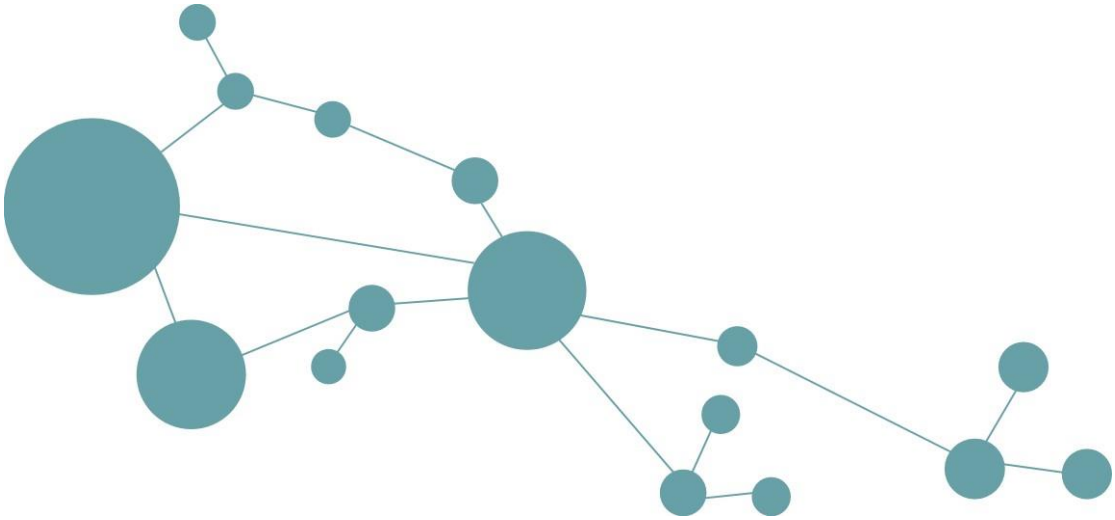


i-views 5.0

i·views

Inhaltsverzeichnis

1 Knowledge-Builder

1.1 Grundlagen

Mit i-views funktionieren Datenbanken so, wie Menschen denken: einfach, agil, flexibel. Deswegen ist in i-views einiges anders als bei relationalen Datenbanken: Wir arbeiten nicht mit Tabellen und Keys, sondern mit Objekten und Beziehungen zwischen ihnen. Die Modellierung der Daten ist visuell und beispielorientiert, so dass wir sie auch mit den Nutzern aus den Fachabteilungen teilen können.

Wir bauen mit i-views keine reinen Datenspeicher, sondern intelligente Datennetze, die bereits viel Business-Logik enthalten und mit denen das Verhalten unserer Anwendung schon weitgehend bestimmt werden kann. Dazu nutzen wir Vererbung, Mechanismen zum Schlussfolgern und zur Definition von Sichten, sowie eine Vielzahl von Suchverfahren, die i-views bietet.

Unser zentrales Werkzeug dazu ist der Knowledge-Builder, eine der Kernkomponenten von i-views. Mit dem Knowledge-Builder können wir:

- Schema definieren, aber auch Beispiele aufbauen und vor allem visualisieren
- Importe und Abbildungen einer Datenquelle definieren
- Abfragen formulieren, vernetzte Daten traversieren, Strings verarbeiten und Nähen berechnen
- Rechte, Trigger und Sichten definieren

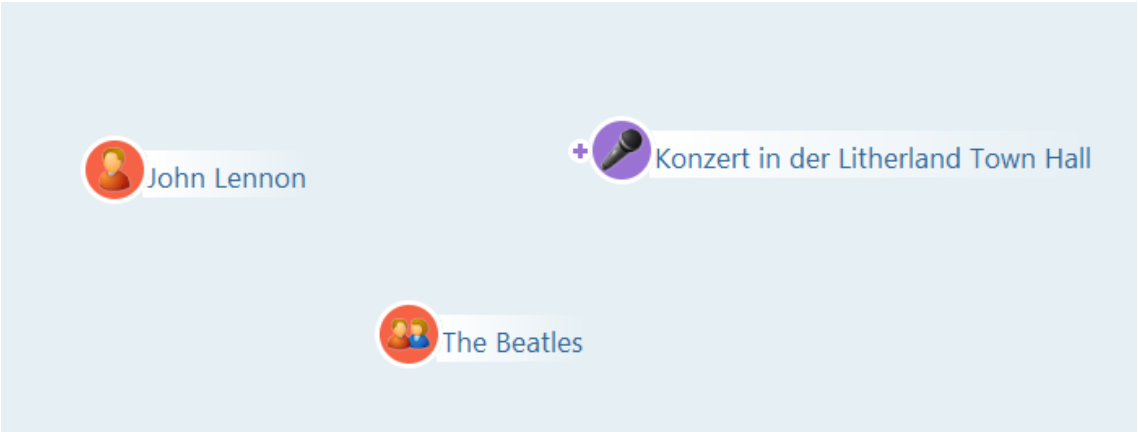
Alle diese Funktionen sind Gegenstand dieser Dokumentation. Als durchgehendes Beispiel dient dabei ein semantisches Netz rund um Musik, Bands, Songs etc.

1.1.1 Grundbausteine

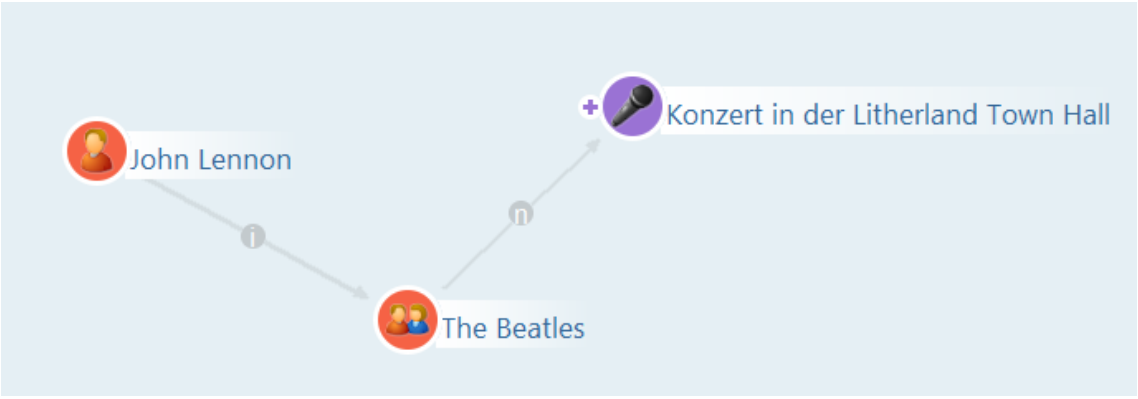
Die Grundbausteine der Modellierung in i-views sind:

- konkrete Objekte
- Beziehungen
- Attribute
- Objekttypen
- Beziehungstypen
- Attributtypen

Konkrete Objekte: Beispiele für konkrete Objekte sind John Lennon, die Beatles, Liverpool, das Konzert in der Litherland Town Hall, die Fußball-WM 1970 in Mexiko, der schiefe Turm von Pisa etc.:



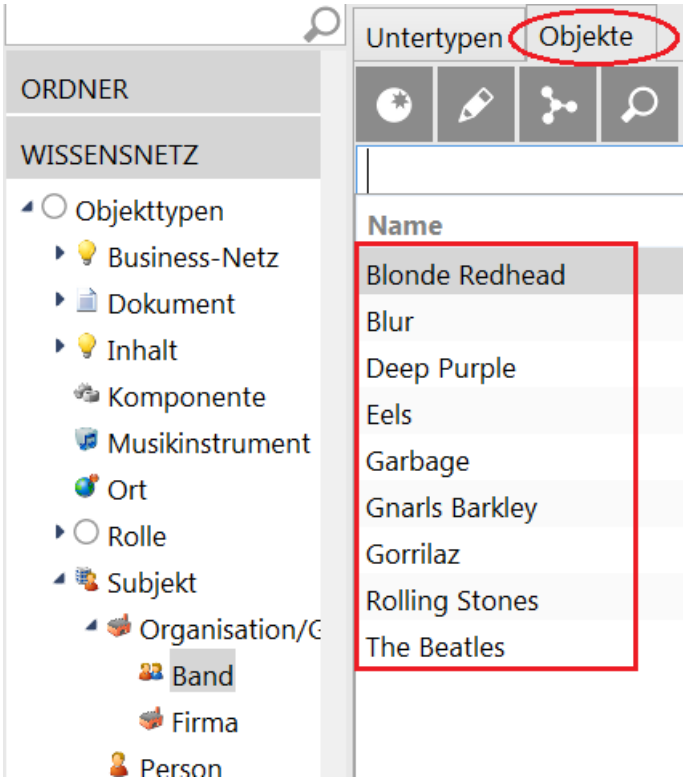
Beziehungen (werden in i-views als Relationen bezeichnet): Diese konkreten Objekte können wir durch Beziehungen miteinander verbinden: „John Lennon ist Mitglied der Beatles“, „Die Beatles treten auf bei einem Konzert in der Litherland Town Hall“.



Attribute: Konkrete Objekte können Attribute haben. Konkrete Attribute von John Lennon sind beisielsweise sein Vorname "John", sein Nachname "Lennon" und sein Geburtsdatum, der 9. Oktober 1940.

Objekttypen: Nebenbei haben wir hier Objekttypen eingeführt. Konkrete Objekte haben immer einen Objekttyp, z.B. den Typ der Personen, den der Städte, der Veranstaltungen oder der Bands - Objekttypen, die Sie in Ihrem Datenmodell frei definieren können.

Ein konkretes Objekt ist dann einem Objekttyp zugehörig, wenn man eine "ist ein"-Beziehung ziehen kann. Die "ist ein"-Beziehung ist gleichbedeutend mit "hat Typ"-Beziehung; in i-views heißt die Beziehung standardmäßig "ist konkretes Objekt von". Beispiele: John Lennon (konkretes Objekt) "ist eine" Person (Objekttyp), The Beatles (konkretes Objekt) "ist eine" Band (Objekttyp) und Konzert in Litherland Town Hall (konkretes Objekt) "ist ein" Konzert (Objekttyp).



Das Hauptfenster von i-views: Links die Objekttypen, rechts die dazugehörigen konkreten Objekte - Hier sehen wir auch: Die Typen der i-views-Netze stehen in einer Hierarchie. Mehr zur Typenhierarchie erfahren Sie im nächsten Abschnitt Typenhierarchie - Vererbung.

Beziehungstypen (auch Relationstypen genannt): Auch die Beziehungen haben unterschiedliche Typen. Zwischen John Lennon und den Beatles gibt es die Beziehung „ist Mitglied von“; zwischen den Beatles und ihrem Konzert kann die Beziehung „treten auf“ heißen - wenn wir etwas mehr verallgemeinern wollen, ist vielleicht „nimmt Teil an“ ein sinnvoller Relationstyp.

Attributtypen: Neben konkreten Objekten und Beziehungen können auch Attribute Typen haben. Im Fall einer Person können dies der Name oder das Geburtsdatum sein. Konkrete Personen (Objekte des Typs 'Person') können dann Namen, Geburtsdatum, Geburts- und Wohnorte, Augenfarbe etc. als Attributtypen haben. Veranstaltungen können einen Ort und eine Zeitspanne haben. Attribute und Relationen werden immer beim Objekt selbst definiert.

1.1.2 Typhierarchie - Vererbung

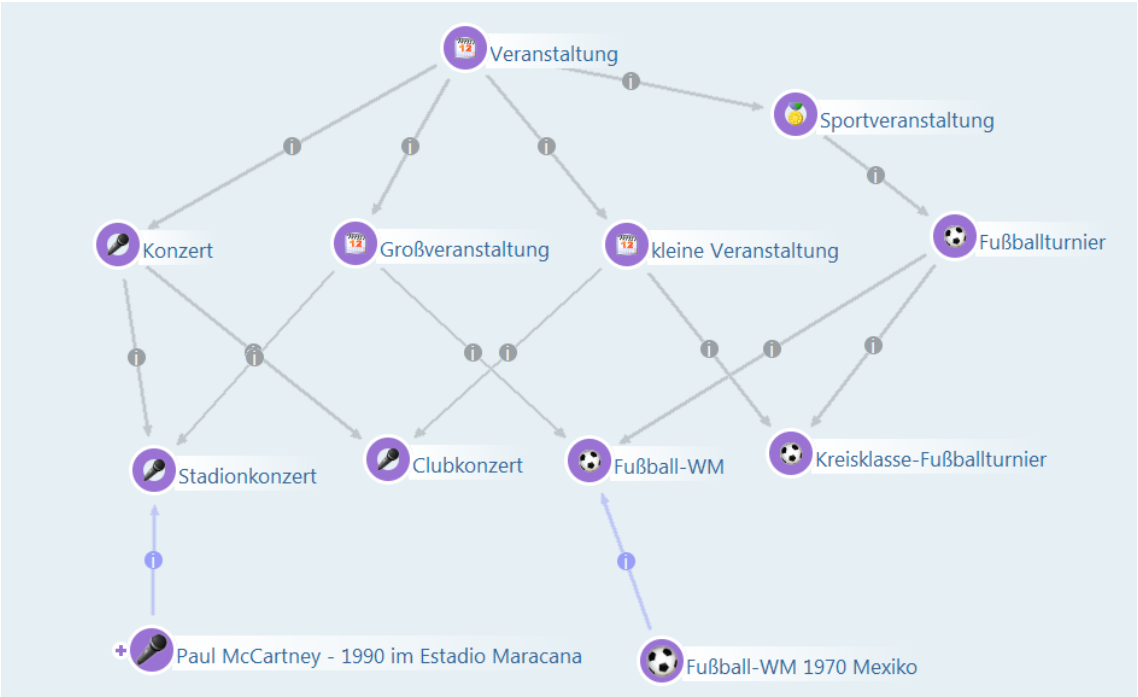
Objekttypen können wir feiner oder weniger fein einteilen: Wir können die Fußball-WM 1970 mit allen anderen Veranstaltungen (die Buchmesse 2015, das Woodstock-Festival...) in einen Topf werfen, dann haben wir nur einen Typ namens „Veranstaltung“ oder wir unterscheiden zwischen Sportveranstaltungen, Messen, Ausstellungen, Musikveranstaltungen etc. Alle diese Typen von Veranstaltungen können wir natürlich auch noch feiner unterteilen: Sportveranstaltungen können z.B. nach Sportarten unterschieden werden (ein Fußballspiel, ein Basketballspiel, ein Fahrradrennen, ein Boxkampf).

So bekommen wir eine Hierarchie von Ober- und Untertypen:

Die Hierarchie ist transitiv: wenn wir i-views nach allen Veranstaltungen fragen, werden nicht nur alle konkreten Objekte angezeigt, die direkt am Objekttyp *Veranstaltung* hängen, sondern auch alle Sportveranstaltungen und alle Radrennen, Boxkämpfe und Fußballspiele. Da der Typ „Boxkampf“ also nicht nur ein Untertyp von „Sportveranstaltung“ sondern damit auch ein Untertyp von „Veranstaltung“ ist, wird i-views eine direkte Ober-/Untertyp-Relation zwischen Veranstaltung und Boxkampf ablehnen - mit dem Hinweis, dass dieser Zusammenhang bereits bekannt ist.

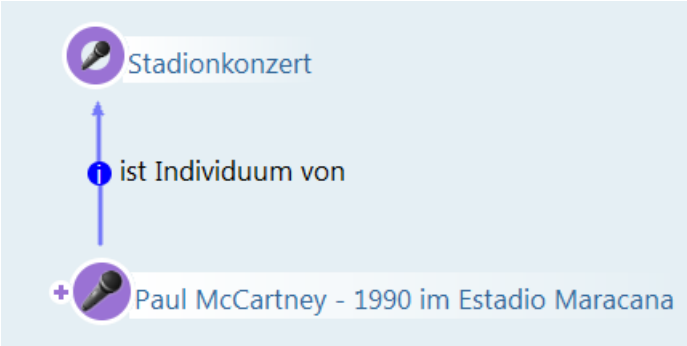
Die hierarchische Struktur muss nicht zwangsläufig die Struktur eines Baumes haben, in der jeder Typ nur genau einen Obertyp haben kann. In einer semantischen Graphdatenbank kann ein Objekttyp mehrere Obertypen haben. Ein konkretes Objekt hingegen kann aber nur genau einen Objekttyp haben.

Am Beispiel des konkreten Konzerts von Paul McCartney 1990, das sowohl ein Konzert, als auch eine Großveranstaltung ist, kann man sehen, was das bedeutet. Da das konkrete Konzert nicht zwei Objekttypen haben kann, wird ein neuer Objekttyp benötigt, der die Aspekte Konzert und Großveranstaltung zusammenführt, hier "Stadionkonzert":

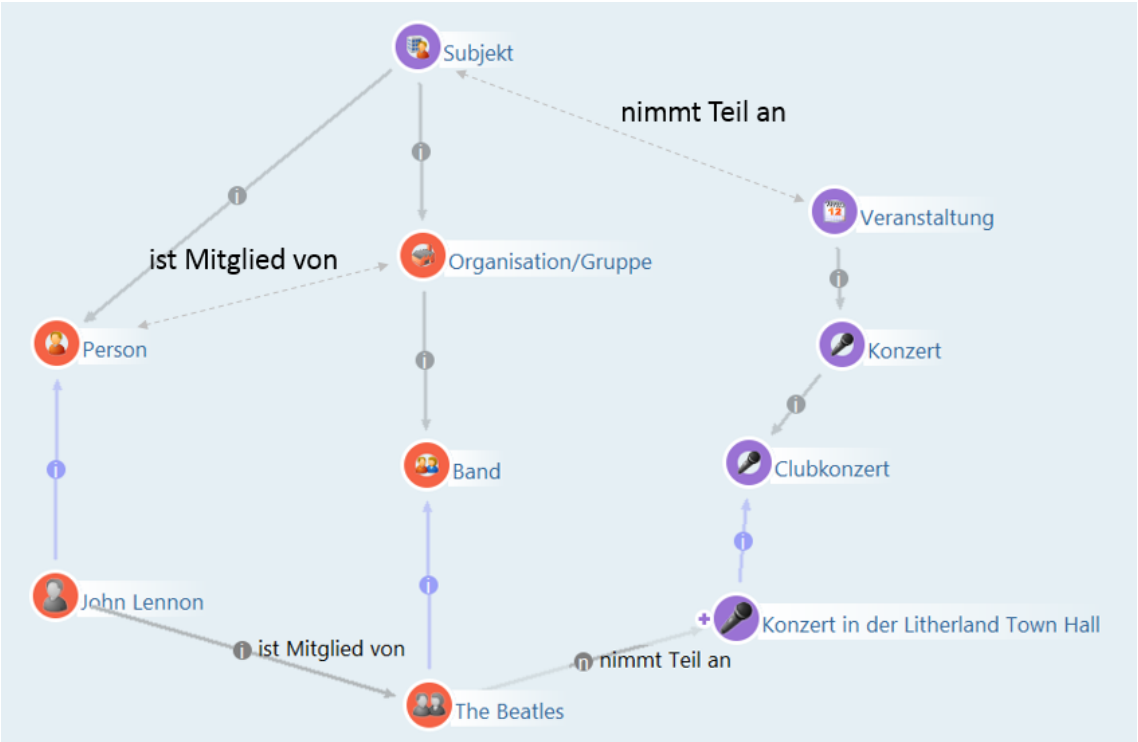


Typenhierarchie mit Mehrfachvererbung

Die Zugehörigkeit von konkreten Objekten zu einem Objekttyp wird in i-views ebenfalls als Relation ausgedrückt und kann als solche abgefragt werden:



Wann unterscheiden wir überhaupt Typen? Typen unterscheiden sich nicht nur ggf. in Icon und Farbe - bei den Objekttypen werden auch die Eigenschaften definiert und nach Typen kann bei Abfragen ganz einfach gefiltert werden. In allen diesen Fragen spielt die Vererbung eine wichtige Rolle: Eigenschaften vererben sich, auch Eigenschaften, welche die Darstellung im Knowledge-Builder beeinflussen wie Icons und Farben vererben sich. Und wenn wir in Abfragen sagen, dass wir Veranstaltungen sehen wollen, dann werden auch alle Objekte der Untertypen als Ergebnis angezeigt.

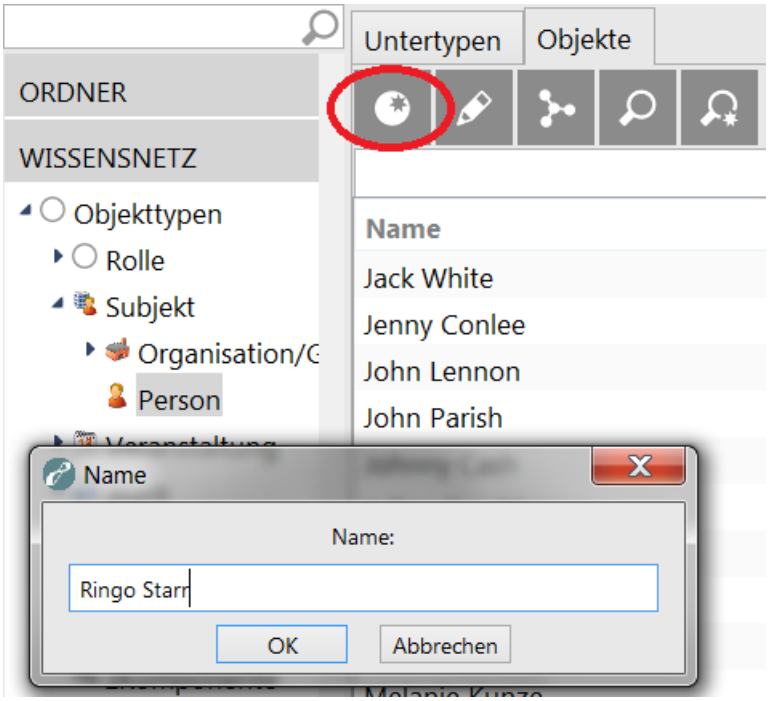


Vererbung macht es möglich, Beziehungstypen (und Attributtypen) weiter oben in der Objekttypen-Hierarchie zu definieren und damit für verschiedene Typen von Objekten (z.B. für Bands und andere Organisationen) zu nutzen.

1.1.3 Objekte anlegen und bearbeiten

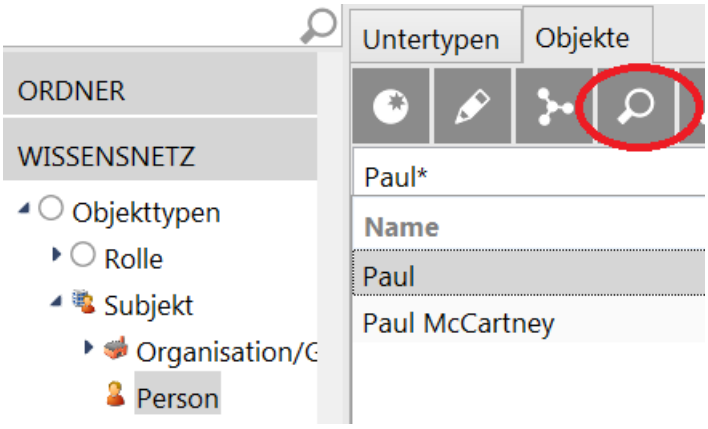
Anlegen von konkreten Objekten

Konkrete Objekte lassen sich im Knowledge-Builder überall dort anlegen, wo Objekttypen zu sehen sind. Ausgehend von den Objekttypen lassen sich über Kontextmenüs die Objekte neu anlegen.



Über die Schaltfläche "Neu" kann ein neues Objekt angelegt werden. Lediglich der Name des Objektes muss zunächst angegeben werden.

Im Hauptfenster befindet sich unter der Kopfzeile die Liste mit bereits vorhandenen konkreten Objekten. Damit Objekte nicht versehentlich doppelt angelegt werden, lässt sich über das Suchfeld in der Kopfzeile der Name des Objektes suchen. Die Suche unterscheidet per Default nicht zwischen Groß- und Kleinschreibung und der Suchbegriff lässt sich links und rechts abschneiden (durch Platzhalter „*“ und „?“ ergänzen):



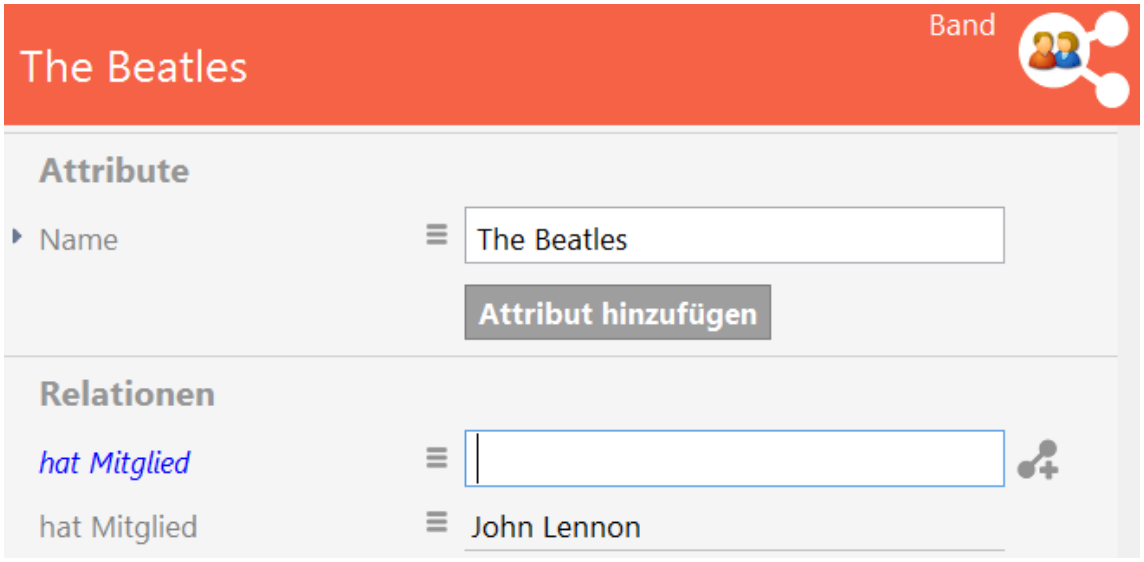
Die Suche nach "Paul" zeigt uns, dass es bereits 2 Personen mit dem Namen "Paul" gibt.*

Bearbeiten von Objekten

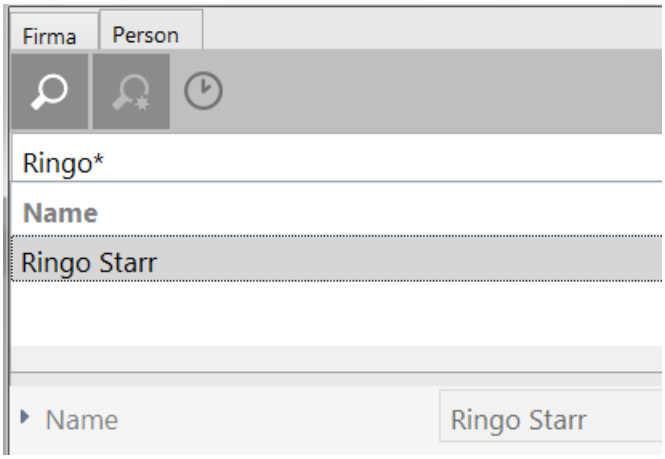
Nach Eingabe und Bestätigung des Objektnamens können im Editor weitere Details für das angelegte Objekt eingeben werden. Dem Objekt lassen sich über Schaltflächen Attribute, Relationen und Erweiterungen zuweisen.

