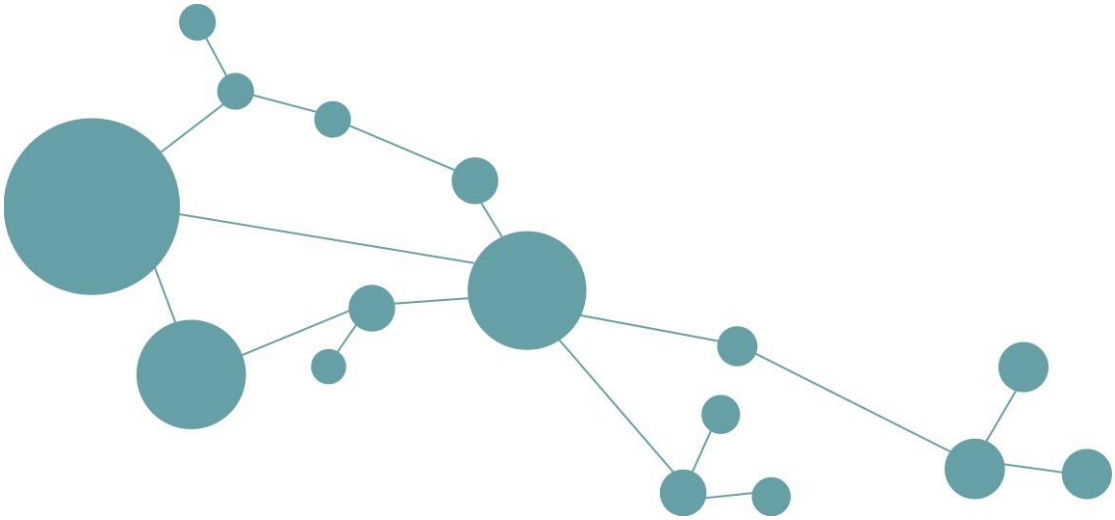

KEM 3 Cookbook

i·views

Inhaltsverzeichnis

1 Voraussetzungen

Das KEM-Cookbook soll einen einfachen Einstieg in die KEM-Programmierung ermöglichen. Dazu werden eine Reihe von typischen Anwendungsbeispielen gegeben. Notwendig hierzu sind dabei Kenntnisse über das KInfinity-Wissensnetzmodell.

2 Überblick

KEM ist die Abkürzung für KInfinity-Editor-Modell. Die KEM-API ist eine Programmierschnittstelle, die eine spezialisierte Sicht auf das KInfinity-Wissensnetzmodell zur Verfügung stellt. Jedes Wissensnetz-Objekt wird durch ein KEM-Editorobjekt repräsentiert, welches bestimmte Zugriffsmethoden auf das Objekt zur Verfügung
stellt. Beispielsweise erhält man als Antwort auf die Frage nach einem Begriffsattribut einen KEMAttributeEditor, der über die Methode getValueString() den aktuellen Attributwert zurückliefert.

Die KEM-API ansich ist programmiersprachenunabhängig definiert. Die Schnittstellendefinition besteht, wie der Name schon besagt, aus einer Reihe von Interfaces. Implementiert ist KEM durch eine KInfinity-Serverkomponente, der KEM-Bridge. Zudem liegt derzeit eine Proxy-Implementierung in Java vor. Diese beiden Komponenten kommunizieren untereinander über eine TCP/IP-Verbindung. Zusammengenommen stellen sie eine logische Einheit dar, welche transparent in einer Java-Umgebung anprogrammiert werden kann. Folgendes Diagramm veranschaulicht dies.

Der Zugriff auf KInfinity-Daten über KEM muss in Sessions organisiert werden. Bevor ein konkretes Netzobjekt "angefasst" werden kann, muss eine KEMSession gestartet werden. KEMSessions werden unterteilt in Read- und Edit-Sessions. Read-Sessions sind für den regulären, schnellen Lesezugriff gedacht. Eine solche Session bietet keinen Transaktionsschutz und kostet wenig Ressourcen. Sie sind also für Anwendungsfälle
geeignet, bei denen es ausschließlich um Lesezugriffe geht, so z.B. Navigation oder Suche. Ressourcen kostspieliger sind Edit-Sessions. Logisch gesehen läßt sich eine solche als zeitlich ausgedehnte Transaktion verstehen. Sämtliche Zugriffe innerhalb einer Edit-Session sind geschützt und verändern einen Netzausschnitt zunächst nur in dieser Session. Erst ein abschliessendes commit() verändert das eigentliche Wissensnetz. Stattdessen verwirft ein cancel() die gemachten Änderungen. In beiden Fällen ist die Edit-Session anschließend ungültig und ein weitere Zugriff auf das Wissensnetz ist nur in einer neuen Session möglich.

