setCategorization*
  public void setCategorization(java.lang.String categorizationId) {
  
  – *Usage*
  * Updates the categorization this category belongs to.
  
  – *Parameters*
  * categorizationId - the new categorizationId

• *setComment*
  public void setComment(java.lang.String comment) {
  
  – *Usage*
  * Sets the comment of this Category
  
  – *Parameters*
  * comment - the new comment

C.1.4 INTERFACE Classification

A classification represents the assignment of an asset to a category.

DECLARATION

```java
public interface Classification
  implements de.infoasset.broker.interfaces.Asset
```

FIELDS

• public static final double confidenceFactor
  – number og significant decimal places := log_{10}(confidenceFactor)
Methods

- **getAssetId**
  ```java
  public String getAssetId() {
  }
  ```
  - **Usage**
    - Return the id of the assigned asset.
  - **Returns** - the id of the classified asset.

- **getAssetKind**
  ```java
  public String getAssetKind() {
  }
  ```
  - **Usage**
    - Return the kind of the assigned asset. E.G. possible kinds are person
      concept
  - **Returns** - the kind of the asset

- **getAuthorId**
  ```java
  public String getAuthorId() {
  }
  ```
  - **Usage**
    - Return the id of the author, that created the classification.
  - **Returns** - the id of the author

- **getAuthorKind**
  ```java
  public String getAuthorKind() {
  }
  ```
  - **Usage**
    - Return the kind of the author, that created the classification.
  - **Returns** - the kind of the author

- **getCategoryId**
  ```java
  public String getCategoryId() {
  }
  ```
  - **Usage**
    - Return the id of the category, that the asset is assigned to.
  - **Returns** - the id of the category

- **getCategoryKind**
  ```java
  public String getCategoryKind() {
  }
  ```
  - **Usage**
    - Return the kind of the category. E.G. possible kinds are "group concept"
• **getComment**
  public String getComment()
  
  − **Usage**
    * Return the comment for this classification.
  − **Returns** - the comment

• **getConfidence**
  public double getConfidence()
  
  − **Usage**
    * Return the confidence value. The confidence value may be interpreted as a probability, that a asset belongs to a category.
  − **Returns** - the confidence value

• **getCreationDate**
  public Date getCreationDate()
  
  − **Usage**
    * Return the date of the creation.
  − **Returns** - the creation date

• **getState**
  public String getState()
  
  − **Usage**
    * Return the state of the classification. As the classification is possible the result of a classification process it might be reviewed by a real person, so that the state of the classification can change. E.G. suggested is possible state.
  − **Returns** - the id of a DomainValue

• **setAssetId**
  public void setAssetId( java.lang.String id )
  
  − **Usage**
    * Update the id of the assigned asset.
  − **Parameters**
    * id - the new id of the assigned asset in this classification

• **setAssetKind**
  public void setAssetKind( java.lang.String kind )
- **Usage**
  * Updates the kind of the assigned asset.

- **Parameters**
  * `kind` - the new kind of the asset

---

- **setAuthorId**
  ```java
  public void setAuthorId( java.lang.String id )
  ```

- **Usage**
  * Updates the id of the author, that created the classification.

- **Parameters**
  * `id` - the id of the new author

---

- **setAuthorKind**
  ```java
  public void setAuthorKind( java.lang.String kind )
  ```

- **Usage**
  * Updates the kind of the author, that created the classification.

- **Parameters**
  * `kind` - the kind of the new author

---

- **setCategoryId**
  ```java
  public void setCategoryId( java.lang.String id )
  ```

- **Usage**
  * Update the id of the category, that the asset is assigned to.

---

- **setCategoryKind**
  ```java
  public void setCategoryKind( java.lang.String kind )
  ```

- **Usage**
  * Updates the kind of the category, the the asset is assigned to.

---

- **setComment**
  ```java
  public void setComment( java.lang.String comment )
  ```

- **Usage**
  * Update the comment for this classification.

- **Parameters**
  * `comment` - the new comment for this classification

---

- **setConfidence**
  ```java
  public void setConfidence( double confidence )
  ```
- **Usage**
  * Update the confidence value.

- **Parameters**
  * `confidence` - the new confidence value

---

- **setCreationDate**
  `public void setCreationDate( java.util.Date date )`

  - **Usage**
    * Update the date of the creation. Usually the creation date should be set once at the creation time.

  - **Parameters**
    * `date` - the date of the creation

---

- **setState**
  `public void setState( java.lang.String state )`

  - **Usage**
    * Updates the state of a classification.

  - **Parameters**
    * `state` - the new state for this classification

---

- **write**
  `public boolean write( )`

  - **Usage**
    * Makes changes of the classification persistent.

  - **Returns** - true, if successful else false

### C.1.5 INTERFACE Classifier

This interface describes the methods of a classifier. Usually the implementing class wraps a third part classifier.