Bei der Bearbeitung eines Objekts können wir neben der Verknüpfung mit einem anderen Objekt gleichzeitig auch das Ziel der Verknüpfung neu erzeugen, sofern es noch nicht existiert.

Beispielsweise sollen Mitglieder einer Musikgruppe vollständig erfasst werden. Über die Relation *hat Mitglied* soll das Zielobjekt Ringo Starr mit dem Objekt „The Beatles“ verknüpft werden. Falls noch nicht bekannt ist, ob das Objekt Ringo Starr schon in i-views erfasst ist, kann im Eingabefeld gesucht werden. Gibt man also "Ringo Starr" ein und bestätigt die Eingabe, und es ist noch kein Objekt mit diesem Namen vorhanden, so öffnet sich ein Dialog, der fragt, ob man ein neues Objekt mit diesem Namen anlegen möchte. Sollte es mehrere Ringo Starrs geben, öffnet sich ein Auswahlfenster.



Man hat zudem die Möglichkeit über die Schaltfläche *Relationsziel wählen* aus einer durchsuchbaren Liste mit allen möglichen Relationszielen ein Objekt auszuwählen.



Das Löschen der Relation *hat Mitglied* kann auf zwei Arten vorgenommen werden:

1. Im Kontextmenü unter der Schaltfläche *Weitere Aktionen* mit der Option *Löschen*.
2. Mit dem Cursor über Schaltfläche *Weitere Aktionen* bei gedrückter Strg-Taste.

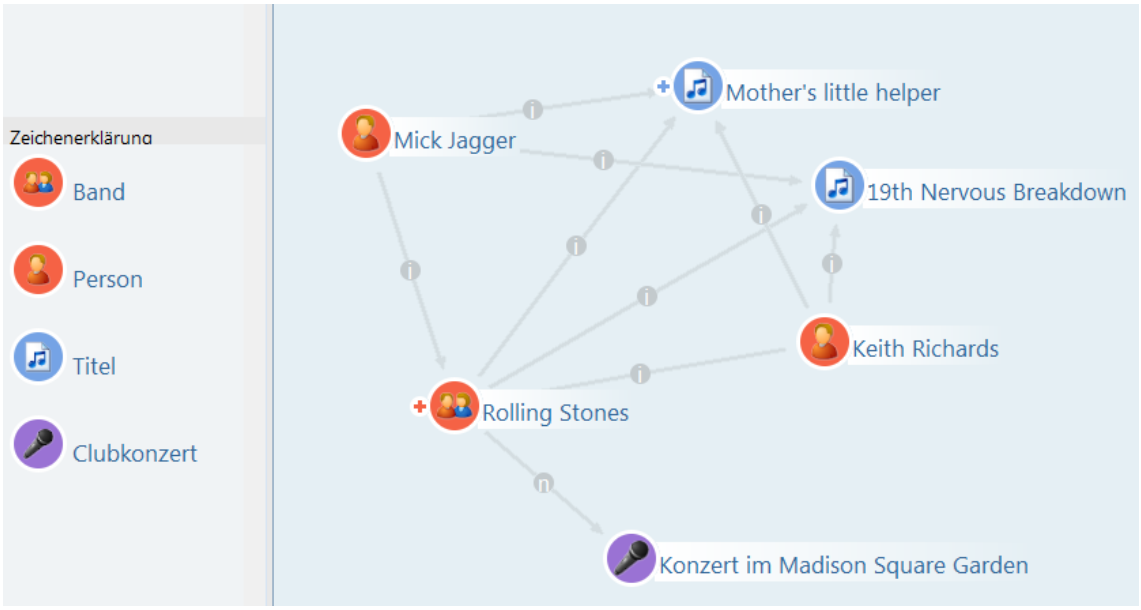
Das Zielobjekt der Relation selbst ist damit jedoch nicht gelöscht. Soll ein Objekt gelöscht werden, so geht das mit der Schaltfläche im Hauptfenster oder über das Kontextmenü direkt auf diesem Objekt.

Objekte können auch über den Graph-Editor angelegt werden. Das Vorgehen hierzu wird im den nachfolgenden Kapitel beschrieben.

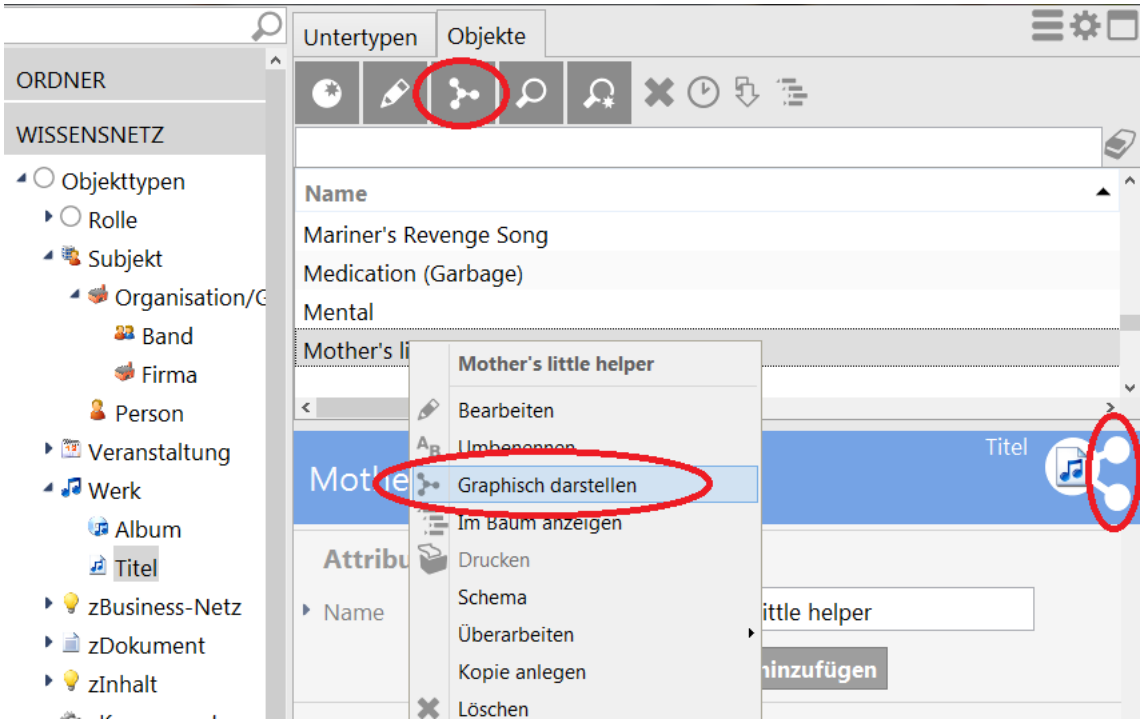
1.1.4 Graph-Editor

1.1.4.1 Einführung Graph-Editor

Mit dem Graph-Editor lassen sich die Inhalte der semantischen Graphdatenbank mit ihren Objekten und Verbindungen graphisch darstellen. Die Objekte werden als Knoten dargestellt, ihre Beziehungen untereinander als Pfeile:



Zugang zum Graph-Editor: Mit der Schaltfläche *Graph* kann der Graph-Editor auf einem selektierten Objekt geöffnet werden. Diese Schaltfläche findet sich im Knowledge-Builder an mehreren Stellen wieder:



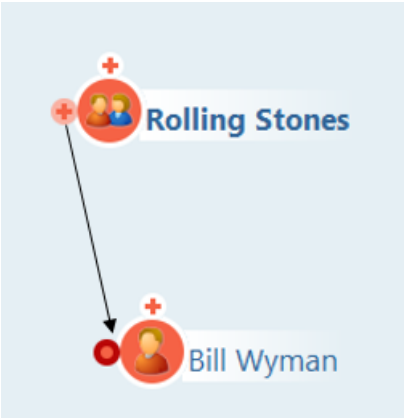
Der Graph zeigt immer nur einen Ausschnitt des Netzes. Objekte können in der Graphansicht ein- und ausgesblendet und es kann durch den Graph navigiert werden.

Doch nicht nur zur übersichtlichen Darstellung der Objekte und Relationen eignet sich der Graph. Im Graph-Editor können Objekte und Relationen außerdem editiert werden.

Auf der linken Seite eines Knotens befindet sich ein Anfasser in Form eines Plus-Zeichens für die Interaktion mit dem Objekt. Durch einen Doppelklick auf den Anfasser werden alle Benutzerrelationen des Objekts angezeigt bzw. ausgeblendet.

Das Verknüpfen von Objekten über eine Relationen wird im Graph-Editor wie folgt vorgenommen:

1. Den Cursor über dem Anfasser links vom Objekt mit der linken Maustaste positionieren.
2. Cursor in gedrückter Position zu einem anderen Objekt ziehen (Drag&Drop). Falls mehrere Relationen zur Auswahl stehen, erscheint eine Liste aller möglichen Relationen zur Auswahl. Falls nur eine mögliche Relation zwischen den beiden Objekten existiert, wird diese gezogen und keine Liste angezeigt.

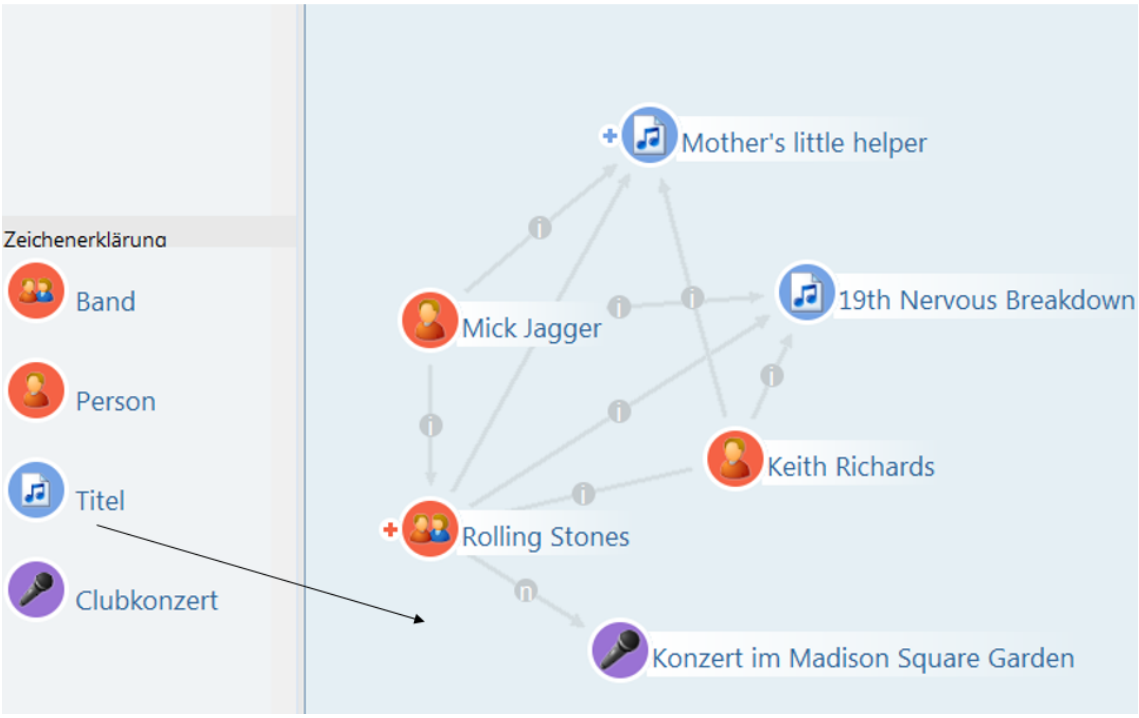


Um Objekte im Graph-Editor anzuzeigen, gibt es verschiedene Möglichkeiten:

- Objekte können aus der Trefferliste im Hauptfenster von i-views mit Drag&Drop in das Fenster des Graph-Editors hineingezogen werden.
- Wenn der Name des Objekts bekannt ist, kann durch einen Rechtklick auf eine leere Stelle im Graph-Editor das Kontextmenü geöffnet werden, wo über die Funktion „Objekt anzeigen“ der Name des gewünschten Objektes eingegeben werden kann.

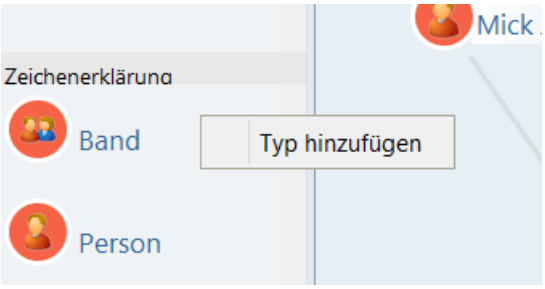
Soll ein Objekt im Graph-Editor ausgeblendet werden, kann es durch Anklicken mit gleichzeitig gedrückter Strg-Taste aus dem Graph-Editor entfernt werden. Damit wird nichts in den Daten geändert: Das Objekt existiert unverändert im semantischen Netz, es wird nur nicht mehr im aktuellen Graph-Editor-Ausschnitt gezeigt, d.h. es wird ausgeblendet.

Im Graph-Editor können auch neue Objekte angelegt werden. Dazu ziehen wir aus der Legende links im Fenster des Graph-Editors den gewünschten Objekttyp mit Drag&Drop auf die Zeichenfläche:



Sind keine Objekttypen in der Legende zu sehen, können diese mit rechtsklick im Feld der

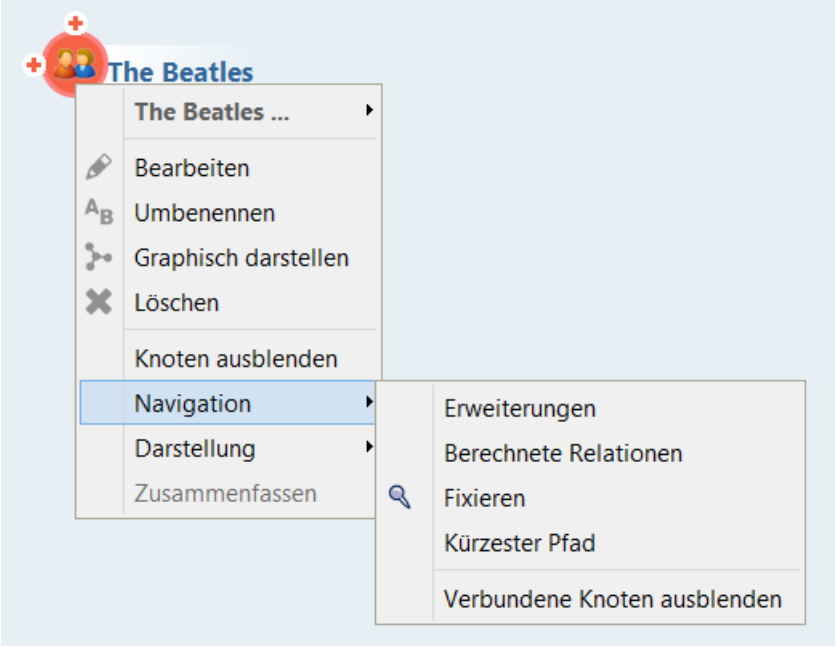
Legende gesucht werden. Anschließen wird der Name des Objektes vergeben.



Es erscheint wieder der Editor, in dem sich die für das Objekt möglichen Relationen, Attribute und Erweiterungen hinzufügen lassen.

1.1.4.2 Operationen auf Objekten im Graph-Editor

Im Graph-Editor können über das Kontextmenü mit Rechtsklick auf das Objekt weitere Operationen ausgeführt werden. Größtenteils bietet dieses Kontextmenü dieselben Funktionen wie der Formular-Editor, enthält aber zusätzlich weitere Graph-Editor-spezifische Komponenten.



In diesem Kontext-Menü stehen folgende Graph-Editor-spezifische Funktionen zur Auswahl:

- **Knoten ausblenden:** Hier kann der Knoten ausgeblendet werden.
- **Navigation - Erweiterungen**: Öffnet die Erweiterungen zu einem Objekt.
- **Navigation - Berechnete Relationen**: Öffnet die berechneten Relationen zu einem Objekt.
- **Navigation - Fixieren:** Fixiert die Position eines Knotens im Graph-Editor, so dass er auch bei einem Neuaufbau des Layouts nicht verschoben wird. Die Fixierung kann mit der Option *Lösen* wieder aufgehoben werden
- **Navigation - Kürzester Pfad:** Berechnet ab einem selektierten Objekt oder Objekttyp den kürzesten Pfad zu vorher selektieren Knoten. Wird für diese Ziele ein Pfad mit

weniger als 6 Schritten gefunden, werden alle (bisher nicht angezeigten) Zwischenstationen eingeblendet. Im Nichterfolgsfall erscheint ein Dialog, der das Fehlschlagen meldet.

- **Navigation - Verbundene Knoten ausblenden**: Blendet alle mit dem ausgewählten Objekt verbundenen Knoten aus.
- **Darstellung:** Hier lässt sich die Darstellungsgröße des Knoten-Icons auswählen.
- **Darstellung - Neue Beschreibung:** Hier kann die Bezeichnung des angezeigten Objekts im Graph-Editor geändert werden. Der Name des Objekts bleibt unverändert.

Auswahl mehrerer Objekte

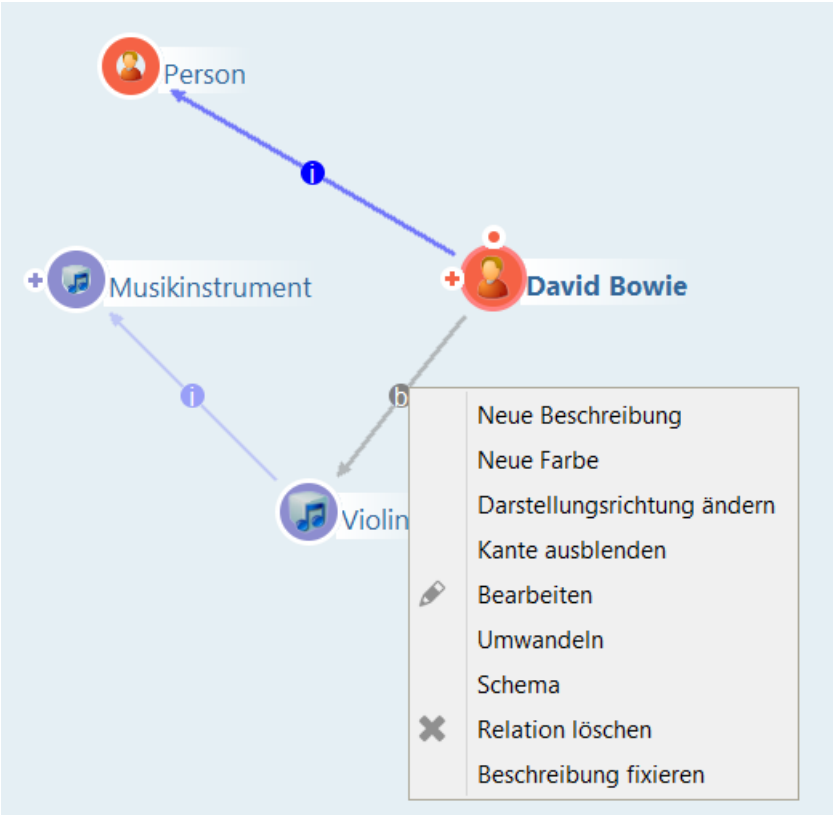
Zusätzlich gibt es die Möglichkeit mehrere Knoten und Relationen gleichzeitig auszuwählen **-** entweder durch Anklicken mit der Maus (Shift-Taste für wiederholte Auswahl) oder durch Aufziehen eines Kastens mit der Maus um die Objekte, die markiert werden sollen. Ein Rechtsklick auf eine leere Stelle im Graph-Editor bietet Zugang zum Menüpunkt "Auswahl", wo mit den Optionen „Alles auswählen“ und „Nichts auswählen“ alle bzw. keine Knoten im Graph-Editor markiert werden. Markierte Objekte werden durch einen roten Kreis gekennzeichnet.

Auch für eine Auswahl mehrerer Objekte kann durch Rechtsklick ein Kontextmenü geöffnet werden. Dieses bietet folgende Graph-Editor spezifische Aktionen:

- **Graphisch darstellen**: Öffnet einen neuen Graph-Editor mit den ausgewählten Knoten.
- **Knoten ausblenden**: Entfernt die ausgewählten Knoten aus dem aktuellen Graph-Editor.
- **Navigation - Fixieren:** Fixiert die Position der Knoten im Graph-Editor.
- **Navigation - Kürzester Pfad:** Berechnet für alle selektierten Objekte den kürzesten Pfad zu vorher selektieren Knoten.

Der Graph-Editor verfügt sowohl über eine Menüleiste als auch über Kontext-Menüs, die von einem Knoten oder einer Relation ausgehend über die rechte Maustaste aufgerufen werden. Aktionen aus der Menüleiste beziehen sich auf den gesamten Graph-Editor oder markierte Knoten, Aktionen aus einem Kontext-Menü hingegen auf den betroffenen Knoten oder eine Relation.

Das Kontext-Menü für Relationen im Graph-Editor kann mit Rechtsklick auf die Relation aufgerufen werden. Hier kann der Relation für die Darstellung im Graph-Editor eine neue Beschreibung gegeben werden, die Farbe und die Darstellungsrichtung lassen sich ändern (standardmäßig wird die Hauptrichtung einer Relation angezeigt), die ausgewählte Kante kann ausgeblendet werden, die Relation kann bearbeitet oder umgewandelt werden, das Schema-Fenster kann geöffnet werden, die Relation kann gelöscht werden und die Beschreibung kann fixiert werden.



1.1.4.3 Ansicht

Das Menü „Ansicht“ stellt viele weitere Funktionen für die graphische Darstellung von Objekten und Objekttypen zur Verfügung:

Voreinstellungen: Öffnet das Menü für globale Einstellungen (auch zugänglich im globalen Einstellungsfenster - Registerkarte „Persönlich“ - Graph). Hier kann eingestellt werden, ob Attribute, Relationen und Erweiterungen in einem kleinen Mouse-over-Fenster über dem Objekt erscheinen sollen Bubble-Help mit Details anzeigen und wie viele Knoten sich im Graph-Editor maximal in einem Schritt einblenden lassen.

Hintergrund ändern: Hier kann die Farbe des Hintergrunds geändert oder ein eigenes Bild als Hintergrund eingefügt werden.

Knoten automatisch ausblenden: Blendet automatisch überschüssige Knoten aus, sobald mehr als die gewünschte Anzahl an Knoten sichtbar ist. Die Anzahl kann im Eingabefeld „max. Knoten“ in der Symbolleiste eingestellt werden.

Knoten automatisch positionieren: Führt für neu eingeblendete Knoten automatisch die Layoutfunktion auf.

Beschreibungen fixieren: Mit dieser Option sind die Namen aller Relationen immer sichtbar, nicht nur beim Roll-over mit der Maus. Alternativ kann gezielt im Kontextmenü einer Relation deren Beschreibung fixiert werden.

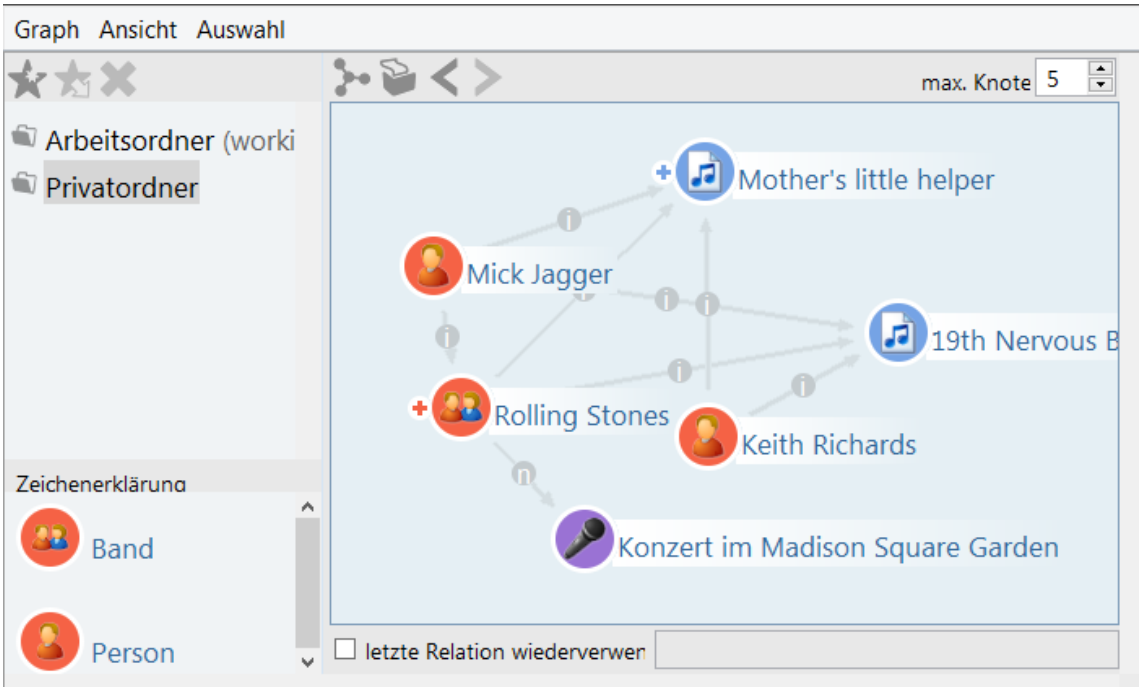
Interne Namen anzeigen: Blendet an Typknoten die internen Namen der Typen ein.

Ausgeblendete Kanten wieder darstellen: Alle Kanten, die per Kontext-Menü ausgeblendet wurden, werden wieder angezeigt.

Kanten automatisch wieder abblenden: Die Kanten werden abgeblendet.

Das Fenster des Graph-Editors und das Hauptfenster des Knowledge-Builders stellen noch weitere Menüpunkte zur Verfügung, die bei der Modellierung eine Hilfestellung bieten können.