Um in der Lage zu sein, eine neue KEMSession zu starten, muss die Anwendung zunächst an einem konkreten Wissensnetz angemeldet sein. Dazu existiert in KEM der KEMUser. Dieser repräsentiert einen der modellierten Wissensnetzbenutzer mit seinen spezifischen Zugriffsrechten (siehe KB-Handbuch/Rechtesystem/Erste Schritte). Von hier aus ist es nun möglich, neue Sessions zu starten.

3 Hello Root

Das erste Beispiel zeigt den Einstieg in und den Ausstieg aus KEM sowie den Zugriff auf den Wurzelbegriff eines Wissensnetzes:

```
KEMManager manager = new SimpleManager("bridge.host", 4713, 4714);
```

```
KEMUser user = manager.newAnonymousUser("DAS-NETZ");
KEMSession session = user.newReadSession();
KEMConceptEditor root = session.getRootConcept();
System.out.println("Der Name der Wurzel ist: " + root.getLabel());
user.logout();
manager.destroy();
```

In der ersten Zeile wird eine KEMRuntime-Instanz erzeugt. Hierzu müssen Hostname und Ports der KEMBridge bekannt sein (im Beispiel wurden die Default-Ports verwendet). Die Ports lassen sich in der bridge.ini der verwendeten Bridge anpassen. Der erste Port bezieht sich auf den Standardkanal, über den die KEM-Bridge angesprochen wird. Der zweite Port bezieht sich auf einen eigenen Streamingkanal. Über diesen werden Binärdaten mit dem Wissensnetz ausgetauscht. Binärdaten tauchen bei KInfinity-Attributen vom Typ Datei auf. Für den Einstieg in ein Wisssensnetz sind die Implementierungen von KEMManager zuständig. Sie sind die oberste Ebene der KEM-Aufrufhierarchie. Über einen KEMManager können z.B. die verfügbaren Volumes abgefragt werden. Um Zugriff auf ein bestimmtes Wissensnetz zu erhalten, muss ein KEMUser am Netz angemeldet werden. Im Beispiel ist dies der anonyme Benutzer (siehe wieder KB-Handbuch/Rechtesystem/Erste Schritte).

In der ersten Zeile wird eine KEMRuntime-Instanz erzeugt. Hierzu müssen Hostname und Ports der KEMBridge bekannt sein (im Beispiel wurden die Default-Ports verwendet). Die Ports lassen sich in der bridge.ini der verwendeten Bridge anpassen. Der erste Port bezieht sich auf den Standardkanal, über den die KEM-Bridge angesprochen wird. Der zweite Port bezieht sich auf einen eigenen Streamingkanal. Über diesen werden Binärdaten mit dem Wissensnetz ausgetauscht. Binärdaten tauchen bei KInfinity-Attributen vom Typ Datei auf. Für den Einstieg in ein Wisssensnetz sind die Implementierungen von KEMManager zuständig. Sie sind die oberste Ebene der KEM-Aufrufhierarchie. Über einen KEMManager können z.B. die verfügbaren Volumes abgefragt werden. Um Zugriff auf ein bestimmtes Wissensnetz zu erhalten, muss ein KEMUser am Netz angemeldet werden. Im Beispiel ist dies der anonyme Benutzer (siehe wieder KB-Handbuch/Rechtesystem/Erste Schritte).

Anschließend wird am KEMUser durch den Aufruf von newReadSession() eine neue Read-Session geöffnet. Erst dadurch wird der Zugriff auf eigentliche Wissensnetzdaten ermöglicht.

Schließlich führt der Aufruf von getRootConcept() zu einem konkreten Wissensnetzobjekt, nämlich einem KEMConceptEditor (der Wurzelbegriff ist ein Wissensnetzobjekt vom Typ Konzept). Dieses Editor-Interface erbt die Methode getLabel() vom obersten aller Editoren, dem KEMEditor, worüber sich bequem der Name eines Objekts, hier also des Wurzelbegriffs, abfragen läßt.

Abschließend sollte der KEM-Zugriff finalisiert werden. Hierzu wird im Beispiel der KEMUser mit logout() abgemeldet und der KEMManager mit destroy() ordentlich beendet. Da hier lediglich eine Read-Session benutzt wurde und im Wissensnetz keine Änderung vorgenommen wurden, bedarf es keinem commit() der Session (was eine KEMReadSession ohnehin nicht kennt).