**Declaration**

```java
public interface Classifier
    implements de.infoasset.broker.interfaces.Asset
```
Methods

- **build**
  
  `public boolean build(boolean parallel)`
  
  - **Usage**
    * does only work if the classifier is either in State CREATED or READY after this operation the Classifier is in State BUILDING until it is built entirely
  
  - **Returns** - whether the operation succeeded
  
  - **Exceptions**
    * if - the ClassifierEngine isn't in the right state

- **classify**
  
  `public boolean classify(boolean parallel)`
  
  - **Usage**
    * does only work if the classifier is in State READY classifies all Assets which come from a special source (e.g. all Assets from some kind of Classification)
  
  - **Parameters**
    * parallel - tells whether the operation will be executed blocking
  
  - **Returns** - whether the operation succeeded
  
  - **Exceptions**
    * if - the ClassifierEngine isn't in the right state

- **classify**
  
  `public boolean classify(java.lang.String assetId, boolean parallel)`
  
  - **Usage**
    * does only work if the classifier is in State READY classifies the Asset with the Id assetId
  
  - **Parameters**
    * parallel - tells whether the operation will be executed blocking
  
  - **Returns** - whether the operation succeeded
  
  - **Exceptions**
    * if - the ClassifierEngine isn't in the right state

- **closeLogs**
  
  `public boolean closeLogs()`
  
  - **Usage**
* Closes the log files of this classifier.

• **getCategorization**
  public Categorization **getCategorization**( )

  – **Usage**
    * Return the associated Categorization this Classifier classifies into.
  – **Returns** - the associated categorization

• **getCategorizationId**
  public String **getCategorizationId**( )

  – **Usage**
    * Return the Id of the associated Categorization this Classifier classifies into.
  – **Returns** - the id of the associated Categorization

• **getDescription**
  public String **getDescription**( )

  – **Usage**
    * Return the description for this classifier
  – **Returns** - a textual description for this classifier

• **getState**
  public String **getState**( )

  – **Usage**
    * Returns the current state of the Classifier
  – **Returns** - the current state, the id of a domainValue

• **getType**
  public String **getType**( )

  – **Usage**
    * Returns the type of this classifier. To get the real name of this type get the equivalent domainValue.
  – **Returns** - the domainValue id of the classifier type

• **initEngine**
  public void **initEngine**( )

  – **Usage**
    * Initializes the engine.
- **log**
  public boolean log()

- **makePersistent**
  public boolean makePersistent( boolean parallel )
  
  - Usage
    * Makes the state of the classifier persistent.
  
  - Parameters
    * parallel - start the process in a own thread or not
  
  - Returns - true. if the classifier was successfully made persistent, else false

- **setCategorizationId**
  public void setCategorizationId( java.lang.String id )
  
  - Usage
    * Updates the id of the associated categorization
  
  - Parameters
    * id - the id of the new categorization

- **setDescription**
  public void setDescription( java.lang.String description )
  
  - Usage
    * Updates the textual description for this classifier.
  
  - Parameters
    * the - new textual description

- **setServices**
  public void setServices( de.infoasset.broker.interfaces.Services services )
  
  - Usage
    * The Classifiers need to know their Services.
  
  - Parameters
    * services - the services singelton

- **setType**
  public void setType( java.lang.String type )
  
  - Usage
    * Updates the classifier type.
  
  - Parameters
    * type - the new type of this Classifier, must be the id of a domainValue
C.1.6 INTERFACE Classifiers

The interface Classifiers represents a collection of Classifiers.

DECLARATION

```java
public interface Classifiers
    implements de.infoasset.broker.interfaces.AssetContainer
```

METHODS

- **clear**
  ```java
  public void clear()
  ```
  - **Usage**
    - Removes all classifiers from the container.

- **createClass**
  ```java
  public Classifier createClassifier( java.lang.String id,
                java.lang.String className )
  ```
  - **Usage**
    - Creates a new classifier that is unrelated to any other existing classifier.
  - **Parameters**
    - *id* - the id of the new classifier
    - *className* - the classname of the implementing class
  - **Returns** - the new created classifier

- **getClassifier**
  ```java
  public Classifier getClassifier( java.lang.String id )
  ```
  - **Usage**
    - Returns the classifier with the given asset id. If the classifier does not exist, return null.
  - **Parameters**
    - *id* - the id of the classifier
  - **Returns** - the classifier

- **getClassifierForCategorization**
  ```java
  public Classifier getClassifierForCategorization( java.lang.String categorizationId )
  ```
Usage
* Returns the Classifier which belongs to the Categorization with the Id categorizationId

Parameters
* categorizationId - the id of the categorization

Returns - the classifier, that belongs to the categorization

• getClassifierIds
public Iterator getClassifierIds( )

Usage
* Returns an iterator over all existing classifier ids. The order of the ids is undetermined.

Returns - an iterator over the ids of all classifiers

• getClassifiers
public Iterator getClassifiers( )

Usage
* Returns an iterator over all existing classifiers. The order of classifiers is undetermined.

Returns - an iterator over all classifiers
Literaturverzeichnis


B. Mobasher et al. 97: B. Mobasher, E. Han, Karypis, G., und Kumar, V. *Clustering based on association rule hypergraphs*, 1997.


Zaïane et al. 98: Zaïane, Osmar R., Xin, Man, und Han, Jiawei. Discovering Web Access Patterns and Trends by Applying OLAP and Data Mining Technology on Web Logs, 1998.