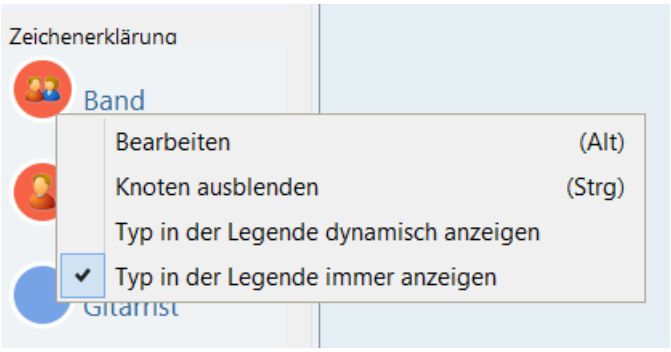
Links im Fenster des Graph-Editoren befindet sich die Legende der Objekttypen.



Diese Legende zeigt die Objekttypen zu den konkreten Objekten auf der rechten Seite.

Durch Drag&Drop eines Eintrags aus der Legende in die Zeichenfläche können Sie ein neues konkretes Objekt des entsprechenden Typs erzeugen.

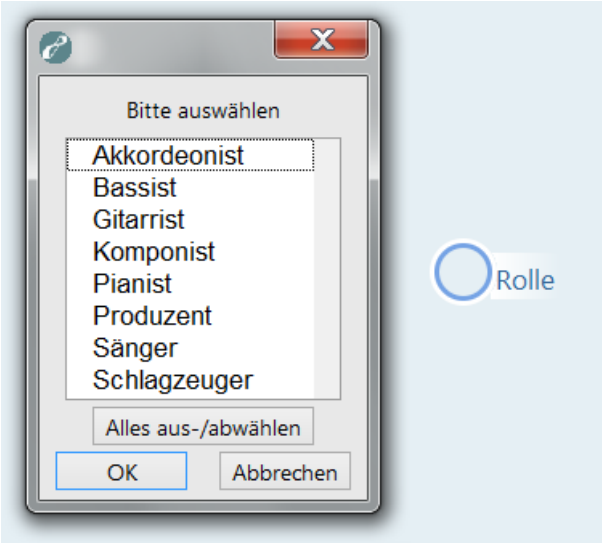
Über das Kontextmenü auf den Legendeneinträgen können alle konkreten Objekte dieses Typs aus der Darstellung ausgeblendet werden. Hier lassen sich auch Legendeneinträge „festhalten“, und neue Objekttypen in die Legende aufnehmen (unabhängig davon, ob konkrete Objekte von diesem Typ in der Darstellung vertreten sind).



Max Knoten: Wenn ein Knoten/Objekt viele Nachbarobjekte hat, ist es oft nicht sinnvoll, alle beim Klick auf den Anfasser gleich einzublenden. Hierfür kann an zwei Stellen eine maximale Anzahl einzublendender Knoten definiert werden.

1. Über das globale Einstellungsfester Registerkarte „Persönlich“ Graph lässt sich die Anzahl bei max. Knoten festlegen, oder
2. im Fenster des Graph-Editors oben rechts ist diese Aktion ebenfalls aufrufbar.

Wird der Anfasser zur Einblendung der Nachbarobjekte angeklickt, erscheint anstelle der Objekte eine Auswahlliste.



Detaillierte Ansicht: Wenn der Knowledge-Builder gestartet wird, ist per Default die Option „Detaillierte Ansicht“ ausgewählt. Durch Doppelklick auf die Anfasser an den Knoten kann zu weiteren Knoten navigiert werden:

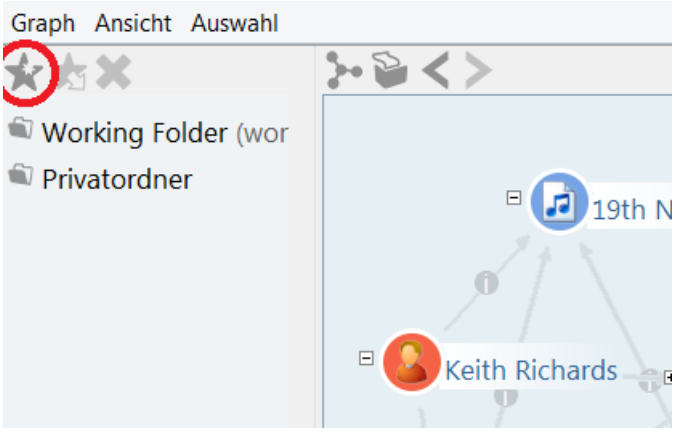
- der oberer Anfasser blendet den Objekttyp bei konkreten Objekten ein bzw. den Obertyp bei Objekttypen ein
- der untere Anfasser ist nur bei Objekttypen existent und führt dazu, dass die Untertypen angezeigt werden
- der linke Anfasser zeigt Relationen zu anderen Objekten an

Wenn der Haken bei der Option „Detaillierte Ansicht“ entfernt wird, gibt es nur noch den linken Anfasser, dargestellt durch ein Kästchen mit Plus-Zeichen:

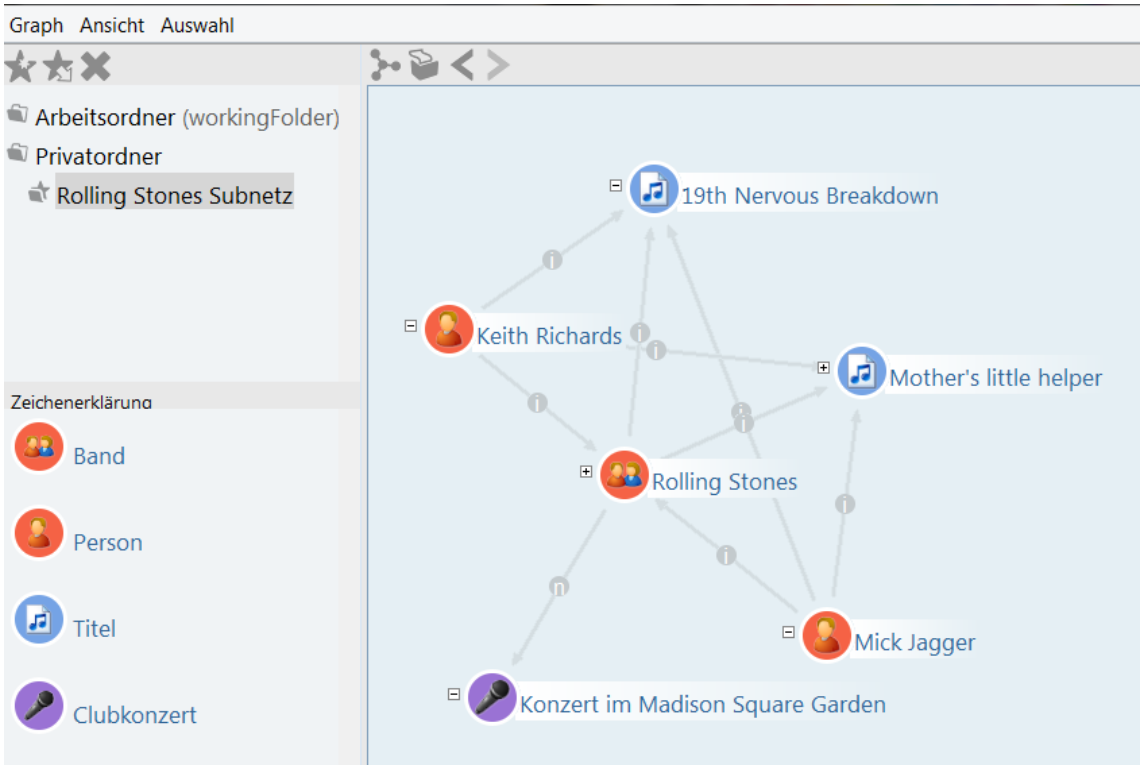
Mit dem Plus-Zeichen lassen sich nur die mit dem angezeigten Objekt über Relationen verknüpften Nachbar-Objekten einblenden. Bei mehreren Verknüpfungen erscheint ebenfalls der Dialog mit der Auswahlliste.

1.1.4.4 Lesezeichen und Historie

Lesezeichen: Die erstellten Ansichten im Graph-Editor lassen sich als Lesezeichen speichern. Die Objekte werden in der Position gespeichert, wie sie im Graph-Editor platziert wurden.

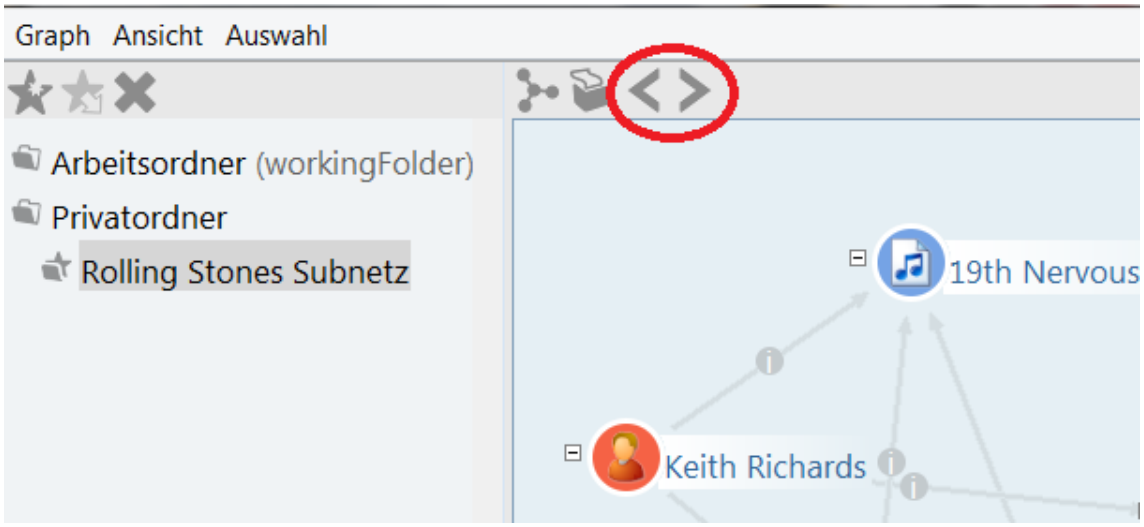


Wenn ein Lesezeichen angelegt wird, muss zunächst noch ein Ordner ausgewählt werden, dann kann dem Lesezeichen noch ein Name gegeben werden.



Lesezeichen sind jedoch keine Datenbackups: Objekte und Beziehungen, die nach dem Speichern eines Lesezeichens gelöscht wurden, sind auch beim Anzeigen des Lesezeichens nicht mehr in der Darstellung vorhanden.

Historie: Mit den Schaltflächen „Navigation rückgängig“ und „Navigation wiederherstellen“ können die Elemente in der Reihenfolge, in der sie eingeblendet wurden, wieder ausgeblendet werden (und vice versa). Außerdem machen die Schaltflächen die neue Anordnung der Knoten des Auto-Layouts rückgängig. Die Schaltflächen befinden sich in der Kopfzeile im Fenster des Graph-Editors, oder im Menü „Graph“.



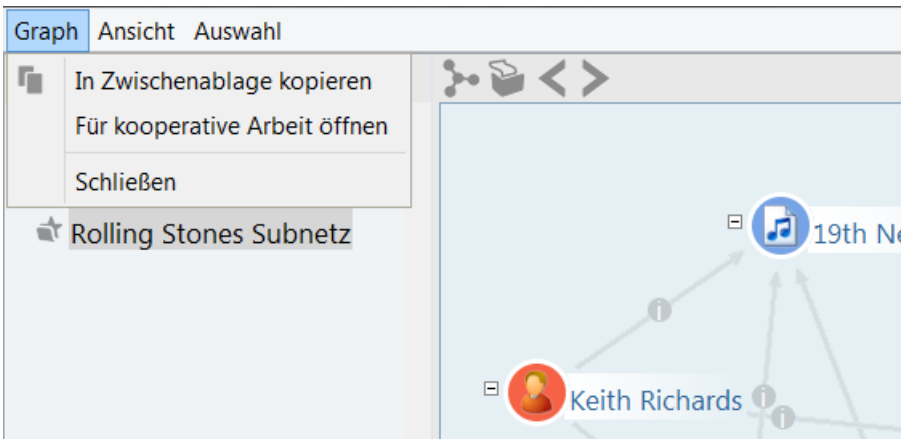
Layout: Mit der Layout-Funktion lassen sich Knoten automatisch positionieren, wenn viele Knoten nicht manuell positioniert werden sollen. Bei Einblenden weiterer Knoten werden diese ebenfalls über die Layout-Funktion automatisch im Graph positioniert.

Drucken: Die Druck-Funktion öffnet das Dialogfenster zum Ausdrucken, oder zur Generierung einer PDF-Datei des angezeigten Graphs.

Im Menü *Graph* stehen weitere Funktionen für den Graph-Editor zur Verfügung:

In Zwischenablage kopieren: Diese Funktion erzeugt einen Screenshot des aktuellen Graph-Editor-Inhalts. Dieses Bild kann dann beispielsweise in ein Zeichen- oder Bildbearbeitungsprogramm eingefügt werden.

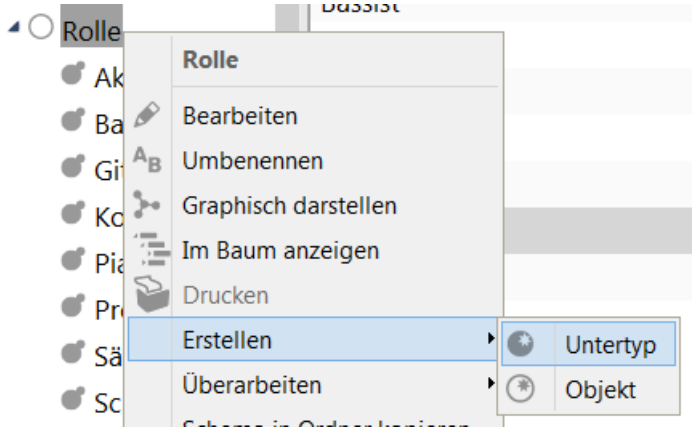
Für kooperative Arbeit öffnen: Diese Funktion ermöglicht anderen Benutzern, gemeinsam und gleichzeitig am Graph zu arbeiten. Alle Änderungen und Selektionen eines Benutzers am Graph (Layout, Ein-/Ausblenden von Knoten usw.) werden dann synchron bei allen anderen Benutzern angezeigt.



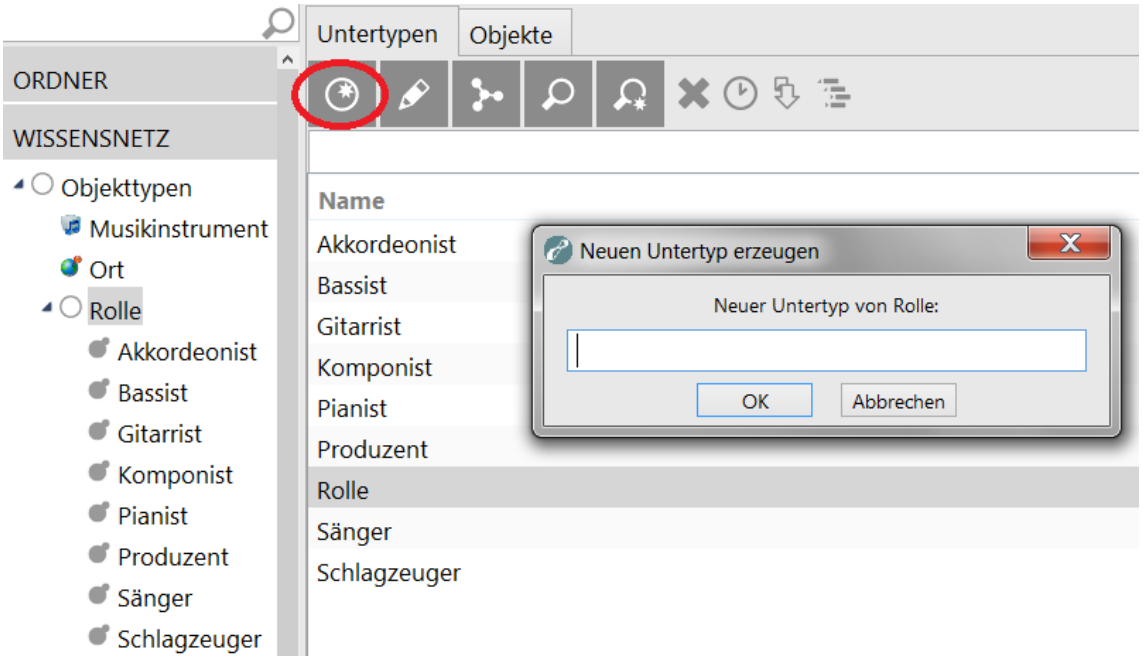
1.2 Schemadefinition / Modell

1.2.1 Typen definieren

Im Kapitel 1.2 ist das Prinzip der Typenhierarchie bereits vorgestellt worden. Sollen neue Typen angelegt werden, erfolgt dies immer als Untertyp eines existierenden Typs. Das Anlegen der Untertypen kann entweder über das Kontextmenü Erstellen -> Untertyp



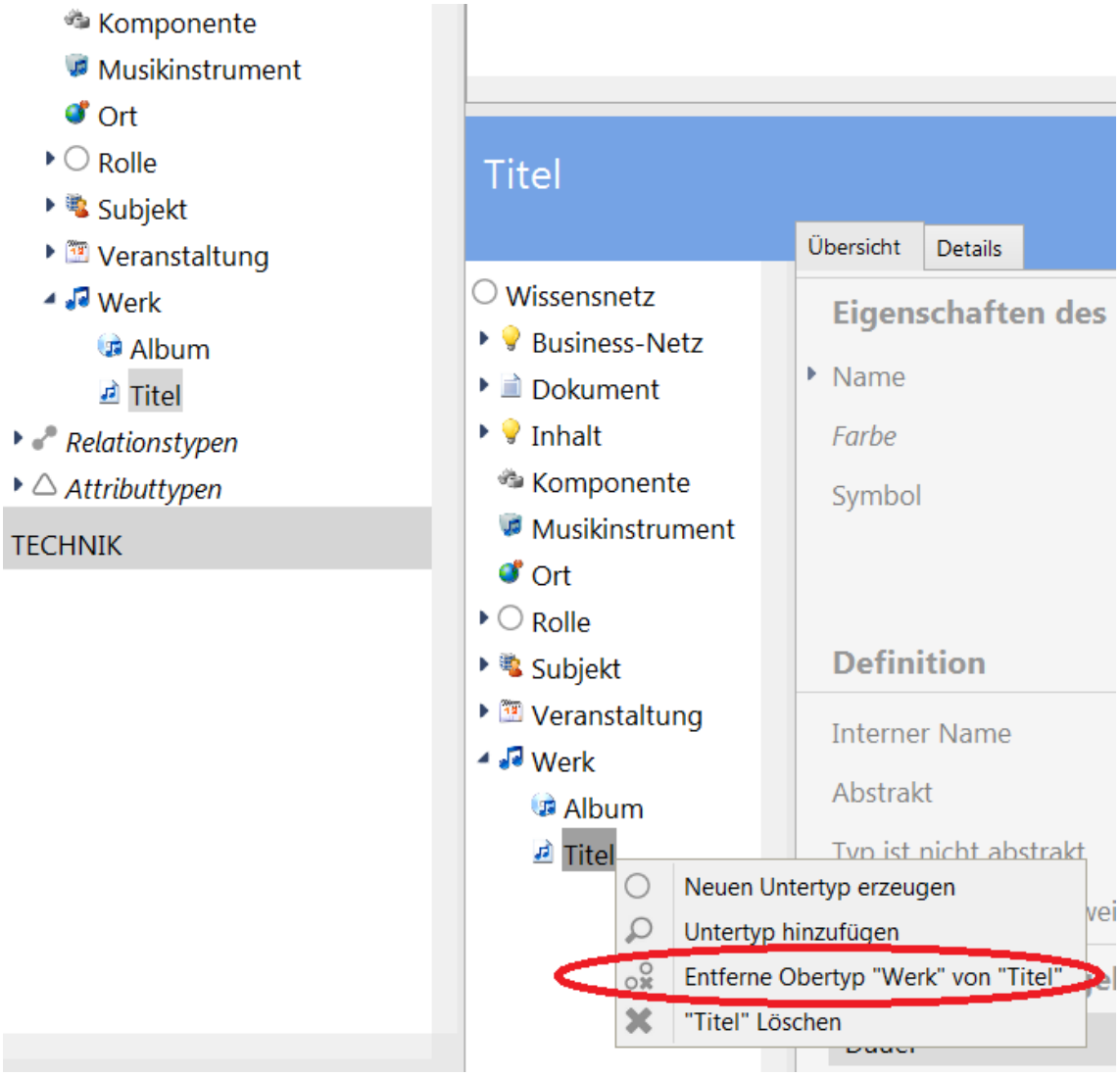
oder im Hauptfenster im Reiter „Untertypen“ über das Suchfeld und die Schaltfläche „Neu“ vorgenommen werden:



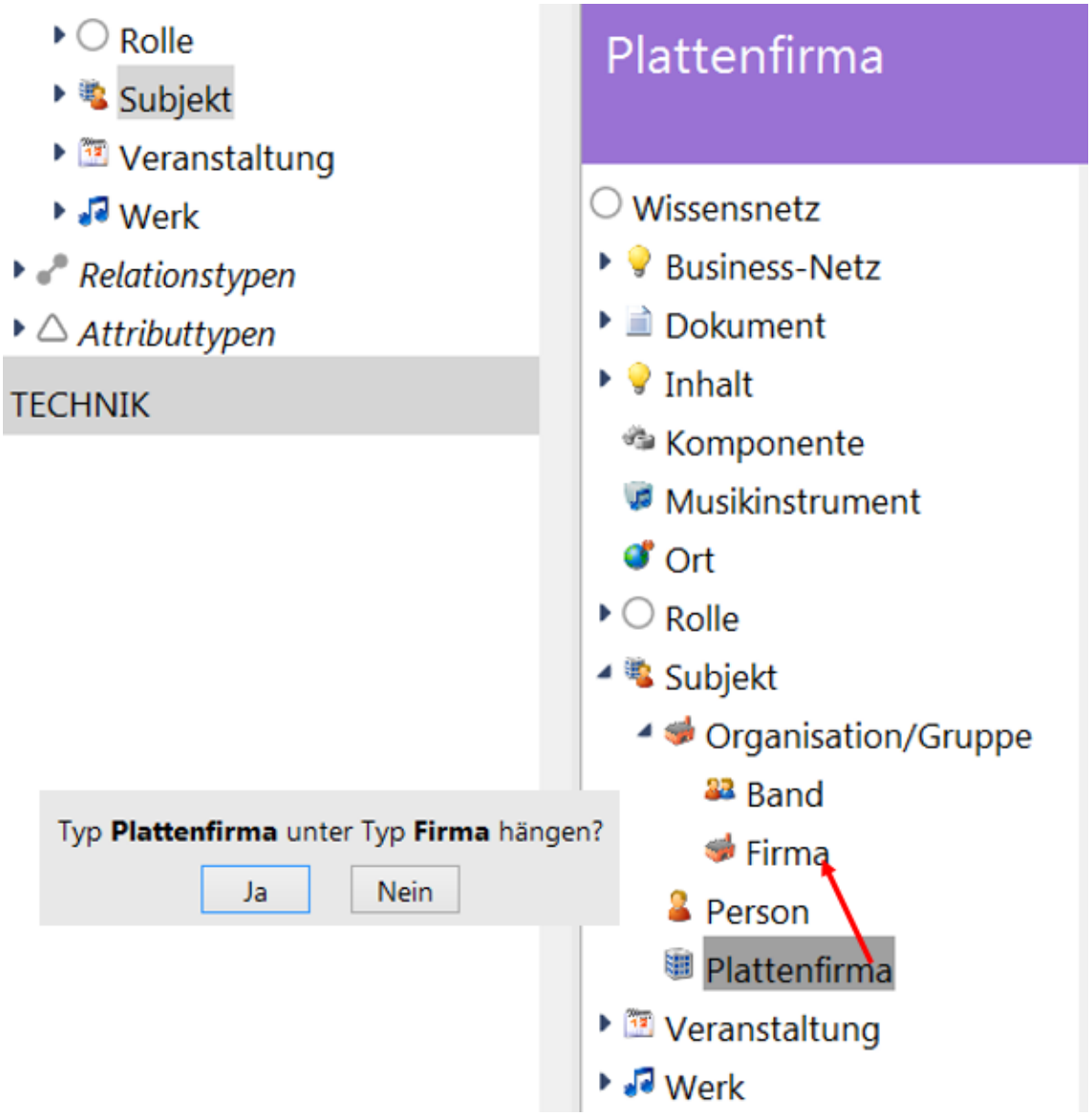
Typenhierarchie ändern

Zum Ändern der Typenhierarchie stehen uns der Baum der Objekttypen im Hauptfenster und der Graph-Editor zur Verfügung.

Im Hierarchie-Baum des Objekt-Editors finden wir im Kontextmenü die Option „Entferne Obertyp xy“.



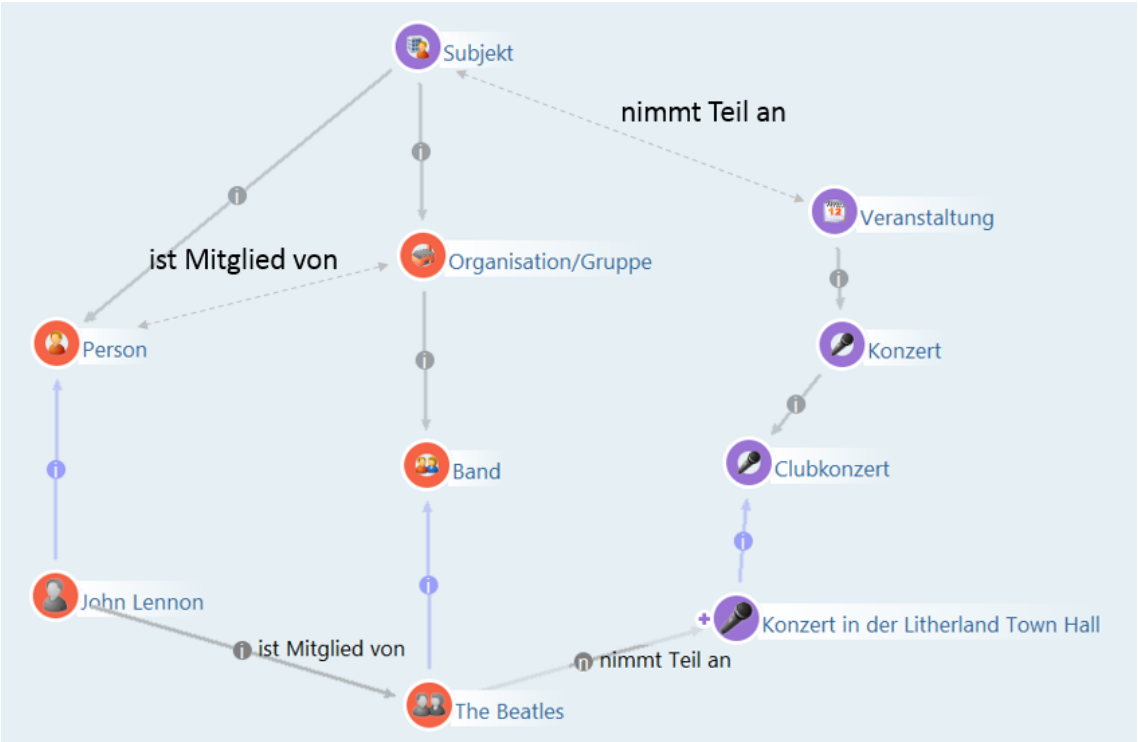
Damit können wir den aktuell selektierten Objekttyp aus seiner Position in der Objekttypen-Hierarchie herauslösen. Mit Drag&Drop können wir einen Objekttyp in einen anderen Ast der Hierarchie verschieben. Halten wir beim Drag&Drop die Strg-Taste gedrückt, wird der Objekttyp nicht verschoben, sondern zusätzlich unter einen weiteren Objekttyp eingeordnet. Nach wie vor gilt: Die Hierarchie der Objekttypen erlaubt Mehrfach-Einordnung und -Vererbung.