4 Begriffsinspektion

Dieses Beispiel demonstriert die generische Inspektion eines Begriffs. Diese besteht im Allgemeinen aus der Abfrage aller Begriffsattribute, Relationen, Ober- und Unterbegriffen sowie den Individuen. Der KEM-Einstieg und -Ausstieg von Hello Root wird hier nicht mehr betrachtet. Ausgangspunkt hier ist ein ordentlich angemeldeter KEMUser. Die folgenden Einzelbeispiele lassen sich unmittelbar in gleicher Reihenfolge zu einem lauffähigen Block zusammenkopieren.

4.1 Begriff auffinden

Zuerst geht es darum, einen bestimmten Begriff im Wissensnetz aufzufinden. In diesem Beispiel benötigt der Begriff einen gesetzen internen Namen:

```
KEMSession session = user.newReadSession();
KEMConceptEditor concept = session.getTopicByInternalName("my.unknown.topic");
System.out.println("Name des Begriffs: " + concept.getLabel());
System.out.println();
```

Um einen bestimmten Begriff über seinen internen Namen zu finden, kann die KEMSession mit "getTopicByInternalName()" unter der Angabe des internen Namen danach gefragt werden. Hier im Beispiel wird darauf der Begriff mit dem internen Namen "my.unknown.topic" zurückgeliefert. Anschließend wird der Name des Begriffs ausgegeben.

4.2 Oberbegriff bestimmen

Das nächste Teilbeispiel demonstriert, wie sich die Oberbegriffe des gefundenen Begriffs abfragen lassen:

```
KEMListEditor superConceptListKEM = concept.getSuperConcepts();
java.util.List superConceptList = superConceptListKEM.getItems();
System.out.println("Der Begriff besitzt " + superConceptList.size()
    + " Oberbegriffe:");
java.util.Iterator superConceptIter = superConceptList.iterator();
while (superConceptIter.hasNext()) {
    KEMConceptEditor superConcept = (KEMConceptEditor)
        superConceptIter.next();
    System.out.println(superConcept.getLabel());
}
System.out.println();
```

getSuperConcepts() am KEMConceptEditor liefert einen speziellen KEMListEditor, der die Oberbegriffe enthält. Ein KEMListEditor ist ein generische Klasse, die in KEM die verschiedenartigsten Listen repräsentiert. Dabei gilt stets, dass die Listeneinträge KEMEditor-Objekte sind. In diesem Fall sind es wiederum KEMConceptEditor-Objekte.
Um mit Standard-Java-Mitteln über eine solche Liste zu iterieren, stellt der KEMListeEditor die Methode getItems() bereit, die eine Java-Liste (java.util.List) zurückliefert. Die Einträge dieser Liste sind dann die interessanten KEMConceptEditor-Objekte.

Hier wird nun die Anzahl der Oberbegriffe (also die Größe der Liste) und für jeden Oberbegriff der Name ausgegeben.

4.3 Unterbegriffe bestimmen

Auf die gleiche Art lassen sich auch die Unterbegriffe bestimmen:

```
KEMListEditor subConceptListKEM = concept.getSubConcepts(null,
    null, Boolean.FALSE);
java.util.List subConceptList = subConceptListKEM.getItems();
System.out.println("Der Begriff besitzt " + subConceptList.size()
```

```
    + " Unterbegriffe:");
java.util.Iterator subConceptIter = subConceptList.iterator();
while (subConceptIter.hasNext()) {
    KEMConceptEditor subConcept = (KEMConceptEditor)
        subConceptIter.next();
    System.out.println(subConcept.getLabel());
}
System.out.println();
```

Dieser Block unterscheidet sich nur in der Abfrage des KEMListEditor für die Unterbegriffe, nämlich mittels getSubConcepts(). Die hier verwendeten zusätzlichen Argumente veranlassen KEM dazu, keine Namensfilterung vorzunehmen und auch kein Unterbegriffe der Unterbegriffe (rekursiv) einzuschließen. Eine ausführlichere Beschreibung dazu liefert die KEM-Javadoc.

4.4 Individuen bestimmen

Auch die Individuen des Begriffs werden auf diese Weise abgefragt:

```
KEMListEditor instanceListKEM = concept.getInstances(null,
    null, Boolean.FALSE);
java.util.List instanceList = instanceListKEM.getItems();
System.out.println("Der Begriff besitzt " + instanceList.size()
    + " Individuen:");
java.util.Iterator instanceIter = instanceList.iterator();
while (instanceIter.hasNext()) {
    KEMInstanceEditor instance = (KEMInstanceEditor)
        instanceIter.next();
    System.out.println(instance.getLabel());
}
System.out.println();
```

Wieder findet keine Namensfilterung statt und es werden nur die direkten Individuen des Begriffs erfragt. Im Unterschied zu den obigen KEMListEditor-Objekten sind die Einträge hier vom Typ KEMInstanceEditor. Ansonsten sieht der Rest wieder gleich aus.

4.5 Begriffsattribute bestimmen

Als nächstes werden die Attribute des Begriffs abgefragt:

```
KEMListEditor attListKEM = concept.getAttributes(null,
    null, KEMTopicEditor.EXISTING, KEMTopicEditor.USER);
java.util.List attList = attListKEM.getItems();
System.out.println("Der Begriff besitzt " + attList.size()
    + " Attribute:");
java.util.Iterator attIter = attList.iterator();
while (attIter.hasNext()) {
    KEMAttributeEditor att = (KEMAttributeEditor)
        attIter.next();
    System.out.print(att.getLabel());
```

```
    System.out.print(": ");
    System.out.println(att.getValueString());
}
System.out.println();
```

Um die Attribute eines Begriffs abzufragen, existiert die Methode getAttributes(), welche der KEMConceptEditor vom KEMTopicEditor erbt. Die hier verwendeten Argumente bedeuten zum Einen wieder keine Namensfilterung. Zum Anderen werden nur existierende (KEMTopicEditor.EXISTING) und benutzerdefinierte Attribute (KEMTopicEditor.USER) berücksicht. Würde der EXISTING-Filter fehlen, so würde die Ergebnisliste auch Attribut-Editoren enthalten, die zur Neuanlage eines Attributs dienen (Attribut-Protos). Ein nicht benutzerdefiniertes Attribut ist z.B. das EID-Attribut (Systemattribut). Der so gewonnene KEMListEditor enthält nun Einträge vom Typ KEMAttributeEditor. Im Schleifenkörper werden schließlich die Namen (getLabel()) und die Werte (getValueString()) der Attribute ausgegeben.

4.6 Begriffsrelationen bestimmen

Schließlich fehlen noch die Relationen:

```
KEMListEditor relListKEM = concept.getRelations(null, null,
KEMTopicEditor.EXISTING, KEMTopicEditor.USER, KEMFullTopicEditor.ALL);
java.util.List relList = relListKEM.getItems();
System.out.println("Der Begriff besitzt " + relList.size() + " Relationen:");
java.util.Iterator relIter = relList.iterator();
while (relIter.hasNext()) {
     KEMSimpleRelationEditor rel = (KEMSimpleRelationEditor)
        relIter.next();
     System.out.print(rel.getLabel());
     System.out.print(": ");
     System.out.println(rel.getTarget().getLabel());
}
```

Auch die Relationen werden nicht namentlich gefiltert und es sollen auch wieder nur existierende und benutzerdefinierte Relationen berücksichtigt werden. Systemrelationen sind beispielsweise die ist Unterbegriff von- oder die hat Individuum-Relation. Das zusätzliche Argument KEMFullTopicEditor.ALL schließt sowohl normale als auch Abkürzungsrelationen ein. Die Einträge der Relationsliste sind vom Typ KEMSimpleRelationEditor. Der Relationsname kann wieder über getLabel() abgefragt werden. Um den Namen des Relationsziels auszugeben wird hier der Relationseditor nach getTarget() gefragt, der wiederum seinen Namen durch getLabel() preisgibt.

4.7 Erweiterungen bestimmen

Ein wichtiger Unterschied zwischen Konzeptbegriffen und Individuen besteht in der Fähigkeit der Indiviuden, Erweiterungen annehmen zu können. Nehmen wir das KEMInstanceEditor-Objekt aus dem obigen Teilbeispiel:

```
KEMListEditor extensionListKEM = instance.getExtensions(null, null);
java.util.List extensionList = extensionListKEM.getItems();
System.out.println("Das Invididuum besitzt " + extensionList.size()
    + " Erweiterungen:");
java.util.Iterator extensionIter = extensionList.iterator();
```

```
while (extensionIter.hasNext()) {
     KEMInstanceEditor extension = (KEMInstanceEditor) extensionIter.next();
     System.out.println(extension.getLabel());
}
```

Mit getExtensions(null, null) wird eine ungefilterte Liste der vorhandenen Erweiterungsinstanzen anfordert. Da Erweiterungen typischerweise keinen eigenen Namen besitzen, liefert ein getLabel() an einer solchen Erweiterungsinstanz den Namen des Kernindividuums plus den Namen des Erweiterungskonzepts in Klammern.

5 Verwendung von Suchen

Die folgenden Beispiele zeigen die Abfrage des Wissensnetzes durch die Verwendung bestimmter Suchen. Dabei stehen zwei verschiedene Grundtypen zur Verfügung, die sich von ihrer Arbeitsweise her unterscheiden: Textsuchen und Expertensuchen. Expertensuchen werden über das KEM-Interface KEMExpertQueryFolderEditor repräsentiert, Textsuchen über KEM-SearchEditor.