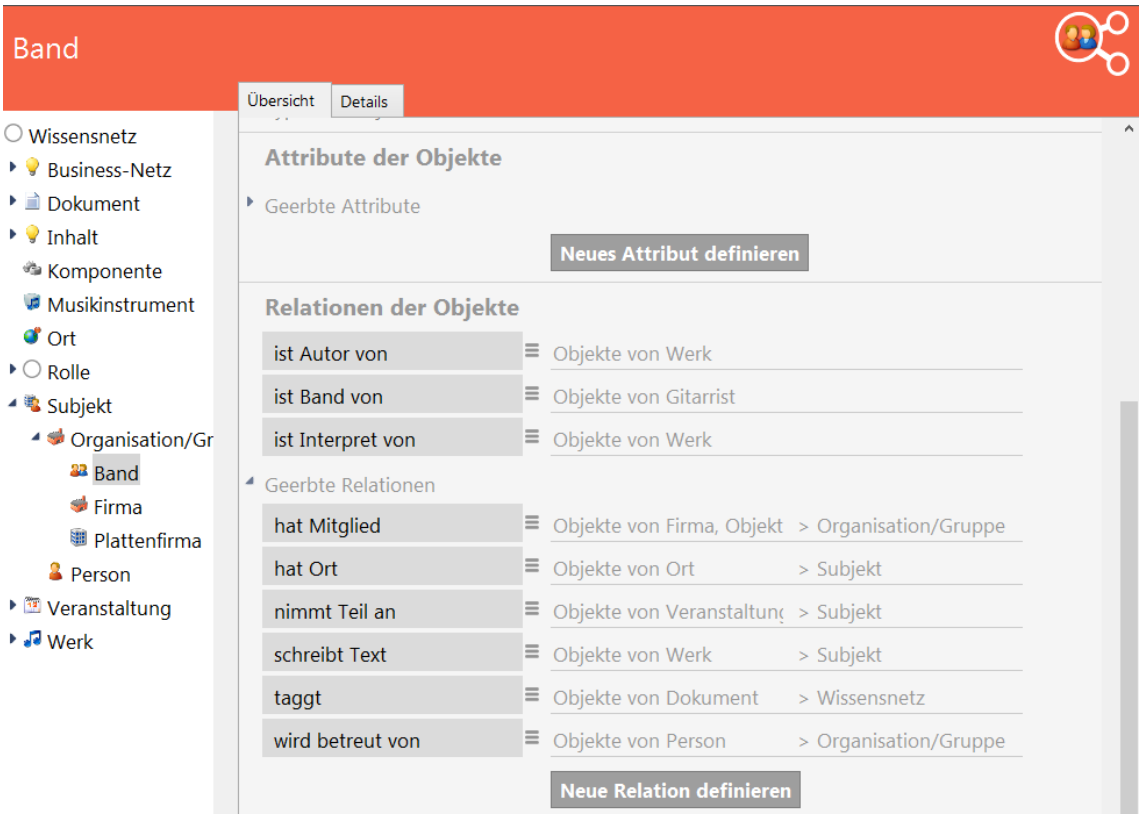


Objekttypen mit Eigenschaften ausstatten

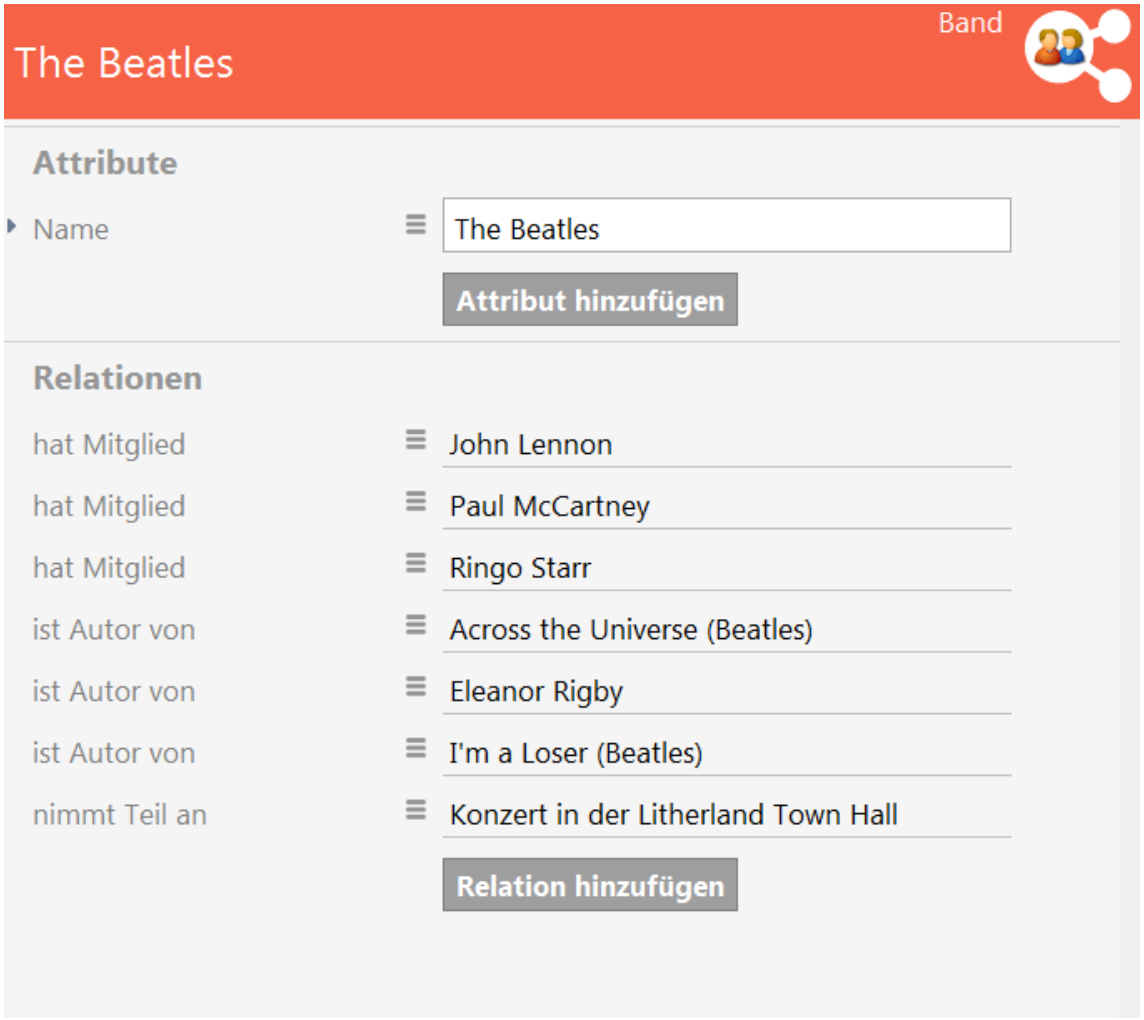
Im einfachsten Fall definieren wir Relationen und Attribute bei einem Objekttyp wie z.B. „Band“ oder „Person“ und stellen sie damit für die konkreten Objekte dieses Typs zur Verfügung. (Bspw. Gründungsjahr und -ort bei Bands, Geburtsdatum und Geschlecht bei Personen, Veranstaltungsort und -datum bei Veranstaltungen.)

Hat der Objekttyp, bei dem die Eigenschaften definiert sind weitere Untertypen, so greift hier das Prinzip der Vererbung: Eigenschaften stehen nun auch für die konkreten Objekte der Untertypen zur Verfügung. Beispiel: eine Band erbt als Untertyp einer Organisation die Möglichkeit, Personen als Mitglieder zu haben. Als Untertyp von „Person oder Gruppe“ erbt die Band auch die Möglichkeit, an Veranstaltungen teilzunehmen:





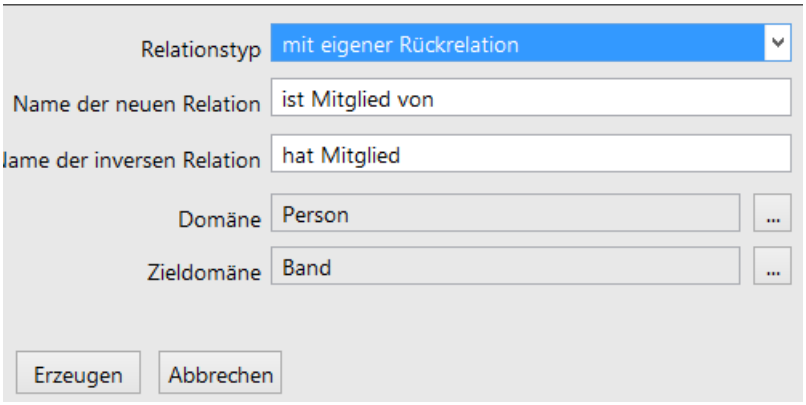
Der Editor für den Objekttyp „Band“ mit direkt dort definierten und geerbten Relationen.



Beim konkreten Objekt stehen die geerbten Eigenschaften ohne Weiteres zur Verfügung, hier wird der Unterschied gar nicht bemerkt.

Beziehungen definieren

Beim Umgang mit Beziehungen herrscht in i-views folgendes Grundprinzip: Eine Beziehung kann nicht nur einseitig sein. Wenn wir für die konkrete Person „John Lennon“ eine Beziehung „ist Mitglied von“ zur Musikgruppe Beatles kennen, dann impliziert das wiederum für die Beatles den Sachverhalt „haben Mitglied John Lennon“. Diese beiden Richtungen sind nicht zu trennen. Deswegen verlangt i-views beim Anlegen neuer Relationstypen von uns die Typen von Quelle ("Domäne") und Ziel der Relationen ("Zieldomäne") - in unserem Beispiel wären das Person und Band - sowie unterschiedliche Benennungen: „ist Mitglied von“ und „hat Mitglied“.



Damit ist die Relation definiert und kann jetzt zwischen Objekten per Drag&Drop gezogen werden.

Attribute definieren

Bei der Definition neuer Attributtypen benötigt i-views neben dem Namen den technischen Datentyp. Folgende technische Datentypen stehen zur Verfügung:

Datentyp	Wie sehen die Werte aus?	Beispiel (Musik-Datenbank)
Attribut	abstraktes Attribut, ohne Attribut-Wert	
Auswahl	frei definierbare Auswahlliste	Bauart eines Musikinstruments (Hollowbody, Fretless usw.)
Boolesch	»ja« oder »nein«	Musikgruppe noch aktiv?
Datei	externe Datei eines beliebigen Formats, die als »Blob« in die semantische Graphdatenbank importiert wird	wav-Datei eines Musiktitels
Datum	Datumsangabe tt.mm.jjjj (in der deutschen Spracheinstellung)	Veröffentlichungsdatum eines Tonträgers
Datum und Uhrzeit	Datums- und Uhrzeitangabe tt.mm.jjjj hh:mm:ss	Beginn einer Veranstaltung, bspw. Konzert
Farbwert	Farbauswahl aus Farbpalette	
Flexible Zeit	Monat, Monat + Tag, Jahreszahl, Uhrzeit, Timestamp	Ungefähres Eintrittsdatum eines Mitglieds in eine Musikgruppe

Fließkommazahl	Zahlenwert mit beliebiger Anzahl von Nachkommastellen	Preis einer Eintrittskarte zu einer Veranstaltung
Ganzzahl	Zahlenwert ohne Nachkommastellen	Laufzeit eines Musiktitels in Sekunden
Geographische Position	Geographische Koordinaten im WGS84-Format	Ort einer Veranstaltung
Gruppe	ohne Attributs-Wert, dient als Träger zu gruppierender Meta-Attribute	
Internet-Verknüpfung	Link auf eine URL	Webseite einer Musikgruppe
Intervall	Datumsintervall: Intervalle von Zahlen, Zeichenkette, Zeit- oder Datumswert	Zeitraum zwischen Produktion eines Albums und seiner Veröffentlichung
Passwort	je Attribut-Instanz und Passwort ein eindeutig gehashter Wert (SHA256), der nur zum Validieren des Passworts verwendet wird	
Verweis auf [...]	Verweis auf Teile der Konfiguration des Netzes: Suchen, Abbildung einer Datenquelle, Skripte und Ordner. Wird beispiels- weise in der REST- Konfiguration benutzt.	
Zeichenkette	beliebige Folge alphanumerischer Zeichen	Rezensionstext zu einem Tonträger
Zeit	Zeitangabe hh:mm:ss	Dauer einer Veranstaltung

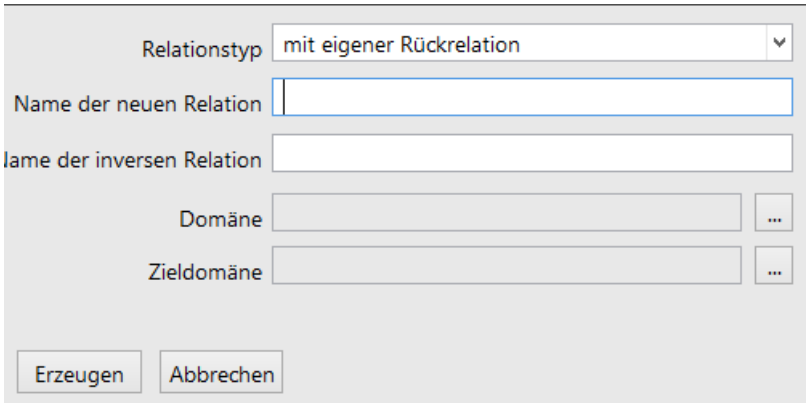
Intension der Nutzung solcher Datentypen ist es, nicht einfach alles als Zeichenkette zu definieren. Technische Datentypen in festgelegter Formatierung bieten später spezielle Möglichkeiten zum abfragen und vergleichen. Beispielweise können Zahlenangaben innerhalb der Strukturabfragen mit größeren oder kleineren Werten verglichen werden, für Geokoordinaten kann eine Umkreissuche definiert werden, u.v.m.

1.2.2 Relations- und Attributtypen

Relations- und Attributtypen (kurz Eigenschaftstypen) sind die Typen der konkreten Eigenschaften, die an den Objekten gespeichert sind.

Neuen Relationstyp anlegen

Über die Schaltfläche im Objekt-Editor „Relation hinzufügen“ startet der Editor zum Anlegen eines neuen Relationstyps:



Editor für das Anlegen eines neuen Relationstyps (siehe auch Kapitel 2.1 Beziehungen definieren)

Name: Benennungen für Relationstypen lassen sich in i-views frei vergeben, sollten aber nach der Prämisse eines verständlichen Datenmodells gewählt werden. Die folgende Konvention kann dazu hilfreich sein: Der Namen der Beziehung wird so formuliert, dass die Struktur [Name des Quellobjekts] [Relationsname] [Name des Zielobjekts] einen verständlichen Satz ergibt:

[John Lennon] [**ist Mitglied von**] [The Beatles]

Weiterhin ist es hilfreich, wenn die Gegenrichtung (inverse Beziehung) die Wortwahl der Hauptrichtung aufgreift: „hat Mitglied / ist Mitglied von“.

Domäne und Zieldomäne: Hier wird bestimmt bei welchen Typen von Objekten die Relation angelegt werden soll: Ein Objekttyp bildet die Quelle der Relation ("Domäne"), ein weiterer Objekttyp das Ziel ("Zieldomäne"). Der Ziel-Objekttyp bildet wiederum den Definitionsbereich der inversen Relation. Beim Anlegen kann der Einfachheit halber an dieser Stelle nur ein Objekttyp eingetragen werden. Im Relationstyp-Editor (siehe unten) lassen sich im Nachhinein noch weitere Objekttypen definieren.

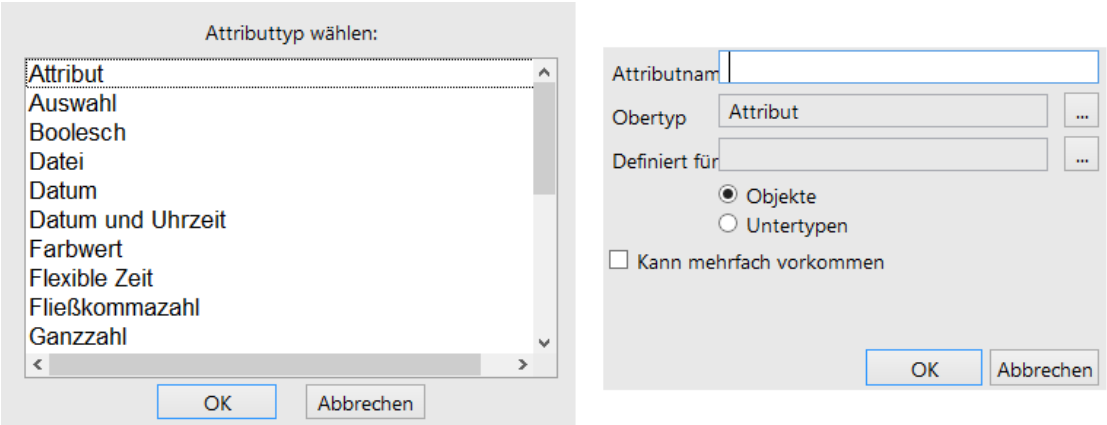
Übergeordneter Relationstyp: Genau wie die Objekttypen können auch die Relations- und Attributtypen in einer Hierarchie stehen. Die Hierarchie der Relationstypen ist ein einfaches, aber mächtiges Instrument, um Komplexität im Griff zu behalten.

Beispiel: Für Abfragen kann über eine Relation „hat Autor“ differenziert werden, wer den Text des Songs, und wer die Musik geschrieben hat. Gleichzeitig haben wir auch Abfragen, bei denen keine Ausdifferenzierung gewünscht ist und alle Beteiligten zugeliefert werden sollen. Ohne die Relationshierarchie müssten wir jetzt alle diese Suchen komplizierter machen, indem wir beide Relationen abfragen. Stattdessen können wir die Relationen „schreibt Text“ und „schreibt Musik“ als Untertypen von „schreibt Song“ (oder: ist Autor von) definieren, damit können wir immer noch auf der Ebene „schreibt Song“ abfragen. i-views fragt automatisch die Unterrelationen mit ab.

Der Untertyp impliziert somit den Obertyp. Dieses Prinzip greift bei Relationen genauso wie bei Attributen und Objekten.

Neuen Attributtyp anlegen

Über die Schaltfläche im Objekt-Editor „Neues Attribut definieren“ startet der Editor zum Anlegen eines neuen Attributtyps:



Zweistufiger Dialog zum Anlegen eines neuen Attributtyps

Im Fenster links wird das Format des Attributtyps definiert (Datum, Gleitkommazahl, Zeichenkette usw.) Nach Auswählen und Bestätigen des Attributtyps kann dieser im Folgedialog mit Attributnamen weiter spezifiziert werden.

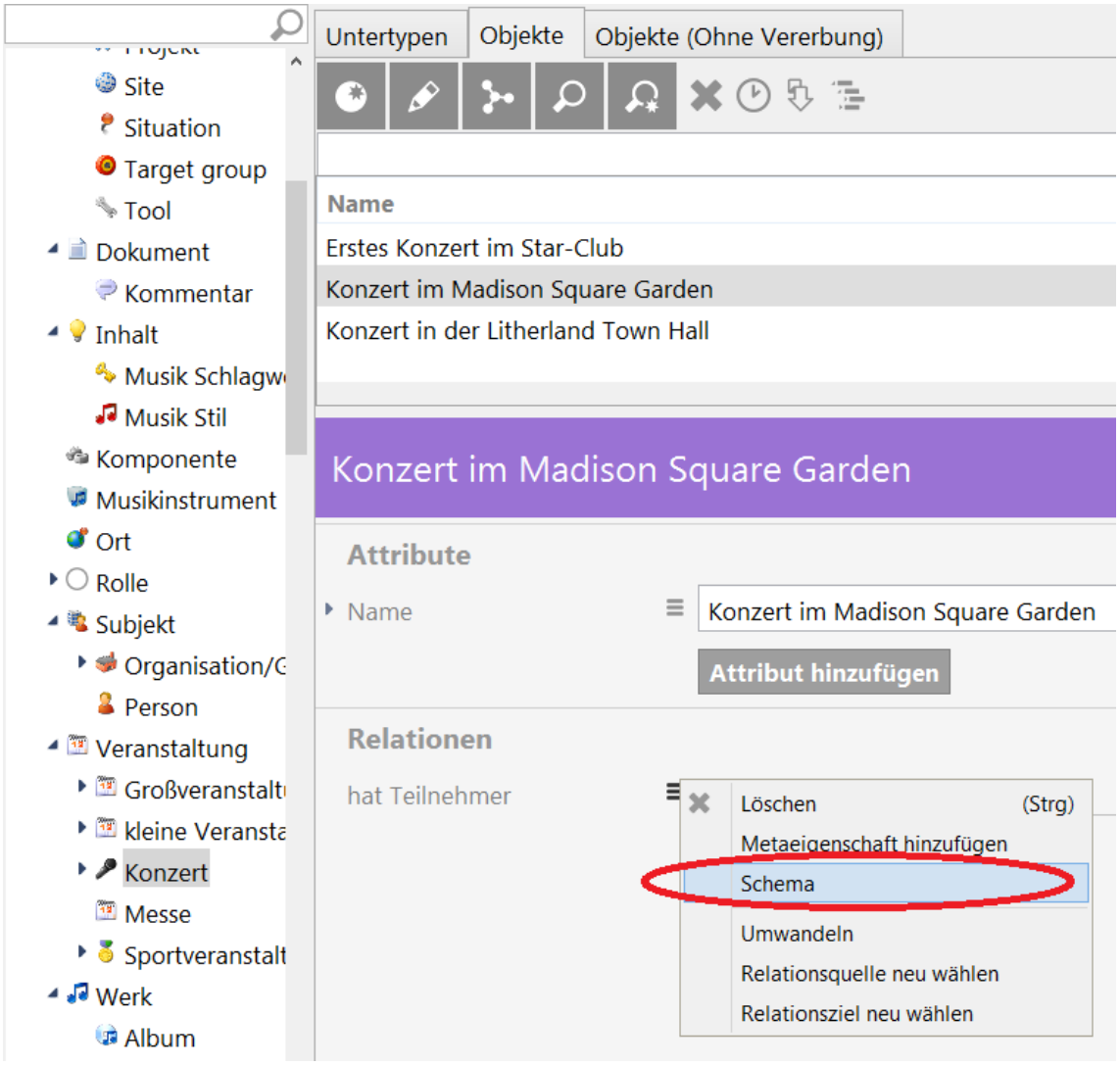
Obertyp: Hier wird festgelegt, an welcher Hierarchiestufe der Attributtyp stehen soll.

Kann mehrfach vorkommen: Attribute können je nach Attributtyp einfach oder mehrfach vorkommen: eine Person hat nur ein Geburtsdatum, kann aber bspw. mehrere akademische Titel zur gleichen Zeit haben (bspw. Doktor, Professor und Honorarkonsul).

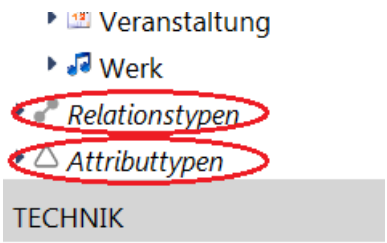
Details bearbeiten

Die Dialoge zum Anlegen von neuen Attribut- und Relationstypen sind reduzierte Ansichten der Attribut- und Relationstyp-Editoren. Zur Bearbeitung von Details von Relationen und Attributen müssen Editoren mit erweitertem Funktionsumfang gestartet werden.

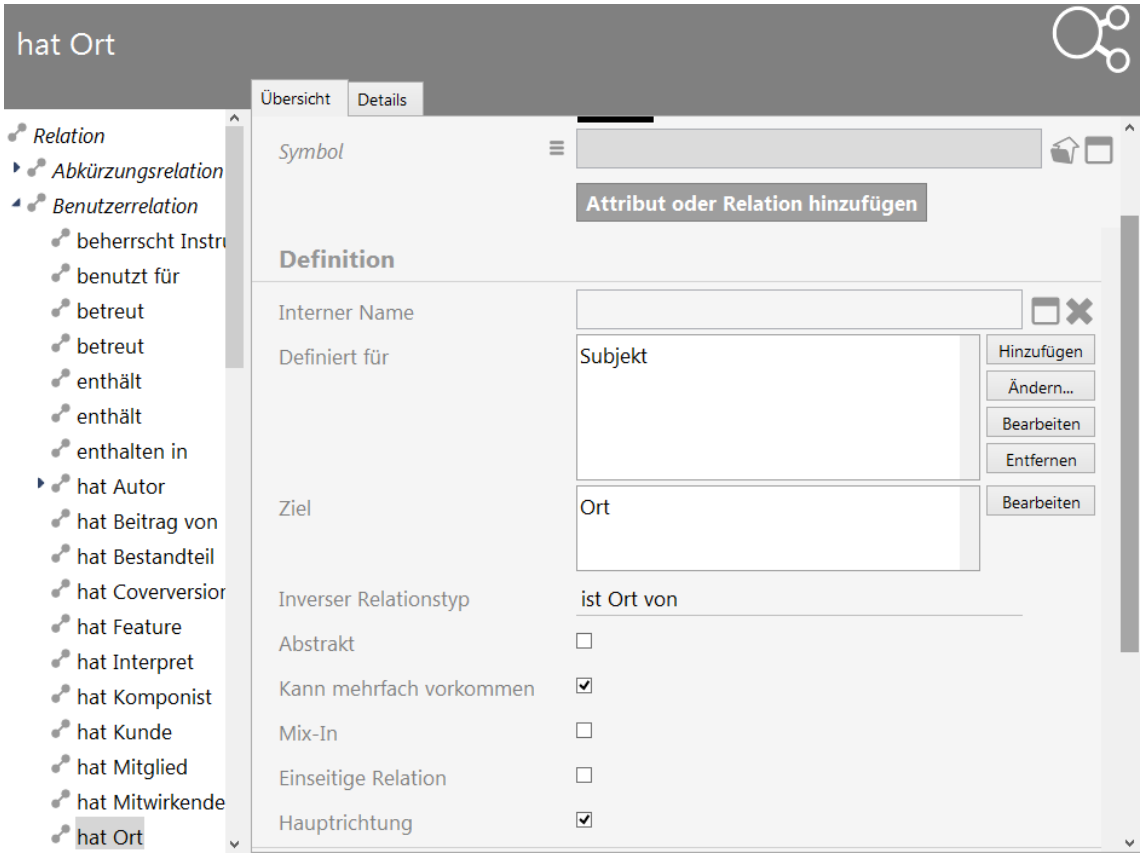
Zu diesen beiden Editoren können Sie über die Auflistung der Relationen und Attribute im Reiter „Schema“ des Objekt-Editors gelangen:



Alternativ kann über den Hierarchie-Baum links im Hauptfenster zugegriffen werden. Unter den Objekttypen befinden sich die Hierarchien für Relations- und Attributtypen. Die Editoren werden mit Rechtsklick auf die zu bearbeitende Relation oder Attribut im Kontextmenü mit „Bearbeiten“ gestartet.