5.1 Expertensuchen

Das folgende Beispiel zeigt die Ausführung einer Expertensuche über KEM. Wichtig ist, dass die Expertensuche mit dem Knowledge-Builder im Wissensnetz angelegt und mit einer externen ID versehen ist (siehe Knowledge-Builder-Handbuch). Nur Expertensuchen mit externer ID können über KEM angesprochen werden:

```
KEMReadSession readSession = kemUser.newReadSession();
KEMExpertQueryFolderEditor kemExpertQuery = (KEMExpertQueryFolderEditor)
    readSession.getFolder("notizMitAroma");
KEMStringParameterEditor kemAromaNameParam = (KEMStringParameterEditor)
kemExpertQuery.getParam("aromaname");
kemAromaNameParam.setValue("ingwer");
System.out.println("\n");
List<?> topicList = kemExpertQuery.getItems();
for (int i=0; i<topicList.size(); i++) {
    Object o = topicList.get(i);
    if (o instanceof KEMTopicEditor) {
        KEMTopicEditor kemTopic = (KEMTopicEditor) o;
        System.out.print(kemTopic.getLabel()+" (");
        KEMListEditor kemExplainTopics = kemExpertQuery.explainItem(i);
        List<?> explainTopics = kemExplainTopics.getItems();
        boolean isFirst = true;
        for (Object o2 : explainTopics) {
            if (o2 instanceof KEMTopicEditor) {
                KEMTopicEditor explainTopic = (KEMTopicEditor) o2;
                if (isFirst) {
                    isFirst = false;
                } else {
                    System.out.print(", ");
                }
                System.out.print(explainTopic.getLabel());
            }
```

```
        }
        System.out.print(")\n");
    }
}
System.out.println("\n");
```

Die Ausführung einer Expertensuche über KEM erfolgt normalerweise in drei Schritten:

1. Erzeugen eines KEMExpertQueryFolderEditor über die Methode "getFolder()" an der KEMSession

2. Setzen von Suchparametern am KEMExpertQueryFolderEditor

3. Abfragen der Treffermenge über die Methode getItems() bzw. Abfragen von Eigenschaften der Treffer über die Methode getTable()

In dem Beispiel wird ein KEMExpertQueryFolderEditor für die Expertensuche mit der external ID "notizMitAroma" erzeugt. Diesen Namen kann man zum Ausprobieren von selbstdefinierten Suchen einfach abändern.

Schritt 2 ist nur dann notwendig, wenn die Expertensuche parametrisierbar ist (siehe dazu Knowledge-Builder-Handbuch / Expertensuchen). Wie im Beispiel zu sehen, muss zum Setzen eines Parameters zunächst die Methode getParam("parametername") am Expertensuch-Editor aufgerufen werden, um einen KEMParameterEditor zu erzeugen. Danach kann man am KEMParameterEditor die Methode setValue() zum Setzen eines Werts aufrufen.

In Schritt 1-2 wird noch keine Treffermenge für die Expertensuche berechnet. Dies geschieht erst dann, wenn eine der Methoden getItems(), getValues(), getValuesFromTo(), getSize() oder getTableFromMapping() aufgerufen wird. Die Treffermenge wird natürlich nur dann berechnet, wenn zum ersten Mal eine dieser Methoden aufgerufen wird oder wenn zwischenzeitlich noch einmal Schritt 2 ausgeführt wurde.

Der KEMExpertQueryFolderEditor kann auch Informationen darüber liefern, wie ein Treffer in der Treffermenge zustande kam. Hierzu kann die Methode explainItem() mit einem Trefferindex als Argument aufgerufen werden. Als Ergebnis dieses Aufrufes wird eine Liste aller Begriffe und Instanzen aus dem Wissensnetz zurückgeliefert, die beim Zustandekommen des Treffers beteiligt waren.