Im Folgenden schauen wir uns die Details der Definition von Eigenschaften am Beispiel des Relationstyp-Editors an (die Attributtypdefinition ist eine Untermenge davon):



Definiert für: Hier können wir nachträglich ändern, bei welchen Objekttypen die Relation angelegt werden kann. Relationen können zwischen mehreren Objekten definiert werden und damit mehrere Quellen und Ziele haben.

So können wir es z.B. im Schema erlauben, dass sowohl Personen als auch Bands Autoren eines Songs sein können oder einem Ort zugeordnet werden - auch wenn sie keinen gemeinsamen Obertyp haben.
Mit der Schaltfläche „Hinzufügen“ können wir weitere Objekttypen hinzunehmen. Mit „Entfernen“ können wir dem selektierten Objekttyp und allen seinen Objekten die Möglichkeit entziehen diese Relation einzugehen.

„Ändern“ ermöglicht das austauschen eines Objekttyps. Bereits existierende Relationen werden dann vom System gelöscht. Falls es zu löschende Relationen gibt, erscheint vor der Durchführung der Änderung ein Bestätigungsdialog.

Ziel: Hier lässt sich nachträglich ändern, zu welchen Typen von Objekten die Relation gezogen werden kann. Um den Ziel-Objekttyp zu ändern, muss zum inversen Relationstyp gewechselt werden: Die Schaltfläche zum Wechseln trägt die Benennung des inversen Relationstyps. Nach anklicken der Schaltfläche erscheint die inverse Relation im Editor und kann auf dieselbe Weise wie die vorherige Relation bearbeitet werden.

Abstrakt: Wenn wir eine Relation definieren wollen, die nur zur Gruppierung dient aber selber keine konkreten Eigenschaften ausprägen soll, dann definieren wir sie als "abstrakt"

Beispiel: Wenn die Relation „schreibt Song“ als abstrakt definiert wird, bedeutet dies: wenn wir Songs und ihre Relation zu Künstlern oder Bands anlegen, können wir nur spezifische Angaben machen (wer hat den Text geschrieben, wer die Musik). Die unspezifische Relation „schreibt Song“ kann nicht in den konkreten Daten angelegt werden, sondern nur für Abfra-

gen verwendet werden.

Kann Mehrfach vorkommen: Ein Merkmal von Relationen ist es, ob sie mehrfach vorkommen können. Beispiel: die Relation „hat Geburtsort“ kann für jede Person nur einmal auftreten, während bspw. die Relation „ist Mitglied von“ mehrfach für eine Person auftreten kann. Somit lassen sich logische Sachverhalte präzise modellieren. Beispielsweise können Musiker als Person nur einen Geburtsort haben, aber (auch zeitgleich) Mitglied in mehreren Musikgruppen sein. Ob die Relation mehrfach vorkommen kann, wird für beide Richtungen der Relation unabhängig angegeben: Eine Person ist nur an einem Ort geboren, der Ort kann aber wiederum Geburtsort von mehreren Personen sein.

Die Option kann nur ausgeschaltet werden wenn die Relation im tatsächlichen Datenbestand nicht mehrfach vorkommt. Bei mehrfachen Vorkommen kann das System nicht automatisch entscheiden, welche der Relationen entfernt werden soll.

Mix-In: Mix-In werden im Kapitel Erweiterungen erklärt.

Einseitige Relation: Ein Relation sollte als einseitige Relation angelegt werden, wenn ansonsten an einem Objekt sehr viele überflüssige Relationen angelegt werden würden. Wird eine Relation als einseitige Relation markiert (dies geht nur an der Hauptrichtung), wird die Gegenrichtung nur noch bei Bedarf angezeigt. Hinweis: Der Haken kann nur gesetzt werden, wenn noch keine Relationen des Relationstyps vorhanden sind. Sind bereits Relationen vorhanden so kann im Kreismenü die Funktion "In einseitige Relation umwandeln" genutzt werden.

Hauptrichtung: Zu jeder Beziehung gehört die Gegenrichtung. Im Kern sind beide Richtungen gleichwertig, aber es gibt zwei Stellen, wo es hilfreich ist eine Hauptrichtung zu bestimmen:

- Im Graph-Editor: Hier stellen sich die Relationen, was Pfeilrichtung und Beschriftung angeht, immer in Hauptrichtung dar - unabhängig davon in welche Richtung sie angelegt wurden.
- Bei einseitigen Relationen (ohne Rückrelation)

1.2.3 Modelländerungen

In i-views lassen sich am Modell zur Laufzeit Änderungen durchführen:

- neue Typen einführen
- die Typenhierarchie beliebig ändern (ohne dafür Tabellen anlegen und uns um Primär- und Fremdschlüssel Gedanken zu machen).

Systemseitig wird für Konsistenz gesorgt. Beim Anlegen von Objekten und Eigenschaften wird die Gegenrichtung einer Relation automatisch mitgezogen. Attributwerte werden darauf geprüft, ob sie zum definierten technischen Datentyp passen (in ein Datumsfeld können wir beispielsweise keine beliebige Zeichenkette eintragen).

Konsistenz ist auch beim Löschen wichtig: Abhängige Elemente müssen immer mit gelöscht werden, sodass keine Reste von Daten gelöschter Elemente in der semantischen Graphdatenbank zurückbleiben.

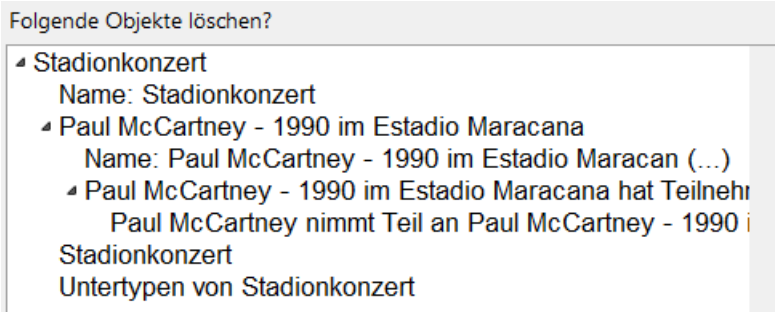
- Wenn also ein Objekt gelöscht wird, werden auch alle seine Eigenschaften mit gelöscht. Wenn wir z.B. das Objekt „John Lennon“ löschen, dann löschen wir damit sein Geburts-

datum, seinen Biographie Text, den wir als Freitext-Attribut bei jeder Person haben können etc. Es wird ebenfalls auch seine Relation „ist Mitglied bei“ zu den Beatles und „ist liiert mit“ zu Yoko Ono. Die Objekte „The Beatles“ und „Yoko Ono“ werden nicht gelöscht, sie verlieren lediglich ihre Verbindung zu John Lennon.

- Beim Löschen einer Relation wird automatisch die Gegenrichtung mit gelöscht.

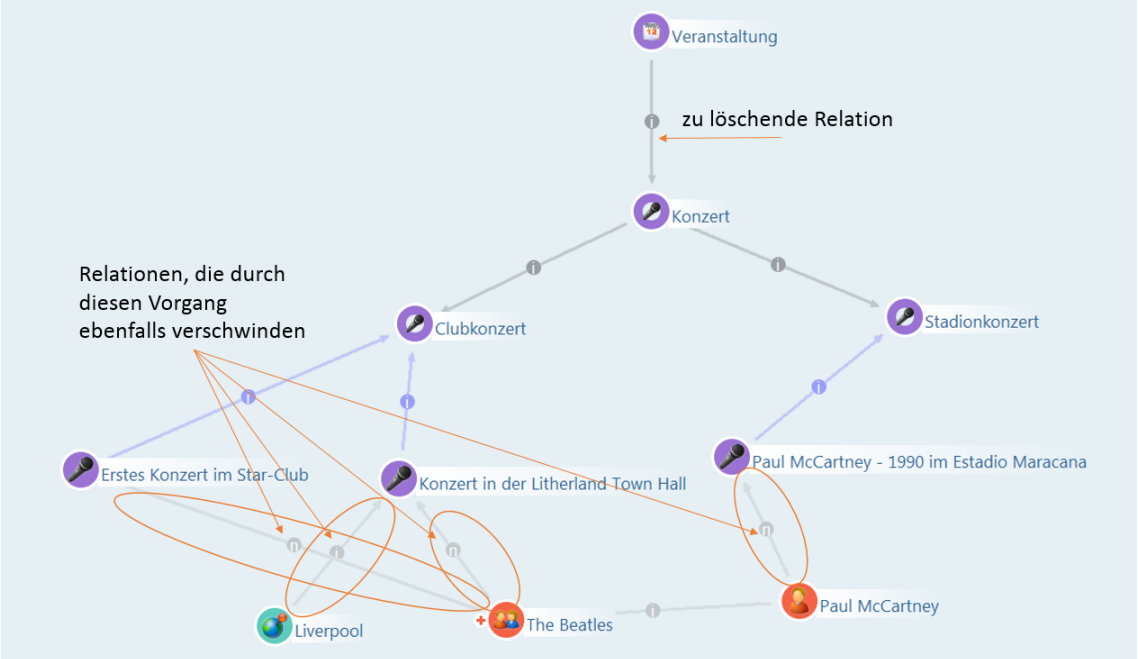
Da i-views immer dafür sorgt, dass die Objekte und Eigenschaften dem Schema entsprechen, ist das Löschen eines Objekttyps ggf. eine Operation mit weitreichenden Konsequenzen: Wenn ein Objekttyp gelöscht wird, werden alle seine konkreten Objekte gelöscht - analog bei Beziehungs- und Attributtypen.

Dabei informiert i-views immer über die Konsequenzen einer Operation. Wenn ein Objekt gelöscht werden soll, listet i-views alle Eigenschaften, die damit wegfallen im Bestätigungsdialog der Löschoperation auf:



i-views kontrolliert, wo durch die Änderung Objekte, Beziehungen oder Attribute verloren gehen. Der Benutzer wird auf die Konsequenzen der Löschung hingewiesen.

Nicht nur das Löschen, auch das Umwandeln und Ändern der Typenhierarchie kann Konsequenzen nach sich ziehen. Etwa wenn Objekte Eigenschaften haben die bei Typwechsel oder bei Wechsel in der Vererbung nicht mehr dem Schema entsprechen.



Nehmen wir an, wir löschen die Relation „ist Obertyp von“ zwischen „Veranstaltung“ und

„Konzert“ und lösen damit den Objekttyp „Konzert“ und alle seine Untertypen aus der Vererbungshierarchie von Event heraus, um ihn z.B. unter „Werk“ zu hängen. Hier macht uns i-views darauf aufmerksam, dass die „hat Teilnehmer“ Relationen der konkreten Konzerte wegfallen würden. Diese Relation ist nämlich bei „Veranstaltung“ definiert und würde damit für die Konzerte nicht mehr gelten.

Es gibt Möglichkeiten zu verhindern, das durch Modelländerungen zu erhaltende Relationen wegfallen. Soll bspw. ein Objekttyp innerhalb der Typenhierarchie umziehen , muss zuerst das Schema der betroffenen Relation vorher angepasst werden.

Beispielsweise soll „Konzert“ in der Hierarchie nicht mehr unter „Veranstaltung“, sondern unter „Werk“ verortet werden. Hierzu wird der Relation „hat Teilnehmer“ eine zweite Quelle zugewiesen: das kann entweder der Objekttyp Konzert selbst sein, oder die neue Position „Werk“. Die Relation geht somit nicht verloren.

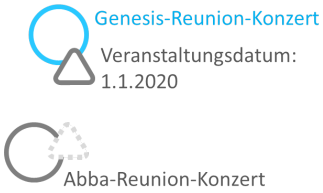
Die Typenhierarchie wird von i-views besonders berücksichtigt. Wenn wir aus der Mitte der Hierarchie einen Typ heraus löschen oder eine Ober-/Untertyp-Relation entfernen, dann schließt i-views die entstehende Lücke und hängt die Typen, denen der Obertypen abhanden gekommen ist, wieder in die Typenhierarchie ein - und zwar so dass die seine Eigenschaften möglichst weitgehend bewahrt bleiben.

Spezielle Funktionen

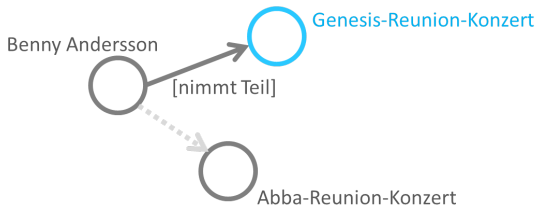
Typ wechseln: Bereits in der semantischen Graphdatenbank vorhandene Objekte lassen sich zu Objekten eines anderen Typs verschieben. Bspw. ist der Objekttyp „Veranstaltung“ ausdifferenziert in „Sportveranstaltung“ und „Konzert“. Sind bereits Objekte des Typs Sportveranstaltung oder Konzert in der semantischen Graphdatenbank vorhanden, können diese in der Liste im Hauptfenster selektiert und ganz einfach mit Drag&Drop unter einen neuen, passenderen Objekttyp verschoben werden.

Alternativ dazu finden wir im Kontextmenü den Punkt „Überarbeiten“.

Typ auswählen: Mit dieser Operation können wir eine Eigenschaft einem neuen Objekt zuweisen.



Relationsziel neu wählen: Bei Relationen gilt das nicht nur für die Quelle, sondern auch für das Relationsziel.

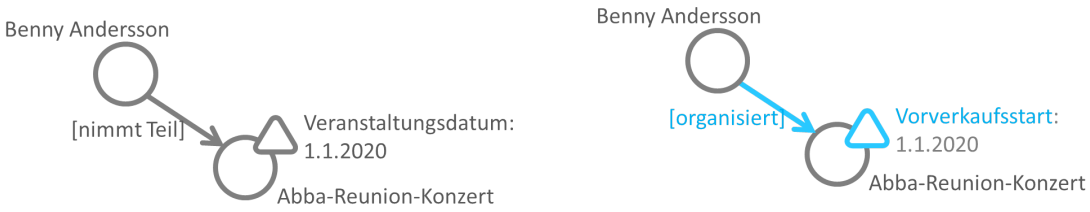


Untertypen in konkrete Objekte umwandeln (und umgekehrt): Die Grenze zwischen

Objekttypen und konkreten Objekten ist in vielen Fällen offensichtlich, aber nicht immer. Statt nur einen Objekttyp namens „Musikrichtung“ einzurichten, wie in unserem Beispielprojekt, hätten wir auch eine ganze Typenhierarchie von Musikrichtungen aufbauen können (Wir haben uns in diesem Netz dagegen entschieden, weil die Musikrichtungen so unterschiedliche Dinge wie Bands, Alben und Songs klassifizieren, und sie daher keine guten Typen abgeben). Es kann aber passieren, dass wir uns mitten in der Modellierung um entscheiden. Aus diesem Grund gibt es die Möglichkeit, Untertypen in konkrete Objekte umzuwandeln und konkrete Objekte in Untertypen. Eventuell bestehende Relationen gehen dabei verloren, falls sie nicht zum neuen Schema passen.

Beziehung umwandeln: Quelle und Ziel der Relation bleiben bestehen, nur der Beziehungstyp wird umgewandelt.

Attribut umwandeln: Quelle/Objekt bleibt erhalten, aber es wird ein anderer Attributtyp zugeordnet:



Bei Umwandlung der einzelnen Relation sind wir i.d.R. schneller, wenn wir diese löschen und durch eine andere ersetzen. Jetzt kann es aber vorkommen, dass an den Eigenschaften Meta-Eigenschaften hängen, die wir nicht verlieren möchten. Zum Andern stehen die Umwandeln-Operationen auch für alle Eigenschaften eines Typs bzw. eine Auswahl davon zur Verfügung. Voraussetzung ist natürlich, dass der neue Relations- oder Attributtyp für die Quell- und Zielobjekte auch definiert ist.

Bei Änderungen am Schema ist Grundsätzlich zu bedenken, das eine Wiederherstellung eines vorherigen Zustandes nur durch Einspielen eines Backups möglich ist. Analog zu relationalen Datenbanken existiert keine „Rückgängig“-Funktion.

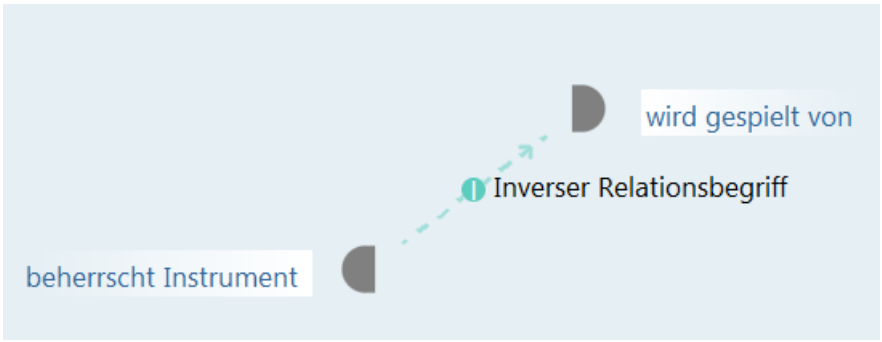
1.2.4 Darstellung von Schema im Graph-Editor

Bisher hatten wir es im Graph-Editor hauptsächlich die Vernetzung konkreter Objekte zu tun. Sich solche konkreten Beispiele vor Augen zu führen, mit anderen zu diskutieren und sie ggf. zu bearbeiten ist auch die Hauptfunktion des Graph-Editors. Wir können aber mit dem Graph-Editor auch das Schema des semantischen Netzes direkt darstellen, beispielsweise die Typenhierarchie eines Netzes.

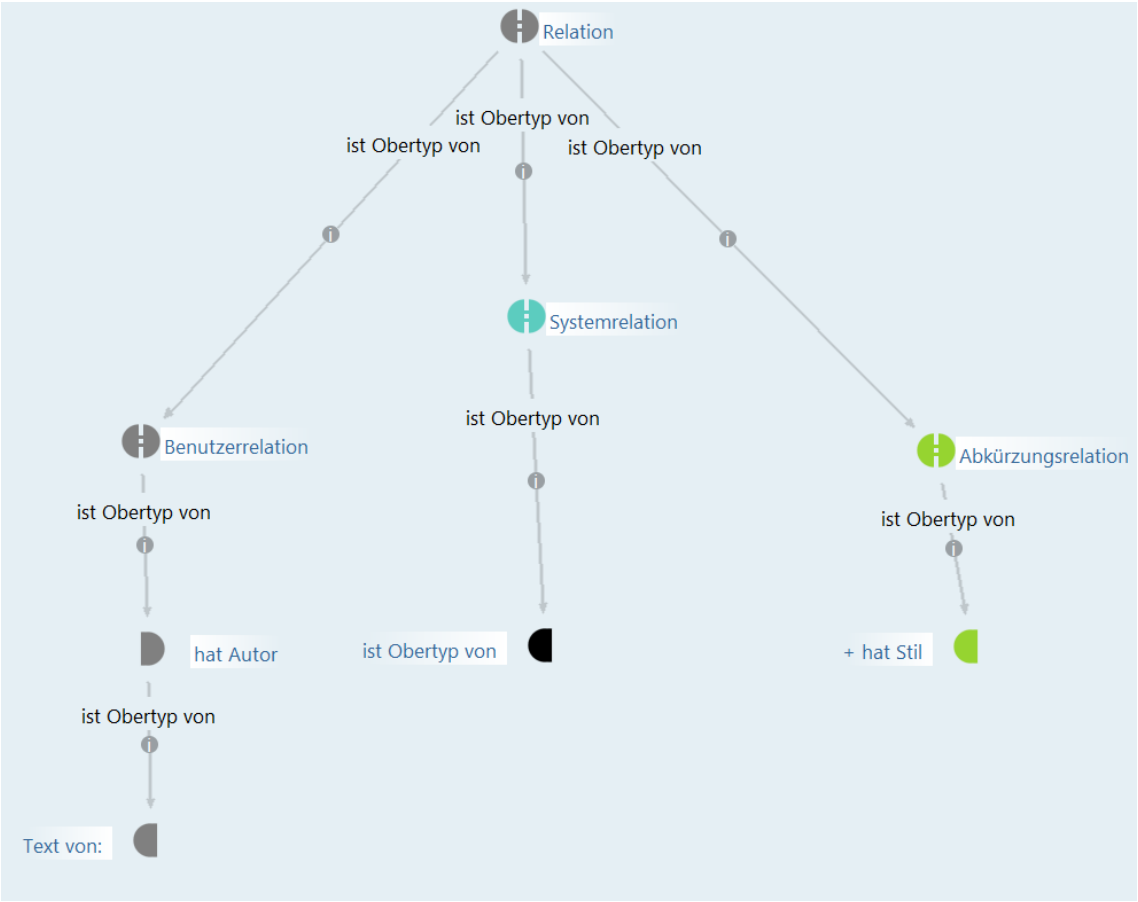
Typen von Objekten werden dabei als farbig hinterlegte Knoten, Typen von Relationen werden als gestrichelte Linie dargestellt:

Relationsbegriffe im Graph-Editor

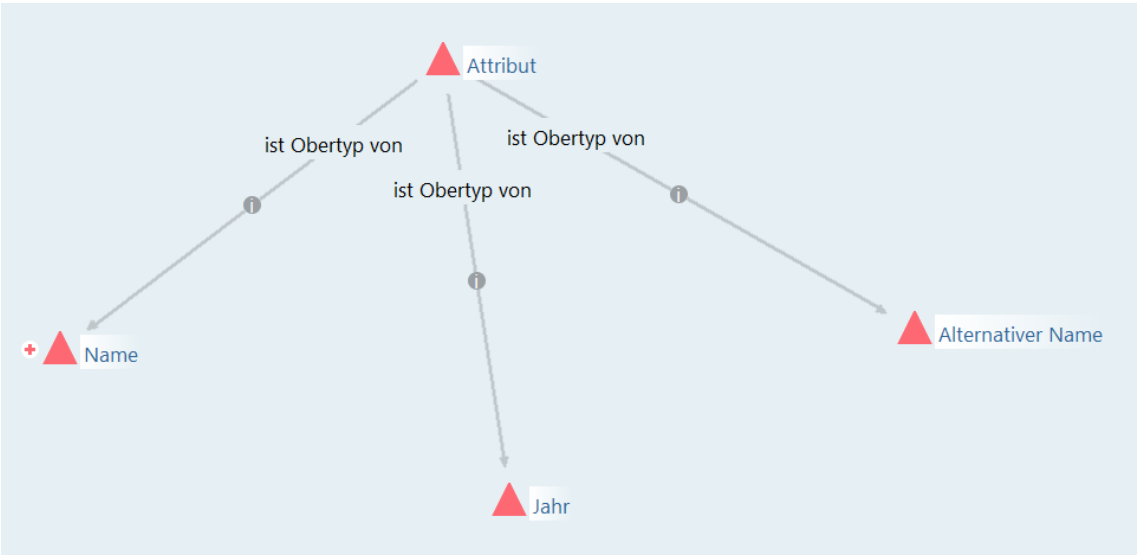
Wenn bisher von Relationen im Graph-Editor die Rede war, so waren damit Relationsobjekte zwischen bestimmten Objekten der semantischen Graph-Datenbank gemeint. Aber auch die allgemeinen Relationstypen (also die Schemata der Relationen) lassen sich im Graph-Editor darstellen. Eine Relation wird im Graph-Editor durch zwei Halbkreise dargestellt, die die beiden Richtungen (Hauptrichtung und inverse Richtung) repräsentieren. Zwischen diesen beiden Knoten besteht also die Beziehung „Inverser Relationstyp“:



Die Darstellung eines Relationsbegriffs und der Hierarchie im Graph-Editor kann analog zum Objekteditor mit allen Ober- und Untertypen dargestellt werden:



Auch Attributtypen lassen sich im Graph-Editor anzeigen - Sie werden durch dreieckige Knoten repräsentiert.