5.2 Textsuchen

Textsuchen werden pro Wissensnetz im KnowledgeBuilder konfiguriert (siehe dazu KB-Handbuch/Suchen). Wie der Name schon sagt, arbeiten die Textsuchen basierend auf einem Suchtext (im einfachsten Fall ein Wort), und liefern eine Ergebnismenge. Die Struktur der Ergebnismenge kann dabei eine einfache Liste sein oder aber eine hierarchische Struktur bestehend aus Teilergebnisordnern, die ihrerseits noch bestimmte Eigenschaften aufweisen.
Das folgende Beispiel zeigt eine Abfrage einer Textsuche. Die Suche muss hier den Namen simpleSearch besitzen (siehe KB-Handbuch/Suchen/Einfache Suche). Auch hier ist des KEMUser-Objekt aus dem Hello Root vorbereitet:

```
KEMSession session = user.newReadSession();
String searchString = "dies und das";
KEMSearchEditor searchEditor = session.newSearch("simpleSearch",searchString);
```

```
// Hier können weitere Parameter gesetzt werden
// Für zusammenstellbare Suchen
// HashMap<String,Object> variables = new HashMap<String, Object>();
// variables.put("Begriffsname", "Modell");
// Parameter "variables" von der Suche holen und als Wert die HashMap mit den
// Variablen setzen
// KEMDictionaryParameterEditor variablesParam = (KEMDictionaryParameterEditor)
//     search.getParam("variables");
// variablesParam.setValue(variables);

System.out.println("Die Suche nach \""+searchString+"\" ergab "
    + queryFolder.getResultSize()+"Treffer und diese sind:\n");
List<?> orderCriteria = new ArrayList<?>();
orderCriteria.add(new OrderByParameter("quality()",false));
orderCriteria.add(new OrderByParameter("label()",true));
searchEditor.setOrderBy(orderCriteria);
List<?> hitProperties = new ArrayList<?>();
hitProperties.add("label()");
hitProperties.add("quality()");
List<?> resultTable = kemSearch.getResultTable(hitProperties));
for (Object o : resultTable) {
    if (o instanceof List<?>) {
         List<?> colValues = (List<?>) o;
         System.out.print("Name: "+ colValues.get(0));
         Double quality = (Double) colValues.get(1);
         System.out.println(" ("+(quality*100)+"%)");
    }
}
```

Die Ausführung einer Textsuche läuft normalerweise in den folgenden Schritten ab:

1. Erzeugen eines KEMSearchEditor über die Methode newSearch() am KEMSession-Editor

2. Setzen von Suchparameter über die Methode getParam() am KEMSearchEditor

3. Setzen von Sortierkriterien anhand von KPath-Ausdrücken. Hierzu muss die Methode setOrderBy() am KEMSearchEditor mit einer Liste von Objekten der Klasse com.iviews.kem.OrderByParame Argument aufgerufen werden. Näheres dazu finden Sie in der KEM-JavaDoc. Eine ausführliche Beschreibung von KPath finden Sie im KPath-Referenz-Handbuch.

4. Setzen einer Expertensuche zum Filtern der Textsuche über die Methode setFilterWithExpertSearch() am KEMSearchEditor, der ein KEMExpertQueryFolderEditor als Argument übergeben werden muss. Näheres zur Erzeugung und Parametrisierung von Expertensuche-Editoren finden Sie im vorangehenden Kapitel "Verwendung von Expertensuchen"

5. Abfragen der Treffermenge über die Methoden getResultTable() bzw. getResultTableFromTo(). Diesen Methoden muss wiederum eine Liste mit KPath-Ausdrücken übergeben werden, die auf jeden Treffer der Suche angewendet werden und deren Ergebnisse als Wertetabelle (geschachtelte Java-List-Objekte) zurückgeliefert wird.

6 Pflege

6.1 Instanzen erzeugen

Die Erzeugung eines neuen Individuum läuft normalerweise in den folgenden Schritten ab:

1. Erzeugen einer neuen KEMEditSession über den Aufruf newSession() am KEMUser-Objekt

2. Erzeugen eines KEMConceptEditor, der das individuenfähige Konzept im Wissensnetz repräsentiert, von dem ein Individuum (Instanz) angelegt werden soll. Dies geschieht normalerweise über eine Suche oder über eine Methode an der KEMSession (z.B. getTopicByInternalName). Hat der Konzeptbegriff einen internen Namen, so ist er über KEM leichter auffindbar.

3. Aufruf der Methode newInstance() am KEMConceptEditor. Dieser Aufruf liefert einen KEMInstanceEditor zurück, der das neu angelegte Individuum repräsentiert.

4. Setzen des Namens der neu angelegten Instanz über den Aufruf getNameAttribute().setValue(...) am KEMInstanceEditor, damit die neu angelegte Instanz wiederauffindbar ist

5. Aufruf der Methode commit() zum Abschließen der EditSession.

Das folgende Beispiel erzeugt ein neues Individuum vom Konzept mit dem internen Namen "notice" (der interne Name von Konzeptbegriffen kann mit dem Knowledge-Builder gesetzt werden):

```
KEMEditSession editSession = kemUser.newSession();
KEMConceptEditor kemConcept = editSession.getTopicByInternalName("notice");
KEMInstanceEditor kemInstanceEditor = kemConcept.newInstance();
String instanceName = "Neue Notiz";
kemInstanceEditor.getNameAttribute().setValue(instanceName);
editSession.commit();
```

6.2 Attribute erzeugen

Voraussetzung zur Erzeugung eines neuen Attributes ist, dass dieses mit dem Knowledge-Builder in der Schemadefinition für Individuen oder Unterbegriffe des jeweiligen Begriffs angelegt wurde. Es vereinfacht die Erzeugung neuer Attribute über KEM, wenn bei der Definition im Knowledge-Builder ein interner Name vergeben wurde.