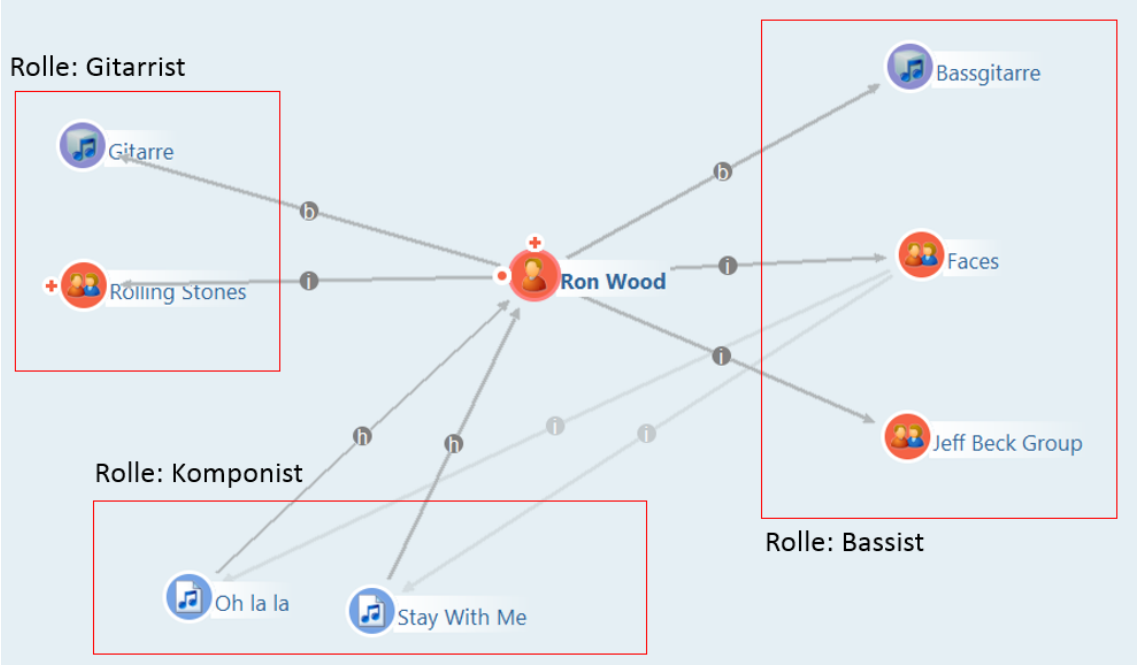
Analog zur Objekttyp-Hierarchie kann auch die Hierarchie der Relationen und Attribute im Graph-Editor durch Löschen und Ziehen der Obertyp-Relation verändert werden.

1.2.5 Metamodellierung und fortgeschrittene Konstrukte

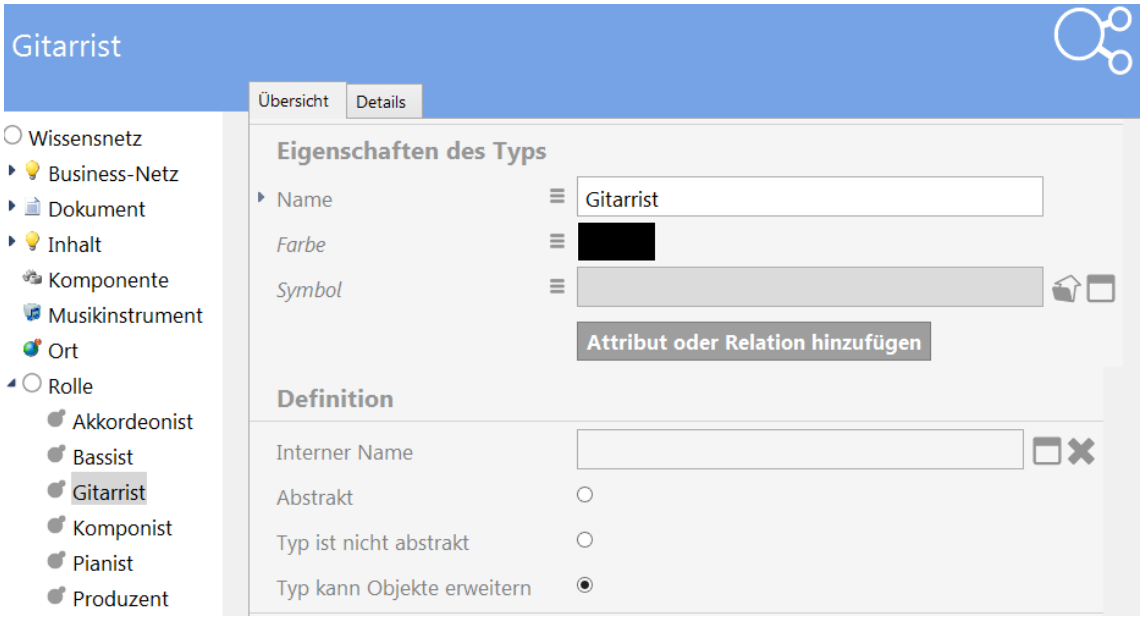
1.2.5.1 Erweiterungen

i-views bietet als weiteres Modellierungsmittel die Möglichkeit, Objekte zu erweitern.

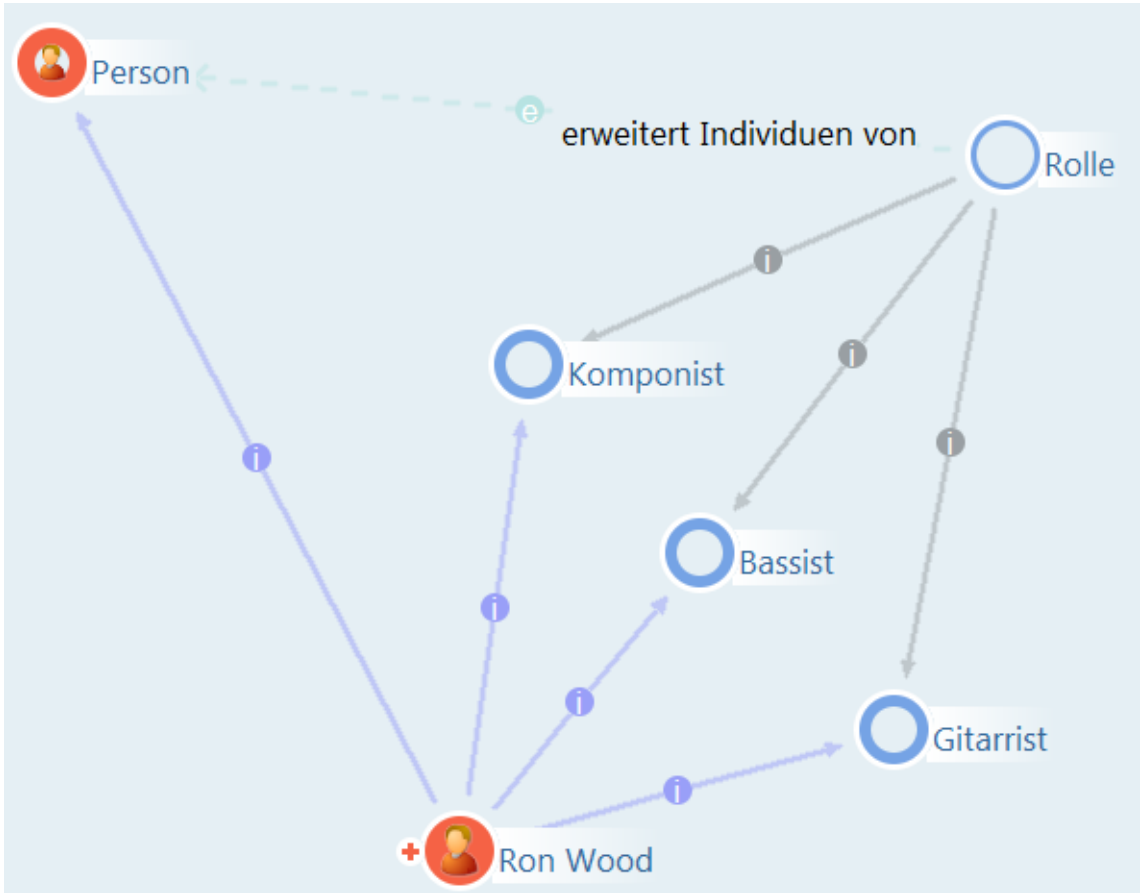
Beispielweise tritt eine Person in der Rolle als Gitarrist in einer Band auf, spielt aber in einer anderen Band ein wiederum anderes Instrument. Zusätzlich übt die Person noch die Rolle des Komponisten aus.



Die verschiedenen Rollen einer Person können über eine spezielle Form eines Objekttyps abgebildet werden. Dieser kann keine Objekte enthalten, jedoch Objekte eines anderen Objekttyps (hier zum Beispiel „Person“) erweitern. Hierzu wird bspw. der Objekttyp „Rolle“ in die semantische Graph-Datenbank eingeführt, und die verschiedenen Rollen für Personen als Untertyp angelegt: Gitarrist, Komponist, Sänger, Bassist usw. Damit diese „Rollen-Objekttypen“ Objekte erweitern können, wird diese Funktion im Objekttyp-Editor definiert, indem „Typ kann Objekte erweitern“ ausgewählt wird:



Im Graph-Editor werden Erweiterungen durch eine blaue gestrichelte Linie dargestellt:

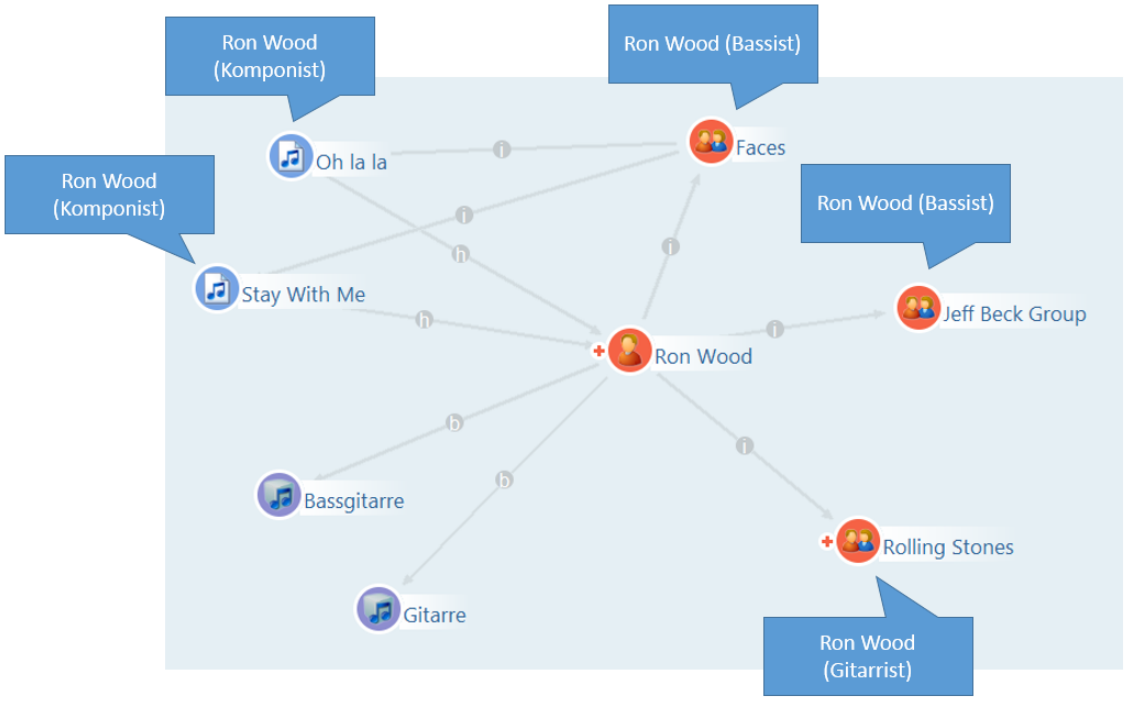


Mit dieser Erweiterung haben wir mehrere Dinge gleichzeitig erreicht:

- Wir haben Subobjekte für die Personen gebildet (können wir uns auch als Abschnitte vorstellen oder - bei Personen - als Rollen). Diese Subobjekte können einzeln betrachtet

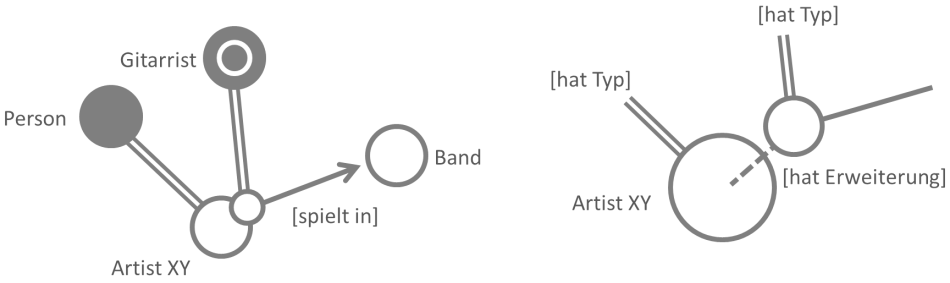
und abgefragt werden. Sie sind unselbstständig, wenn die Person gelöscht wird, ist auch die Erweiterung „Gitarrist“ mitsamt den Relationen zu den Bands oder Titeln weg.

- Wir haben einen mehrstelligen Sachverhalt ausgedrückt. Über separate Relationen zwischen Person, Instrument, Titel/Band können wir das nicht ausdrücken - hier würde die Zuordnung nicht mehr gelingen.



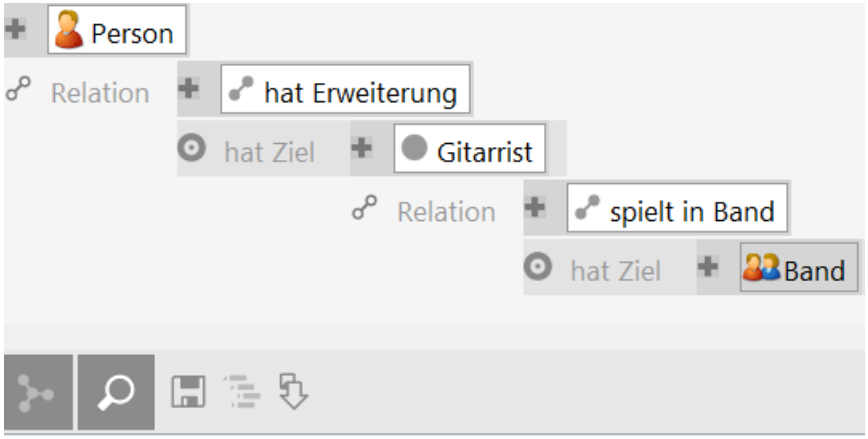
Dazu muss die Relation „spielt in Band“ bei der Erweiterung „Gitarrist" definiert sein. Dieser Effekt, dass Personen über die Erweiterung zusätzliches Schema erben, kann auch unabhängig von mehrstelligen Sachverhalten hilfreich sein.

Technisch gesehen ist die Erweiterung ein unselbstständiges Objekt, das durch die Systemrelation „hat Erweiterung“ oder invers „erweitert Individuum“ mit dem Kernindividuum verbunden ist. Sein Typ (Systemrelation „hat Typ“) bildet den Erweiterungstyp.



Bei der Definition einer neuen Erweiterung sind spielen zwei Objekttypen eine Rolle: in unserem Beispiel wollen wir Personen eine Erweiterung geben, das müssen wir ihrem Typ „Person“ mitteilen. Die Erweiterung selber hat auch wieder einen Objekttyp (meist sogar eine ganze Menge von Objekttypen); in unserem Fall „Gitarrist“. Beim Typ „Gitarrist“ (und bei allen anderen mit denen wir Personen erweitern wollen) werden seine konkreten Objekte unselbstständig sein.

Beim Abfragen von Erweiterungen in der Struktursuche müssen wir die einzelnen Relationen traversieren. Von der konkreten Person über die Relation „hat Erweiterung“ über das Erweiterungsobjekt „Gitarrist“. Von dort aus lässt über die Relation „spielt in Band“ zur Band gelangen.



Name
Ron Wood

Mix-In

Die Essenz dieses Beispiels mit der Rolle „Gitarrist“ ist, dass die Relation „spielt in Band“ an der Erweiterung, jedoch nicht mit der Person verknüpft ist. Somit wird auch bei mehreren Instrumenten und mehreren Bands einen konsistente Zuordnung möglich ist.

Ist die Option Mix-In angewählt ist, wird die Relation dagegen beim Kernobjekt (Person) selbst angelegt. Grund dafür ist, das Erweiterungen gelegentlich nicht benutzt werden um komplexere Sachverhalte auszudrücken, sondern um ein Objekt polyhierarchisch an verschiedene Typen zuzuordnen. Dieses Objekt erbt auf diese Weise Relationen und Attribute mehrerer Typen.

Wenn wir bspw. eine umfangreiche Typen-Hierarchie von Veranstaltungen aufbauen, mit der Unterteilung in große und kleine Veranstaltungen, Freiluft- und Hallenveranstaltungen, Sport- und Kulturveranstaltungen, können wir entweder alle Kombinationen ausprägen (großes Freiluftkonzert, kleines Hallenfußballturnier etc.) oder legen die verschiedenen Veranstaltungstypen als mögliche Erweiterungen der Objekte vom Typ „Veranstaltung“ an. Dann können wir eine Veranstaltung über ihre Erweiterungen als Fußballturnier und gleichzeitig als Freiluft-Veranstaltung sowie als Großveranstaltung einordnen. Über die Erweiterung „Fußballturnier“ wird dann vielleicht die Relation „teilnehmende Mannschaft“ geerbt, über die Erweiterung „Freiluft-Veranstaltung“ bspw. noch die Eigenschaft „Flutlicht vorhanden“. Wenn wir diese Eigenschaften auf Mix-In gesetzt haben, können sie bei den Veranstaltungen wie direkte Eigenschaften abgefragt werden.

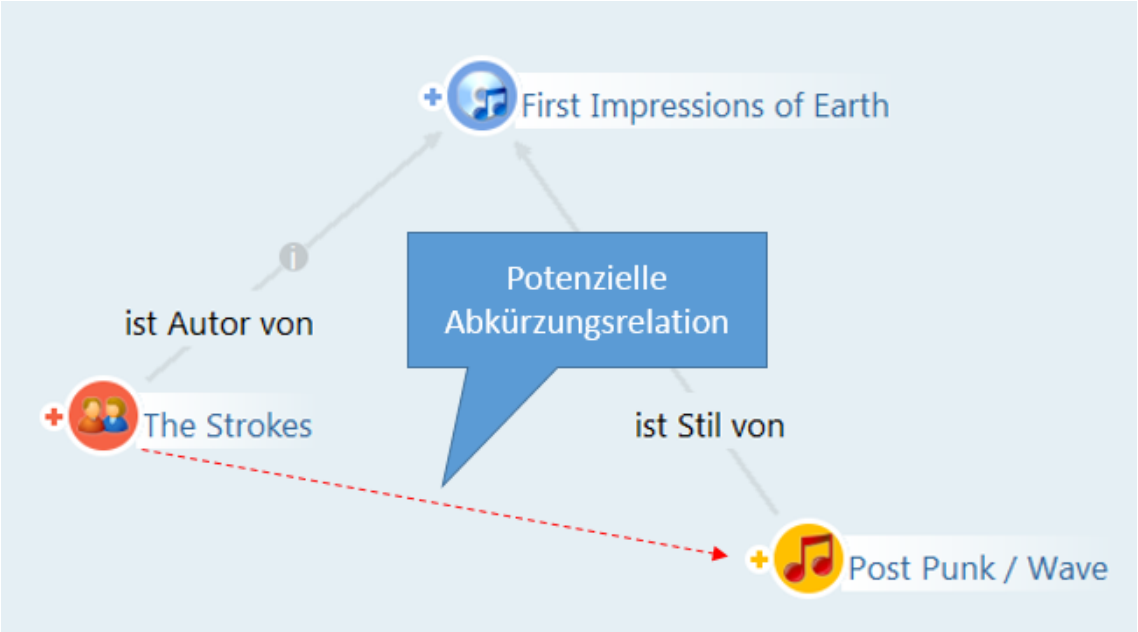
Wird eine Mix-In Erweiterung gelöscht, verhält sie sich wie eine „normale“ Erweiterung: Es muss mindestens eine Erweiterung vorhanden sein, die die Mix-In-Eigenschaft mit sich bringt. Wenn die letzte dieser Erweiterungen gelöscht wird, wird auch die Relation oder das Attribut beim Kernobjekt gelöscht.

1.2.5.2 Berechnete Relationen

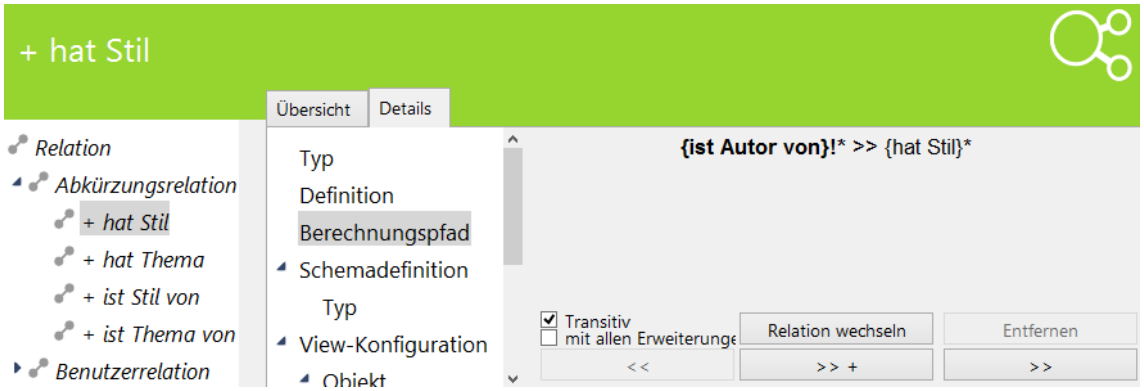
Eine spezielle Form der Relation ist die berechnete Relation. Dahinter verbirgt sich die Möglichkeit, mehrere bereits vorhandene und definierte Relationen, die in einer semantis-

chen Graph-Datenbank in hintereinandergeschalteter Form vorliegen, durch eine geeignete Relation abzukürzen. Auf diese Weise kann das System in gewissem Rahmen von einem Objekt A der semantischen Graph-Datenbank, das über mehrere Knoten mit einem anderen Objekt B verbunden ist, einen direkten Schluss von A auf B ziehen.

Beispielsweise veröffentlicht eine Musikgruppe einen Tonträger in einem bestimmten Musikstil, ergo kann dieser Musikstil ebenfalls der Musikgruppe zugewiesen werden:



Im Formular-Editor wird der Berechnungspfad über die Relationen „ist Autor von“ und „hat Stil“ definiert.



Einstellungsmöglichkeiten bei der Definition des Berechnungspfades:

- "Transitiv": Die Relation darf beliebig oft auftreten (ein bis unendlich).
- "mit allen Erweiterungen": Die Erweiterungen werden berücksichtigt. (Die Einstellung wird an der Relation ausgewählt, die an der Erweiterung definiert ist. Möchte man einen Berechnungspfad definieren, in dem man von der Erweiterung zurück zum Kern der Erweiterung laufen möchte, so muss die Relation "erweitert Objekt" explizit im Berechnungspfad angegeben werden.)

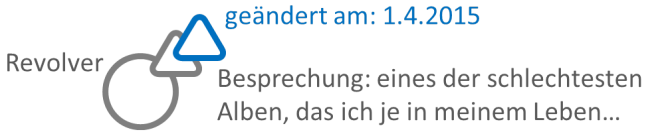
In den Abfragen kann die berechnete Relation benutzt werden wie jede andere Relation auch.

Anmerkung: In der aktuellen Version von i-views wird wegen der besseren Übersicht bei Strukturabfragen empfohlen, mehrere Knoten und Kanten über Suchbausteine abzufragen.

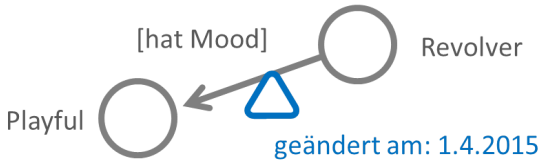
1.2.5.3 Meta-Eigenschaften

In Kapitel 2.1 wurden Eigenschaften von geringer Komplexität bei Objekttypen für Objekte definiert. Beispielsweisen können Anwender über eine Webanwendung in der hier als Beispiel behandelten Musik-Datenbank Inhalte hinzuzufügen oder zu editieren. Es soll aber festhalten werden, von wem welche Information zu welchem Zeitpunkt geändert wurde. Dazu werden Attribute und Beziehungen wiederum für Attribute und Beziehungen, in allen Kombinationen benötigt.

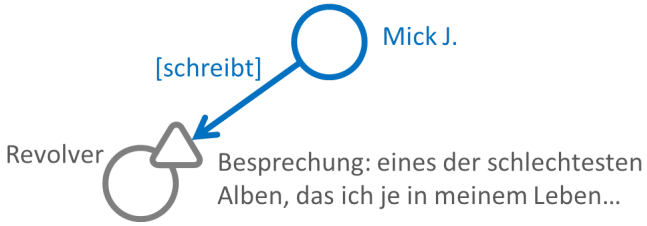
Attribute auf Attributen: Beispielsweise sind für Musik-Alben in der Musik-Datenbank Besprechungen als Textattribute hinterlegt. Soll festhalten wollen, wann die Besprechung hinzugefügt wurde oder wann das letzte Mal geändert, können wir ein Datumsattribut definieren, das den Besprechungs-Attributen angegliedert wird:

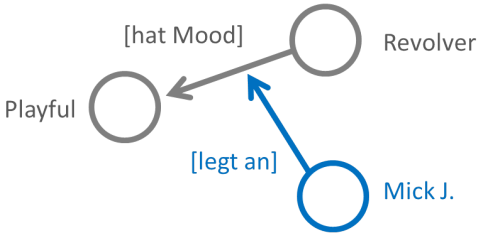


Attribute auf Relationen: Dieses Datumsattribut kann sich auch auf einer Relation zwischen Alben und persönlichen Stimmungen, etwa „Moods“ befinden, wenn den Nutzern die Möglichkeit zum Tagging gegeben wurde:



Relationen lassen sich auf Attribute und auf Relationen anwenden. Bspw. sollen Nutzer protokolliert werden, die zu bestimmten Zeitpunkten Attribute (etwa eine Besprechung eines Albums), oder eine Relation zwischen einem Album und einem Mood erzeugt oder geändert haben:





Diese Beispiele wird mit den Bearbeitungsinformationen bilden eine klar abgegrenzte Meta-Ebene. Eigenschaften auf Eigenschaften sind aber auch für komplexe „Primärinformationen“

verwendbar:

Soll bspw. die Zuordnung von Bands oder Titeln zu den Genres gewichtet werden, kann ein Wert als „Gewicht“ als Attribut an die Relation vergeben werden.

Ein Attribut einer Relation kann aber auch der Betrag einer Überweisung, oder die Dauer einer Teilnahme oder Mitgliedschaft sein.

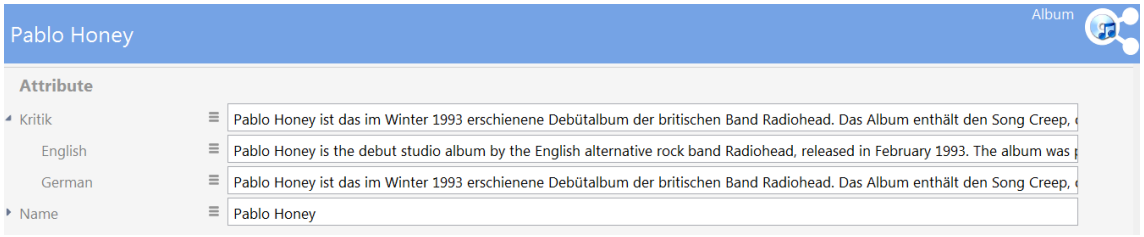
Mit Relationen auf Relationen lassen sich ebenso „mehrstellige Sachverhalte“ ausdrücken. Bspw. die Tatsache, dass eine Band bei einem Festival auftritt (das ist eine Relation) und sich dabei einen Gastmusiker dazu holt. Der spielt nicht immer bei der Band und hat somit keine direkte Relation zu ihr. Ebenfalls kann er auch nicht pauschal dem Festival zugewiesen werden, sondern wird der Auftrittsrelation zugewiesen.