Die Erzeugung eines neuen Attributes über KEM läuft normalerweise in den folgenden Schritte ab:

1. Erzeugung einer KEMEditSession durch Aufruf der Methode newSession() am KEMUser-Objeket
2. Auswahl der Instanz oder des Konzepts an dem das neue Attribut angebracht werden soll. Hierzu muss über eine Suche oder über die Methoden der KEMSession der KEMInstanceEditor oder KEMConceptEditor beschafft werden, der das Konzept/die Instanz repräsentiert.

3. Aufruf der Methode getAttribute() bzw. getAttributes() am ausgewählten Objekt. Die Aufrufe liefern entweder direkt einen KEMAttributeEditor zurück, der das neu zu erzeugende Attribut repräsentiert oder eine KEMListEditor, der eine Liste von möglichen Attributen repräsentiert.
4. Aufruf der Methode setValue() oder setValueString() an einem KEMAttributeEditor. Dieser Aufruf erzeugt das Attribut.
5. Abschließen der KEMEditSession durch Aufruf der Methode commit() am KEMEditSessionEditor.

Hierzu ein Beispiel:

```
KEMEditSession editSession = kemUser.newSession();
KEMConceptEditor kemConcept = editSession.getTopicByInternalName("notice");
KEMInstanceEditor kemInstanceEditor = kemConcept.newInstance();
kemInstanceEditor.getNameAttribute().setValue("Neue Notiz");
KEMListEditor kemAttList = kemInstanceEditor.getAttributes(null,
    null, ExistingOrPossibleFilter.POSSIBLE, null);
for (Object o : kemAttList.getItems()) {
     if (o instanceof KEMAttributeEditor) {
          KEMAttributeEditor att = (KEMAttributeEditor) o;
          System.out.println(att.getLabel());
     }
}
kemAttList = kemInstanceEditor.getAttributes("description",
    null, ExistingOrPossibleFilter.POSSIBLE, null);
KEMStringAttributeEditor kemDescriptionAtt = (KEMStringAttributeEditor)
    kemAttList.getItem(0);
kemDescriptionAtt.setValue("Dies ist die Beschreibung zu einer neuen Notiz");
editSession.commit();
```

Im Beispiel wird zunächst ein KEMConceptEditor beschafft, der das individuenfähige Konzept "Notiz" im Wissensnetz repräsentiert. Danach wird ein neues Notiz-Individuum erzeugt. Danach wird mit dem Aufruf von getAttributes() eine Liste aller an dem Individuum möglichen Attribute abgefragt und ausgegeben. Anschließend wird eine Liste mit möglichen Attributen abgefragt, für die der interne Name "description" vergeben ist. Da diese Liste nur ein (oder kein) Attribut enthalten kann, wird der erste KEMAttributeEditor in der Liste ausgewählt und an diesem dann der Attributwert gesetzt. Somit ist ein neues Attribut "description" mit dem angegebenen Wert an dem Individuum "Neue Notiz" angelegt. Am Ende wird die Transaktion mit dem Aufruf von commit() abgeschlossen.

HINWEIS: In K-Infinity 3 ist es auch möglich, Meta-Attribute an Attributen anzulegen. Dies geschieht im Grund genauso wie oben beschrieben, die Methoden getAttribute() bzw. getAttributes() müssen dazu einfach an einem KEMAttributeEditor aufgerufen werden.

6.3 Relationen erzeugen

Anlegen von Relationen erfolgt in KEM im Grunde genauso wie das Anlegen von Attributen:

1. Erzeugung einer KEMEditSession durch Aufruf der Methode newSession() am KEMUser-Objekt
2. Auswahl der Instanz oder des Konzepts von dem aus die neue Relation gezogen werden soll. Hierzu muss über eine Suche oder über die Methoden der KEMSession der

KEMInstanceEditor oder KEMConceptEditor beschafft werden, der das Konzept/die Instanz repräsentiert.

3. Aufruf der Methode getRelation() bzw. getRelations() am ausgewählten Objekt mit dem Parameter POSSIBLE. Die Aufrufe liefern entweder direkt einen KEMRelationEditor zurück, der die neu zu erzeugende Relation repräsentiert oder einen KEMListEditor, der eine Liste von möglichen Relationen repräsentiert.
4. Auswahl des Zielobjektes der neu anzulegenden Relation durch Aufruf der Methoden setTargetEditor(), setNewTarget() oder getPossibleTargets() am KEMRelationEditor (nähere Informationen dazu in der Javadoc von KEMRelationEditor). Hiermit wird die Relation angelegt.
5. Abschließen der KEMEditSession durch Aufruf der Methode commit() am KEMEditSessionEditor.