Das modellieren von Meta-Eigenschaften lässt sich natürlich auch dadurch realisieren, dass zusätzliche Objekte eingeführt werden. Im letzten Beispiel ließe sich die Tatsache, dass die Band bei einem Festival auftritt, ebenfalls als Objekt vom Typ „Auftritt“ modellieren. Ein wesentlicher Unterschied besteht darin, dass im Metamodell die Primärinformationen einfach von der Meta-Ebene getrennt werden können: der Graph-Editor zeigt die Meta-Informationen erst auf Anforderungen, und in Abfragen und in der Definition von Sichten kann die Meta-Information einfach weggelassen werden. Der zweite Unterschied liegt im Löschverhalten: Objekte sind eigenständig lebensfähig. Eigenschaften, auch Meta-Eigenschaften wiederum nicht; wenn Primärobjekte und ihre Eigenschaften gelöscht werden, werden die Meta-Eigenschaften ebenfalls gelöscht.

Übrigens: Eigenschaften können nicht nur für konkrete Objekte sondern auch für die Typen selbst definiert werden. Ein typisches Beispiel dafür ist eine ausführliche schriftliche Definition bei einem Objekttyp, bspw. „was verstehen wir unter einer Firma?“ Deswegen werden wir beim Anlegen neuer Eigenschaften immer wieder gefragt ob wir sie für konkrete Objekte oder Untertypen anlegen wollen.

1.2.5.4 Mehrsprachigkeit

Die Attribute „Zeichenkette“, „Datei-Attribut“ und „Auswahl“ können mehrsprachig angelegt werden. Beim Zeichenketten-Attribut und bei Dateien können dann mehrere Zeichketten für ein Attribut eingegeben werden:



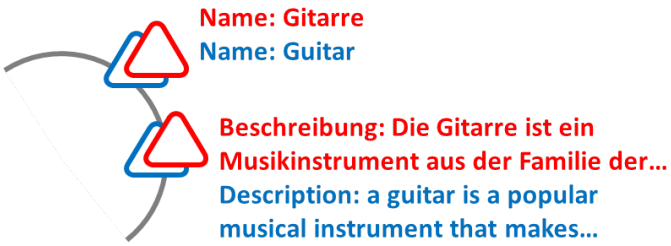
Bei Datei-Attributen können analog mehrere Bilder (z.B. mit anderssprachigen Beschriftungen) hochgeladen werden. Bei Auswahl-Attributen werden alle Auswahlmöglichkeiten in der Attributdefinition hinterlegt; hier ist es egal, in welcher Sprache dann die Auswahl für das konkrete Objekt getroffen wird.

Alle anderen Attribute stellen sich in allen Sprachen gleich dar, wie z.B. Boolesche Attribute, Ganzzahlen oder URLs.

Sofern die Darstellung in anderen Sprachen abweicht, passen Attribute ihre Darstellung je nach Sprache automatisch an: Etwa werden Datumsangaben nach europäischer Schreibweise Tag|Monat|Jahr im US-amerikanische Format Monat|Tag|Jahr dargestellt.

In i-views werden für anderssprachige Werte nicht einfach separate Attribute angelegt, son-

dern es bleibt bei einen Attribut mit Sprachvarianten als separate Layer. Es muss bei Entwicklung einer Anwendung nicht um das Management der verschiedenen Sprachen gekümmert werden, sondern nur die gewünschte Sprache bei der jeweiligen Anfrage:



In i-views lassen sich bevorzugte Ersatzsprachen definieren: sollte ein Attributwert, z.B. ein Beschreibungstext in der angefragten Sprache nicht vorliegen, kann der fehlende Text, wenn er in anderen Sprachen vorliegt angezeigt werden. Die Reihenfolge der Ersatzsprachen lässt sich ebenfalls festlegen.

Verwendet werden die Mehrsprachigkeitseinstellungen beispielsweise in der Suche.

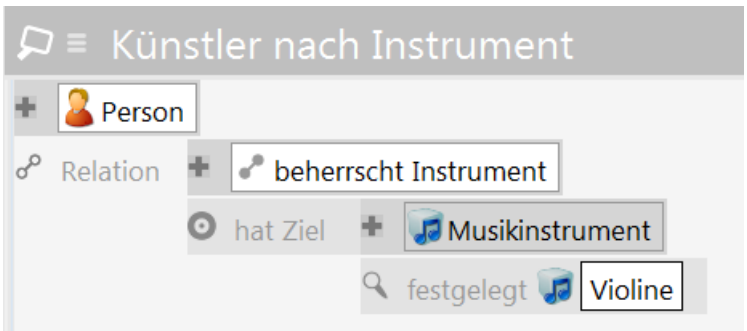
1.3 Suchen / Abfragen

Bei Abfragen des semantische Netzes unterscheiden wir zwischen verschiedenen Teilaufgaben: Gelegentlich möchten wir eine Eingabe des Nutzers über ein Suchfeld (Zeichenkette) verarbeiten. Meistens möchten wir bestimmte Pfade im semantischen Netz herausfiltern, manchmal wollen wir dabei Gewichte vergeben. In i-views stehen dazu verschiedene Arten von Suchen zur Verfügung:

- Strukturabfragen
- Direkte Abfragen
- Such-Pipeline-Abfragen

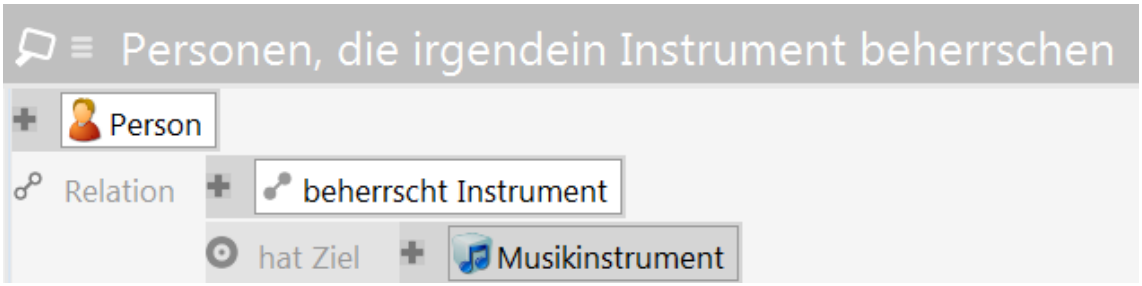
1.3.1 Strukturabfragen

Mit Strukturabfragen können Objekte zusammengesucht werden, die bestimmte Bedingungen erfüllten. Ein einfaches Beispiel für eine Strukturabfrage ist folgende: Es sollen alle Personen gefiltert werden, die ein bestimmtes Instrument beherrschen.

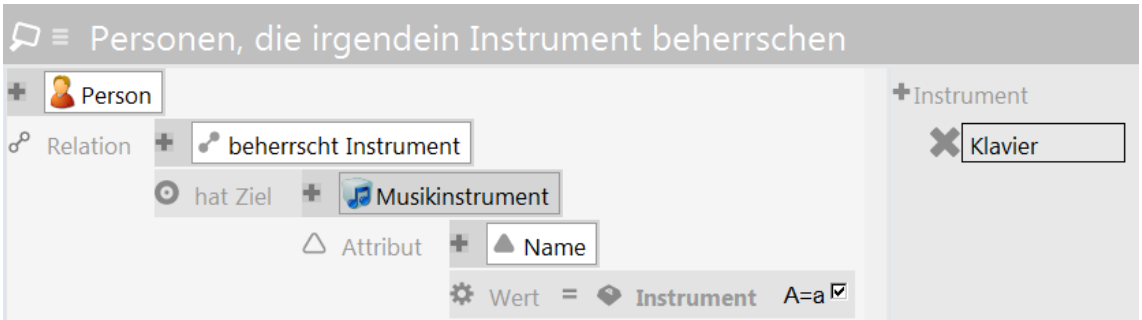


Als erstes kommt die Typbedingung: Es werden Objekte des Typs Person gesucht. Die zweite Bedingung: die Personen müssen ein Instrument beherrschen. Dritte Bedingung: dieses Instrument muss die Violine sein.

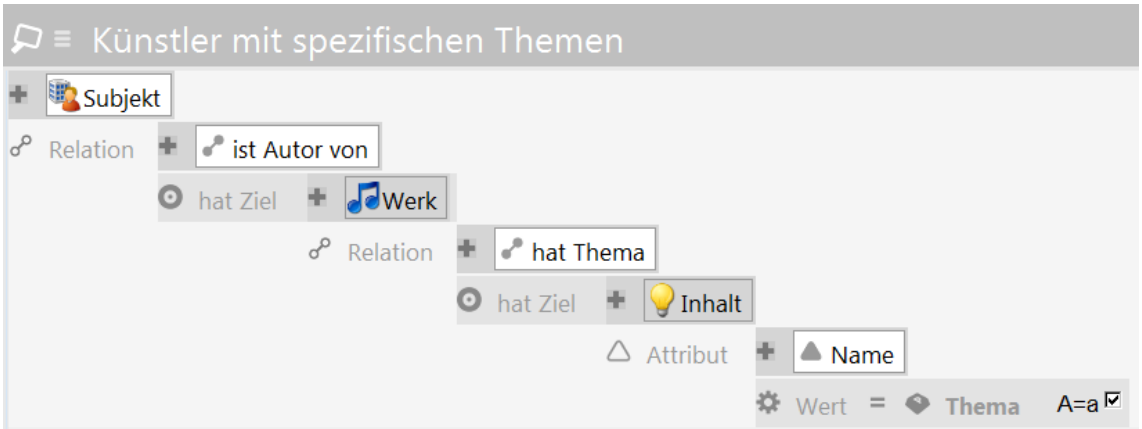
In der Strukturabfrage bilden die Relation „beherrscht Instrument“ und ihr Ziel „Violine“ zwei verschiedene Zeilen und somit auch zwei Suchbedingungen. Die zweite Bedingung, dass das Instrument eine Violine sein muss, kann optional auch weggelassen werden. In der Treffermenge werden dann alle Personen zu finden sein, welche ein beliebiges Instrument spielen:



Oft sollen Bedingungen (hier das Instrument) nicht schon vorher festlegt, oder aber vollständig zugelassen werden. Je nach Situation lässt sich in der Anwendung ein Instrument als Parameter mitgeben:



Die Bedingungen können dabei beliebig komplex werden und das Netz beliebig weit traversieren:



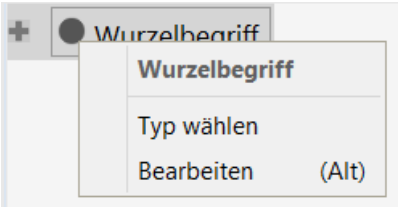
Geringfügig komplexeres Beispiel: Personen oder Bands, die in ihren Songs (genauer gesagt in mindestens einem) ein bestimmtes Thema behandeln. Hier wird der Suche nicht der Name, sondern die ID des Themas als Parameter mitgegeben - typisch für Suchen, die z.B. über einen REST-Service von der Anwendung aus aufgerufen werden [Bild - statt „Name“ „ID“]

Die Typhierarchien werden in den Strukturabfragen automatisch mit berücksichtigt: Die

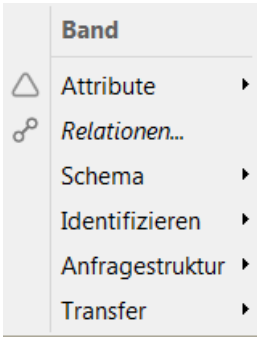
Typbedingung „Werk“ in der Suche oben schließt seine Untertypen Alben und Songs mit ein. Auch die Relationshierarchie wird berücksichtigt: Wenn es unterhalb von „ist Autor von“ noch eine Differenzierung gibt (z.B. „schreibt Text“ oder „schreibt Musik“) werden beide Unterrelationen mit in die Suche einbezogen. Das gleiche gilt für die Attributtyp-Hierarchie.

Interaktion

Wird eine neue Strukturabfrage angelegt, ist per Default erst einmal der oberste aller Typen eingetragen. Um die Abfrage weiter einzuschränken kann der Namen einfach überschrieben , oder über Klick auf das Icon über „Typ ändern“ auswählt werden.



Die Schaltfläche lässt weitere Bedingungen zur Strukturabfrage hinzufügen. Das Löschen von Bedingungen geschieht jeweils am Anfang jeder Zeile, wo die Art der Bedingung aufgeführt ist (Relation, Attribut, Ziel etc.). Beim Klick auf die Schaltfläche erscheint folgendes Menu, das je nach Kontext leicht abweichen kann.



Von allen möglichen Bedingungen wurde bisher auf die allerersten Punkte im Menu fokussiert. Eine vollständige Erläuterung aller Bedingungen und Optionen der Strukturabfragen befinden sich in den nächsten Kapiteln.

1.3.1.1 Verwendung Strukturabfragen

Ein Hauptzweck der Strukturabfragen ist, in Anwendungen zu einem bestimmten Kontext Informationen zu liefern. Die Strukturabfrage aus dem letzten Abschnitt kann z.B. dem Endnutzer in einem Musikportal zu einem Thema wie Liebe, Drogen, Gewalt usw. eine Liste aller Künstler oder Bands generieren, die das Thema in ihren Liedern behandeln.

Dazu wird die Strukturabfrage in der Regel über ihren **Registrierungsschlüssel** in einen **REST-Service** eingebaut. Das Thema, für das sich der Nutzer gerade interessiert, geben wir der Abfrage mit seiner ID als Parameter mit.

Beispielszenario: Ein Nutzer sucht durch Eingabe eines Suchstrings nach einem Thema. Es liegt also keine ID vor, sondern nur einen String anhand dessen das Thema identifiziert werden soll. Dabei soll aber im Suchergebnis gleich angezeigt werden, welche Bands Songs zu dem Thema geschrieben haben. Zu diesem Zweck kann eine Strukturabfrage als eine Kom-

ponente in eine **Such-Pipeline** eingebaut werden - hinter die Abfrage, die den Suchstring verarbeitet.

Strukturabfragen sind unter anderem deshalb ein zentrales Werkzeug innerhalb von i-views, weil auch die Bedingungen für **Rechte und Trigger** mit Strukturabfragen formuliert werden: Angenommen es wird in einem Musikportal nur Künstlern und Bands erlaubt, Kommentare zu hinterlassen. Im Rechtesystem lässt sich entsprechend formulieren, dass nur Künstler und Bands, die zu einem bestimmten Thema mindestens einen Song geschrieben haben, zu diesem Thema Kommentare hinterlassen dürfen. Strukturabfragen können auch in Exporten benutzt werden, um zu bestimmen, welche Objekte exportiert werden soll.

Alle diese Verwendungen haben eines gemeinsam: wir sind nur an qualitativen, keinen gewichteten Aussagen interessiert. Das ist die Domäne der Strukturabfragen gegenüber den Such-Pipelines.

Nicht zuletzt sind Strukturabfragen auch Hilfsmittel für uns Knowledge-Engineers. Mit ihnen können wir uns einen Überblick über das Netz verschaffen und **Reports** sowie **To-do-Listen** zusammenstellen. Beispielfragen, die mithilfe von Strukturabfargen beantwortet werden können, sind:

- Zu welchen Themen gibt es wie viele Künstler/Bands?
- Müssen bestimmte Themen ausgebaut werden weil sich zu viele Relationen ansammelt haben, oder sollten umgekehrt spärlich besetzte Themen zusammenlegt oder geschlossen werden?

Für diese Verwendung ist es sinnvoll, die Strukturabfragen in **Ordnern** organisieren zu können.

Ausführen

Durch einen Klick auf die Schaltfläche *Suchen* über dem Fenster für die Trefferliste wird die Strukturabfrage ausgeführt.

Das Suchergebnis kann dann weiter bearbeitet (z.B. mithilfe des Speichersymbols in einen neuen Ordner kopiert) werden.

Den Weg, den die Strukturabfrage genommen hat, kann zur Rückverfolgung der Suchergebnisse im Graph-Editor betrachtet werden. Damit die entsprechende Schaltfläche angeklickt werden kann, müssen die zu betrachtenden Suchergebnisse zunächst selektiert werden.

Eine Strukturabfrage kann kopiert werden, etwa um verschiedene Varianten zu erstellen. Ebenfalls besteht die Möglichkeit sie auch unabhängig vom Netz im XML-Format zu speichern. Die Strukturabfrage ließe sich somit in ein anderes Netz importieren. Das beschränkt sich aber auf Versionen desselben Netzes, z.B. auf Sicherheitskopien, da die Strukturabfrage Objekttypen, Relations- und Attributtypen über ihre internen IDs referenziert.

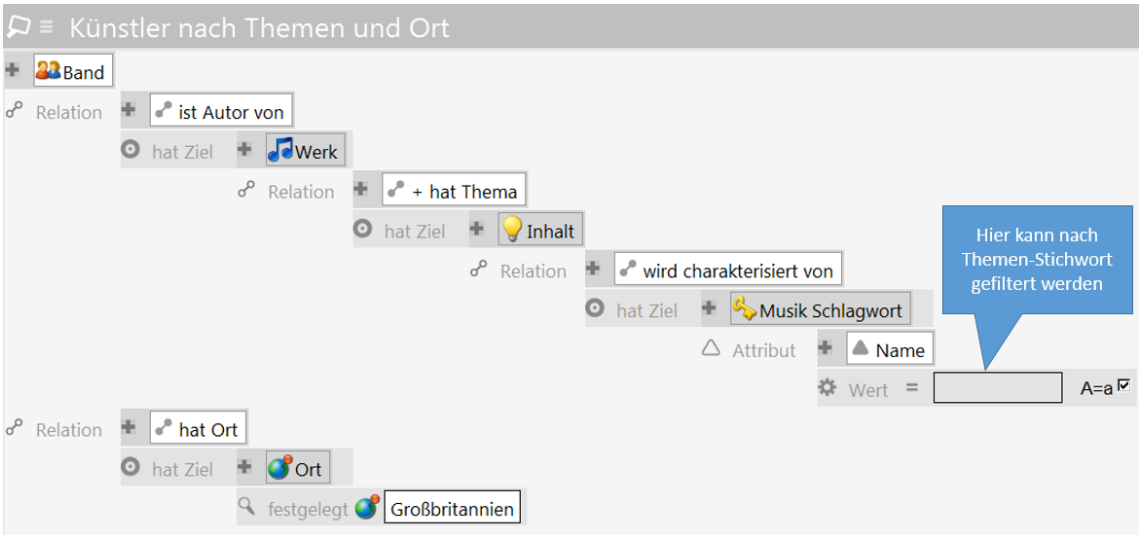
1.3.1.2 Aufbau von Strukturabfragen

Innerhalb von Strukturabfragen lassen sich sehr indirekte Bedingungen formulieren: durch die Struktur der semantischen Graph-Datenbank kann beliebig zwischen den Elementen

traversiert werden. Es lassen sich Künstler oder Bands heraussuchen, die Songs zu bestimmten Themen geschrieben haben, welche wir jedoch nicht konkret mit Titel benennen können.

1.3.1.2.1 Mehrere Bedingungen

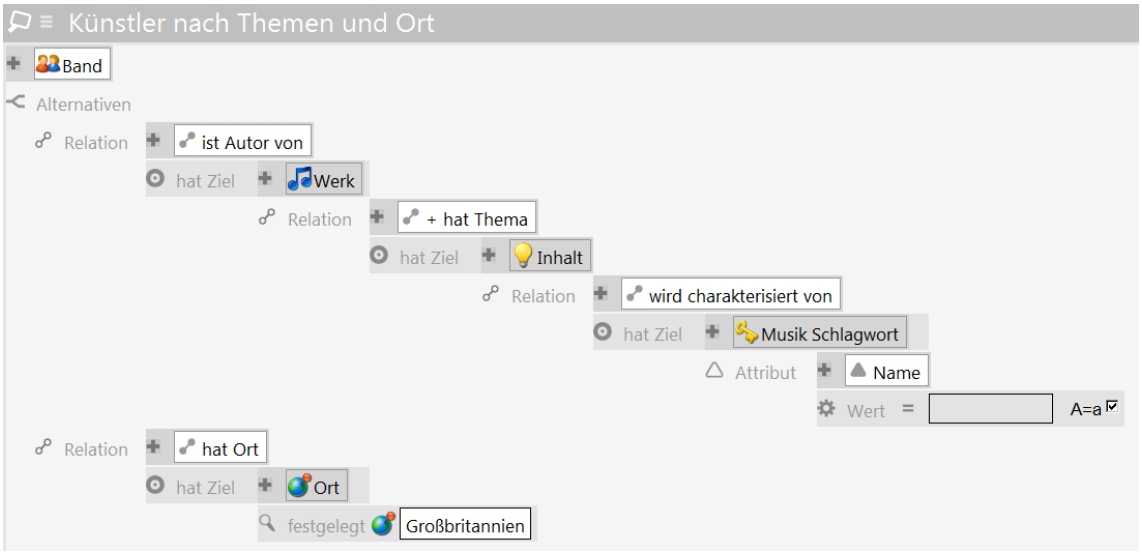
Entweder können Bedingungsketten beliebig tief sein, oder es lassen sich mehrere parallele Bedingungen formulieren: an einem beliebigen Bedingungselement werden zusätzliche Bedingungen als weiterer Ast angefügt:



Mehrere Bedingungen: Englische Bands mit Songs über ein bestimmtes Thema

1.3.1.2.2 Alternative Bedingungen

Im oben genannten Beispiel werden nur Künstler oder Bands gefunden, die sowohl Songs zu einem festgelegten Thema geschaffen haben, als auch aus England stammen. Wenn wir stattdessen alle Künstler und Bands finden wollen, auf die eine der beiden Bedingungen zutrifft, werden sie als Alternative formuliert. Durch anklicken des Symbols  der Bedingung in Form der Relation „ist Autor von“ kann dort im Menü eine Alternative gewählt werden:



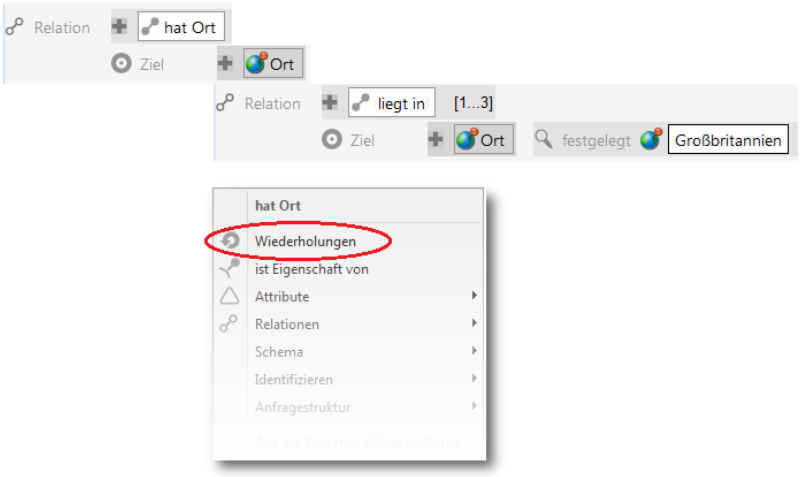
Alternative Bedingungen - Die Band muss entweder englisch sein oder Songs zu einem bestimmten Thema haben

Sind weitere Bedingungen außerhalb der Alternativ-Klammer vorhanden, befinden sich Objekte in der Treffermenge die **eine** der Alternativen, sowie **alle weiteren** Bedingungen erfüllen.

1.3.1.2.3 Transitivität / Wiederholungen

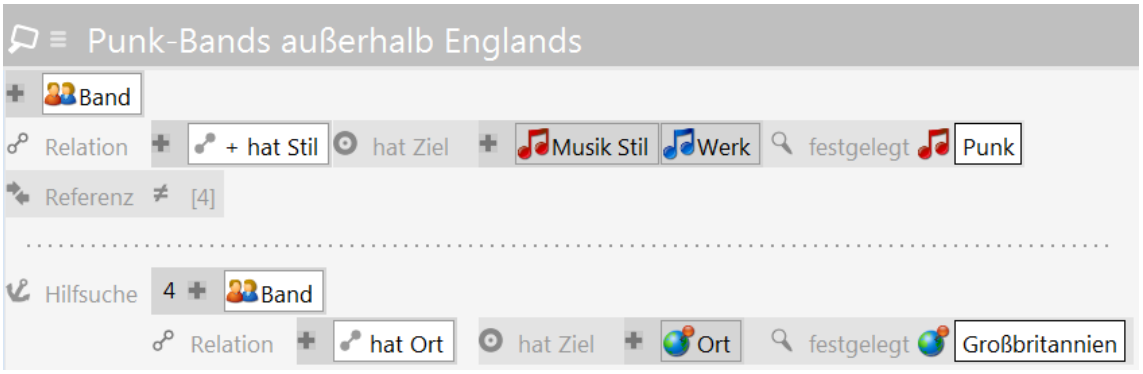
Angenommen, im Netz sind die Bands entweder Städten oder Ländern zugeordnet. Von diesen wiederum ist bekannt, welche Städte in welchen Ländern liegen. Um diesen Sachverhalt in der Suche zu erfassen, ließe sich die Bedingungskette einfach erweitern: wir könnten etwa nach Bands suchen, die einer Stadt zugeordnet sind, die wiederum in England liegt. Auf diese Weise werden jedoch die Bands nicht gefunden, welche direkt England zugeordnet sind. Um das zu vermeiden können wir bei der Relation „liegt in“ angeben, dass sie optional ist, also nicht vorliegen muss.