Hier ein Beispiel:

```
KEMConceptEditor kemConcept = editSession.getTopicByInternalName("notice");
KEMInstanceEditor kemInstanceEditor = kemConcept.newInstance();
kemInstanceEditor.getNameAttribute().setValue("Neue Notiz ");
KEMListEditor kemRelList = kemInstanceEditor.getRelations(null,
    null,
    ExistingOrPossibleFilter.POSSIBLE,
    null,
    RelationTypeFilter.WITHOUT_SHORTCUT);
for (Object o : kemRelList.getItems()) {
    if (o instanceof KEMRelationEditor) {
        KEMRelationEditor rel = (KEMRelationEditor) o;
        System.out.println(rel.getLabel());
    }
}
KEMRelationEditor kemHasAuthorRel = (KEMRelationEditor) kemInstanceEditor
    .getRelation(
           "hatAutor",
           null,
           ExistingOrPossibleFilter.POSSIBLE,
           null,
           RelationTypeFilter.WITHOUT_SHORTCUT);
kemHasAuthorRel.setTargetEditor(editSession.getUserTopic());
```

Wie im Beispiel zuvor wird hier zunächst ein neues Individuum von Notiz mit dem Namen "Neue Notiz" angelegt. Danach wird eine Liste der an diesem Objekt möglichen Relationen ausgegeben und anschließend eine neue Relation vom Typ "hatAutor" zu dem Objekt im Wissensnetz gezogen, das den aktuellen Benutzer repräsentiert. Danach wird die Transaktion abgeschlossen.

Alternativ zu dem oben beschriebenen Weg ist es in KEM 3 auch möglich, neue Relationen mittels des KEMNewRelationAgentEditor anzulegen. Dieser kann über die Methode getNewRelationAgent() an allen KEM-Editoren aufgerufen werden, die von KEMTopicEditor erben (also an Instanzen, Konzepten, Attributen und Relationen) und bietet eine Reihe erweiterter Funktionen (z.B. Kontrolle über die erzeugte inverse Relation) und ermöglicht mehr Interaktivität bei der Relationserzeugung (z.B. Rückfragen an den Benutzer im Falle von mehreren möglichen inversen Relationen).

HINWEIS: Wie auch bei Attributen ist es in K-Infinity 3 möglich, an Relationen neue Attribute und sogar Relationen anzulegen. Hierzu ruft man einfach getAttribute(), getAttributes(), getRelation() oder getRelations() an einem KEMRelationEditor auf.

7 Effizienzsteigerung

7.1 Verwendung mehrerer KEM-Bridges

Im "Hello Root" Beispiel wurde ein SimpleManager verwendet. Dieser kann mit genau einer KEM-Bridge umgehen, d.h. alle Aufrufe (mit Ausnahme von Streaming) werden über eine TCP/IP Verbindung geschickt. Durch den Einsatz eines BalancedManagers und mehrerer KEM-Bridges kann die Performance der Aufrufe erhöht werden. Eine Session ist immer fest an eine KEM-Bridge gebunden, d.h. alle Aufrufe innerhalb dieser Sessions werden von der bei der Initialisierung festgelegten KEM-Bridge verarbeitet. Der BalancedManager entscheidet beim Erzeugen einer neuen Session auf welcher KEM-Bridge diese initialisiert werden sollte. Hierfür kann der BalancedManager mit mehreren KEM-Bridges konfiguriert werden:

```
KEMManagerConfig managerConfig = new KEMManagerConfig();
managerConfig.addKEMBridge("host1", 4713, KEMBridgeConfig.READ);
managerConfig.addKEMBridge("host2", 4713, KEMBridgeConfig.READ);
managerConfig.addKEMBridge("host1", 5713, KEMBridgeConfig.EDIT);
managerConfig.addKEMBridge("host2", 5713, KEMBridgeConfig.EDIT);
managerConfig.addKEMBridge("host3", 6713, KEMBridgeConfig.MIXED);
managerConfig.addStreamingBridge("host3", 6715);
BalancedManager manager = new BalancedManager(managerConfig);
```

In diesem Beispiel wird ein BalancedManager mit 2 Readbridges (host1:4713 und host2:4713), 2 Edit-Bridges (host1:5713 und host2:5713), einer Mixedbridge (host3:6713) und einer Streamingbridge (host3:6715) erzeugt. Beim hinzufügen einer KEM-Bridge kann noch der Typ der Session angegeben werden, für die die Bridge zuständig ist:

- READ Nur für ReadSessions verwenden
- EDIT Nur für EditSessions verwenden
- MIXED Sowohl für Read- als auch für Editsessions verwenden