Gleichzeitig können wir mit der Funktion "Wiederholungen" auch Hierarchien berücksichtigen, die mehrere Ebenen tief sind. Beispielsweise ist von der Band ZZ Top bekannt, dass sie aus der Stadt Houston stammt. Diese Stadt liegt wiederum in Texas. Um die Band auch als Ergebnis zu erhalten, wenn nach Bands aus den USA gefragt wird, können wir bei der Relation "liegt in" angeben, dass bis zu n Wiederholungen der Relation nachgegangen werden soll:



1.3.1.2.4 Negativ-Bedingungen

Bedingungen lassen sich ebenso gezielt negieren. Beispielsweise sollen Punk-Bands gesucht werden, die jedoch nicht aus England kommen. Dazu wird die Negativ-Bedingung als eine sogenannte Hilfssuche aufgebaut

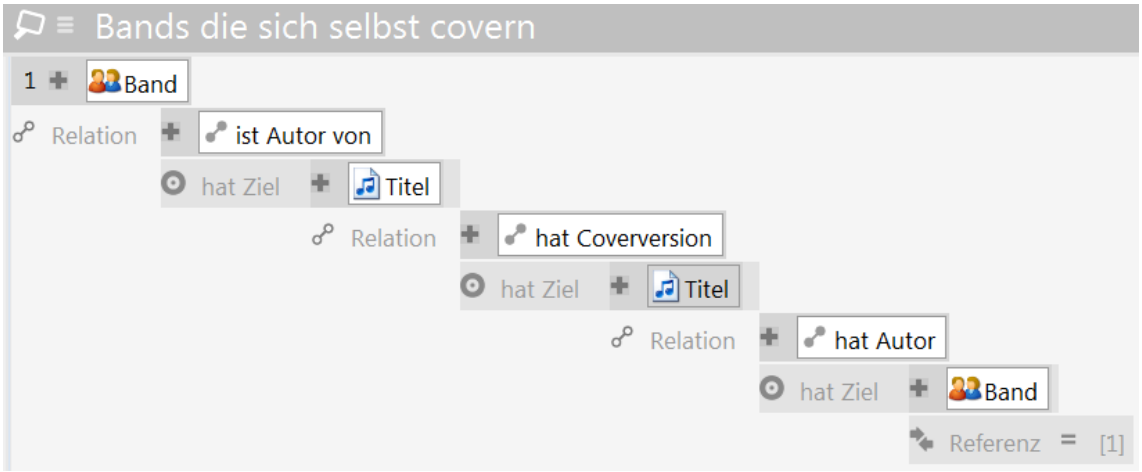


Die Hilfssuche liefert Bands auf England - von der Hauptsuche aus kann eine Referenz hergestellt und dabei angegeben werden, dass die Suchergebnisse den Kriterien der Hilfssuche gerade nicht entsprechen dürfen - damit ziehen wir die Ergebnisse der Hilfssuche von denen der Hauptsuche ab und erhalten nur Bands, die nicht aus England kommen.

Die Interaktion gestaltet sich folgendermaßen: Die Hilfssuche wird bei der Typbedingung eine zusammengestellt und kann im Anschluss von der Hauptsuche oben mit dem Menüpunkt „Referenz“ verbunden werden. Hier kann dann ausgewählt werden welcher Art die Referenz sein soll (in diesem Fall negativ).

1.3.1.2.5 Entspricht / Referenz

Die Referenz ermöglicht sich innerhalb einer Strukturabfrage auf andere Bedingungen der selben Abfrage zu beziehen:

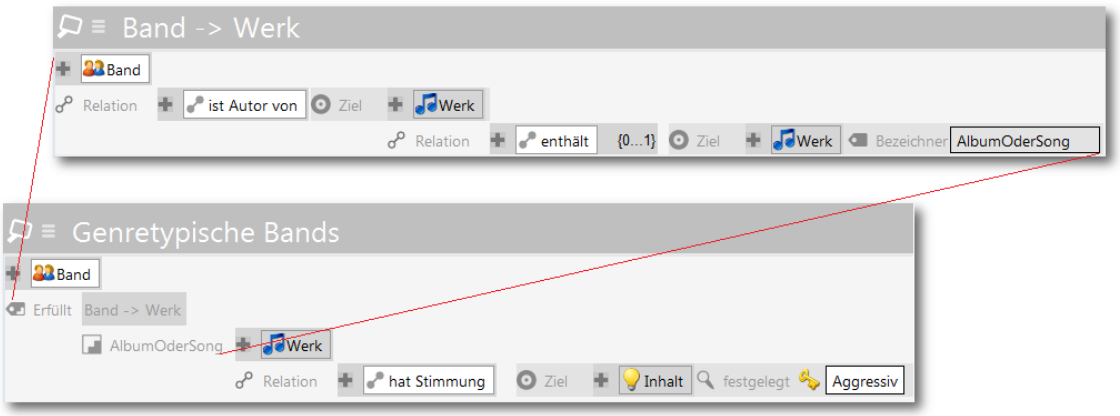


Hier referenziert die letzte Bedingung (Nr. 7) die allererste, d.h. die Band, die die Coverversion schreibt, muss auch Autor des Originals sein. Ohne Referenz würde sich die Suche folgendermaßen lesen: Bands, die Songs geschrieben haben, die andere Songs covern, die von (beliebigen) Bands geschrieben wurden. Übrigens, die Auflösung ist u.a. die Band „Radiohead" (haben ihren eigenen Song „Like Spinning Plates" gecovert).

1.3.1.2.6 Weitere Optionen im Aufbau der Strukturabfragen

Suchbausteine: Andere Strukturabfragen, aber auch andere Suchen lassen sich als Baustein in Strukturabfragen einbinden. Dadurch besteht die Möglichkeit sich wiederholende Teilabfragen in eigene Bausteine auszulagern und so z.B. bei Modelländerung das Verhalten an einer zentralen Stelle anzupassen. Ein Baustein kann bei jeder Bedingungszeile eingebaut werden.

Beispiel aus unserem Musiknetz: Von Bands zu allen ihren Werken kommen, egal ob es Alben sind, Songs, die der Band direkt zugeordnet sind, oder Songs auf den Alben der Band. Diese Teilabfrage brauchen wir häufiger, u.a. in einer Strukturabfrage, die Bands zu einem bestimmten *Mood* zurückliefert. Wir beginnen diese Abfrage mit einer Typbedingungen - wir suchen Bands - und binden als Bedingung für diese Bands den vorher definierten Baustein ein:



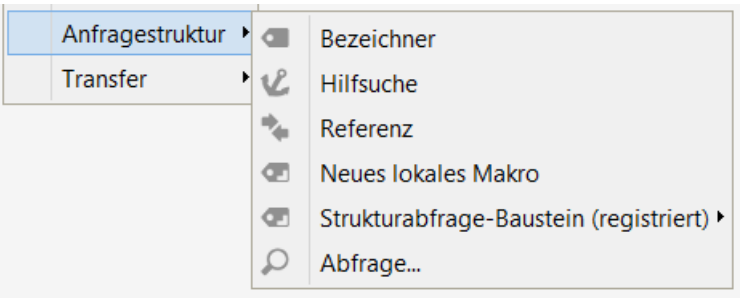
Die Objekte, die die als Baustein eingebundene Strukturabfrage zurückliefert, müssen selb-

stverständlich vom Typ her zu der Bedingung, bei der sie eingebunden ist, passen. Mit Hilfe der **Bezeichner**-Funktion kann die Abfrage (von der "aufrufenden" Abfrage aus) noch mit zusätzlichen Bedingungen weitergeführt werden. In unserem Fall werden die Alben und Songs, von denen die Baustein-Abfrage ausgeht, von der aufrufenden Abfrage bestimmt: nämlich Alben und Songs mit dem Mood "Agressive". Das einbinden des Suchbausteins in eine Strukturabfrage wird über das Menü "Anfragestruktur" vorgenommen. Unter *Strukturabfrage-Baustein (registriert)* liegt eine Auswahlliste mit allen registrieren Bausteinen.

Einfache Suche: Mit der Suchbedingung „einfache Suche“ kann das Ergebnis einer einfachen Suche oder einer Such-Pipeline als Input für eine Strukturabfrage dienen. Mit dem Auswahlsymbol kann die jeweilige einfache Suche ausgewählt werden. Das Eingabefeld beinhaltet die Sucheingabe an die einfache Suche. Durch weitere Bedingungen können z.B. einfache Suchen weiter gefiltert werden.

Kardinalitätsbedingung: Eine Suche nach Attributen bzw. Relationen ohne eigene Bedingungen kann mit einem Kardinalitäts-Operator (durch # gekennzeichnet) versehen werden. Möglich sind Kardinalität größer-gleich, kleiner-gleich und gleich. Der normale gleich Operator der Relations- oder Attributsbedingung entspricht der Kardinalität größer gleich 1.

Damit haben wir alles abgedeckt, was wir im Menu "Anfragestruktur" finden:

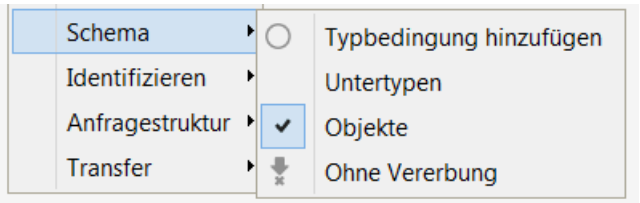


1.3.1.3 Die Bedingungen im Detail

Die Typbedingung

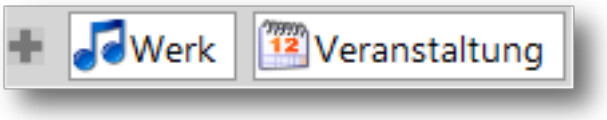
Am Anfang wird in der Strukturabfrage definiert, welches Objekte als Ergebnisse erscheinen soll. Dazu wird das Typ-Icon der ersten Bedingung angeklickt und im Menü „Typ wählen“ die Eingabemaske gestartet, in die der Objektname eingeben werden kann.
Alternativ kann hinter dem Typ-Icon der Text mit dem Objektnamen überschrieben werden.

Im zweiten Schritt wird die Relationsbedingung hinzugefügt. Bspw. wird nach dem Herkunftsort einer Band gesucht und „hat Ort“ als Relationsbedingung gesetzt. Es wird automatisch der Zieltyp der Relation eingefügt, der aber ebenfalls geändert werden kann (wenn z.B. die „hat Ort“ Relation für Länder, Städte, Regionen gilt, wir aber in unserer Abfrage nur Städte zurückbekommen möchten).

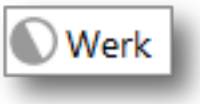


Für eine Typbedingung stehen noch weitere Funktionen zur Verfügung. Unter dem Punkt dazu gibt es den Punkt „Schema“ im allgemeinen Bedingungsmenu, das wir über die Schaltfläche + erreichen: mehrere Typbedingungen hintereinander definiert werden, was in der Abfrage als

„oder“ interpretiert wird. Folgendermaßen suchen wir z.B. Werke oder Veranstaltungen zu einem bestimmten Musikstil:



Wir können statt konkreter Objekte nur Objekttypen suchen oder beides gleichzeitig, indem im Menü „Schema“ der Haken zusätzlich bei „Untertypen“ gesetzt wird.

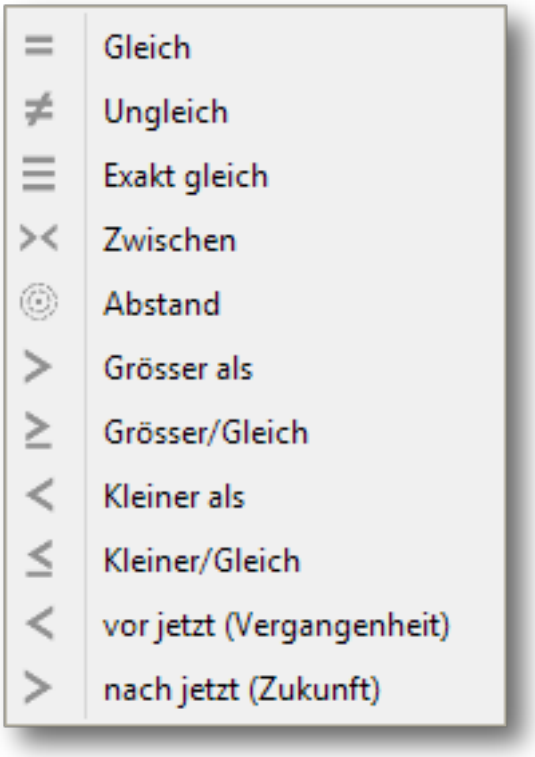


So sieht die Bedingung aus, wenn sowohl konkrete Werke als auch Untertypen von Werk (Alben, Songs) gesucht werden.

Ohne Vererbung: Normalerweise wirkt bei allen Typbedingungen der Strukturabfragen automatisch die Vererbung. Wird nach Veranstaltungen gesucht, bei denen Bands einer bestimmten Musikrichtung spielen, dann werden automatisch alle Untertypen von Veranstaltung mit in die Suche einbezogen und entsprechend Hallenkonzerte, Clubkonzerte, Festivals etc. mit zurückgeliefert. In der überwiegenden Mehrzahl der Fälle ist das genauso gewünscht. Für die Ausnahmen gibt es die Möglichkeit, die Vererbung auszuschalten und die Suche auf direkte Objekte vom Typ Veranstaltung einzuschränken, d.h. die Objekte der Untertypen auszuschließen.

Operatoren zum Vergleich von Attributwerten

Attribute können ebenfalls eine als Bedingungen für Strukturabfragen spielen. Bspw. wenn es nicht reicht Objekte ermitteln zu können, die genau einen vorgegebenen oder als Parameter eingegebenen Wert aufweisen. Etwa sollen Bands, die in der Zeit nach dem Jahre 2005 gegründet wurden oder Songs, die mehr oder weniger 3 Minuten lang sind oder Songs, die im Titel das Wort „Planet“ enthalten. Hierzu werden Vergleichsoperatoren benötigt. Welche Vergleichsoperationen uns i-views anbietet, ist abhängig vom technischen Datentyp des Attributs:

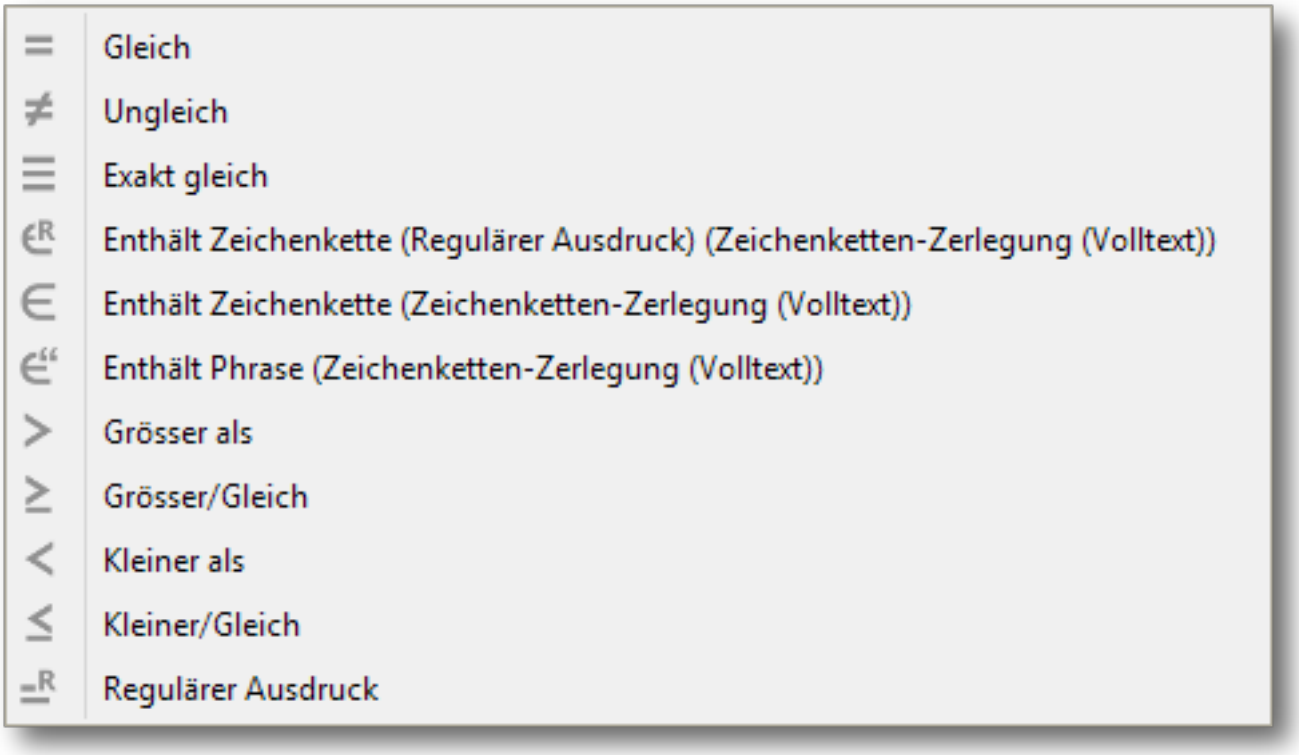


Vergleichsoperationen für Datumsangaben und Quantitäten

Der Vergleichsoperator *Exakt Gleich* bildet einen Sonderfall: Der Indexfilter wird abgeschaltet und es lässt sich nach dem Sonderzeichen * suchen, welches normalerweise als Wildcard benutzt wird.

Der Vergleichsoperator *Zwischen* erfordert eine Schreibung des Parameterwerts mit Bindestrich, also z.B. "10.1.2005 - 20.1.2005".

Der Vergleichsoperator *Abstand* erfordert eine Schreibung des Parameterwerts mit Tilde, also z.B. "15.1.2005 ~ 5" - d.h. am 15.1.2005 plus/minus 5 Tage.



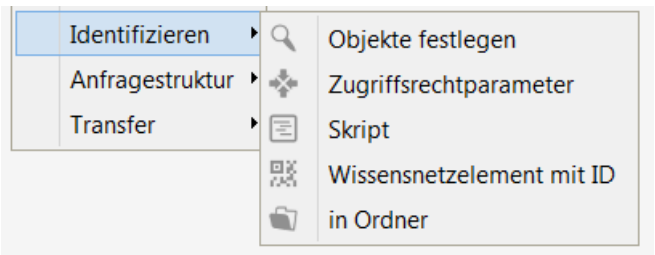
Vergleichsoperationen für Zeichenketten

Vergleichswert ergibt sich aus Skript: Attributwertbedingungen lassen sich in Teilsuchen entfernen und durch eine Skript- Attributwertbedingung ersetzen. Die Ergebnisse des Skripts werden dann als Vergleichswert für die Attributwertbedingung benutzt, bspw. falls die Vergleichsoperationen für eine spezifische Abfrage nicht ausreichen.

Objekte identifizieren

Die Strukturabfrage sieht mehrere Möglichkeiten vor, Objekte in der semantischen Graphdatenbank zu identifizieren. In den bisherigen Beispielen haben der Einfachheit halber oft Objekte festgelegt. Eine solche manuelle Festlegung kann in der Praxis hilfreich sein um Strukturabfragen zu testen oder um für eine Parametereingabe einen (austauschbaren) Default festzulegen.

Hier haben wir bereits die Kombination mit dem Namensattribut kennen gelernt, das kann natürlich ein beliebiges Attribut sein. Im Menüpunkt "Identifizieren" finden wir noch einige andere Möglichkeiten, Startpunkte für die Strukturabfrage zu bestimmen:

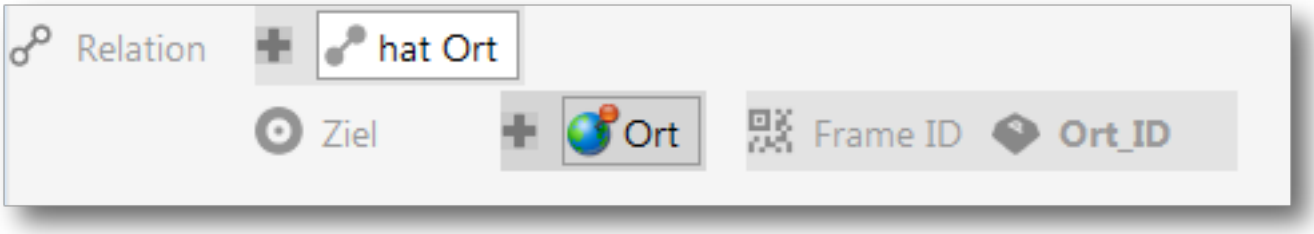


Zugriffsrechtparameter: Das Ergebnis der Abfrage vom kann vom Anwendungskontext ab-

hängig gemacht werden. Vor allem in Verbindung mit der Konfiguration von Rechten und Triggern trifft dies zu, wenn im allgemeinen nur „Benutzer“ verwendbar ist.

Skript: Die an dieser Stelle einzugebenden Objekte werden mit dem Ergebnis eines Skriptes bestimmt.

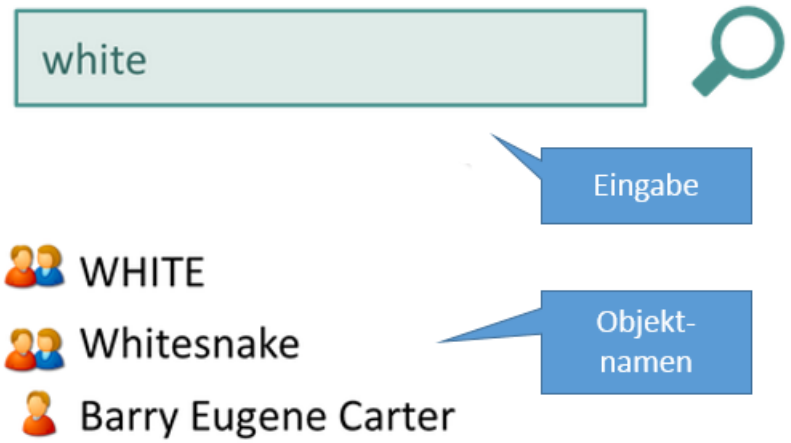
Wissensnetzelement mit ID: Man kann ein Objekt auch über seine interne ID festlegen. Diese Bedingung wird in der Regel nur in Verbindung mit Parametern und der Benutzung über die REST-Schnittstelle benutzt.



Ordner: Mit der Suchbedingung „liegt in Ordner“ kann der Inhalt einer Sammlung semantischer Objekte als Input in eine Strukturabfrage rein gegeben werden. Mit dem Auswahlsymbol kann ein Ordner innerhalb der Arbeitsordnerhierarchie ausgewählt werden. Die Objekte der Sammlung werden bezüglich aller weiteren Bedingungen (inkl. Begriffsbedingungen) gefiltert.

1.3.2 Einfache Suche / Volltextsuche

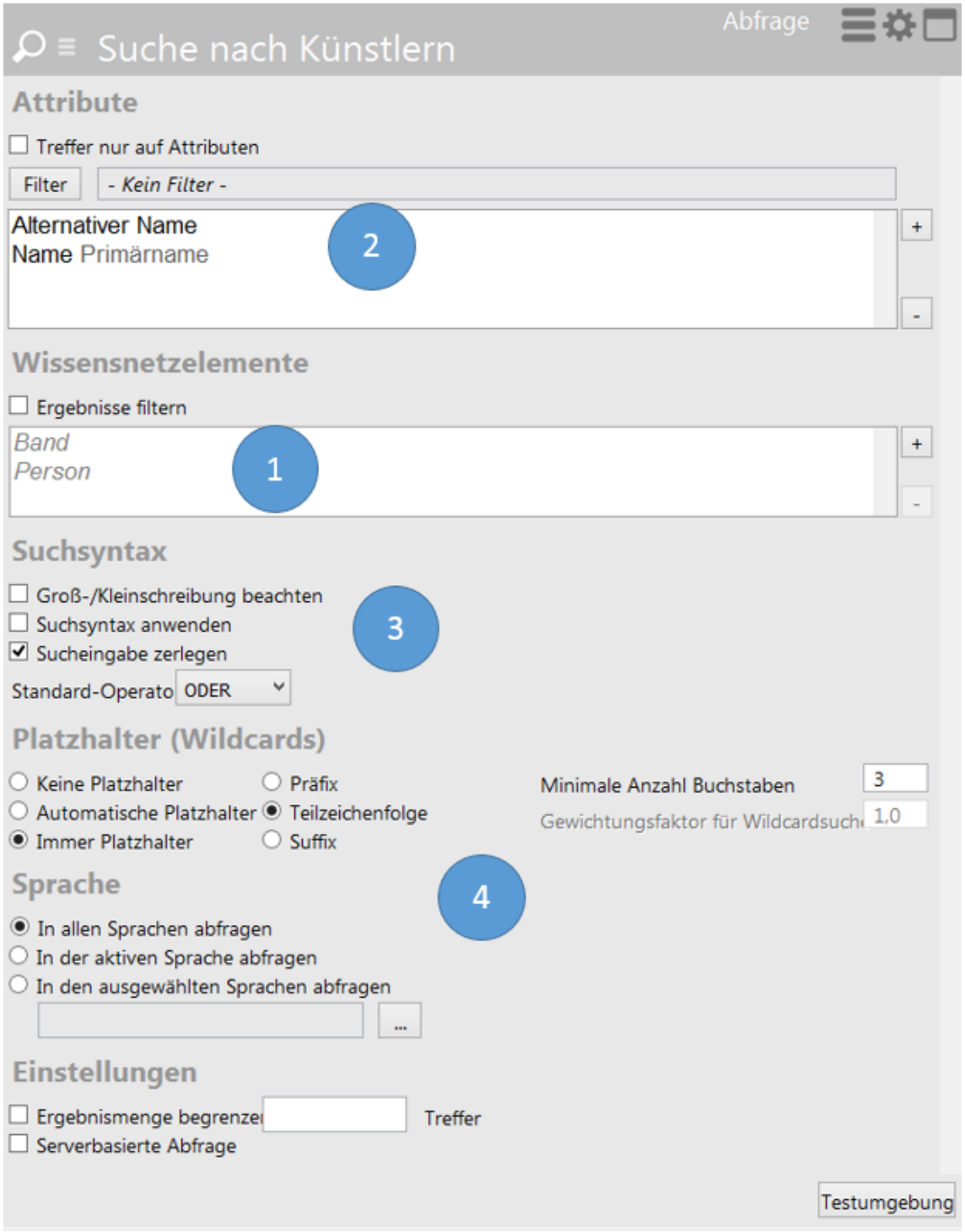
Die Verarbeitung der Suchanfragen von Nutzern kann ohne oder mit Interaktion (z.B. mit Type-Ahead-Vorschlägen) vorgenommen werden. Ausgangspunkt ist in jedem Fall die eingegebene Zeichenkette. In der Konfiguration der einfachen Suchen können wir nun festlegen, bei welchen Objekten wir in welchen Attributen nach der Nutzereingabe suchen und wie weit wir dabei von der eingegebenen Zeichenkette abweichen. Dazu ein Beispiel:



Wie müssen wir die Suche gestalten, um auf die Eingabe „white“ die Objekte unten zurückzuliefern? In allen Fällen müssen wir bei der Abfrage konfiguriert haben, dass wir Personen und Bands als Ergebnisse haben wollen. Wie steht es aber mit den Abweichungen von der Nutzereingabe?

- Wann ist die (völlig unbekannte) chinesische Experimentalband namens "WHITE" ein Treffer? Wenn wir angeben, dass Klein-/Großschreibung egal ist
- Wann bekommen wir „Whitesnake“ als Treffer geliefert? Wenn wir die Eingabe als Teilstring verstehen und eine Wildcard anhängen
- Wann „Barry Eugene Carter“? Wenn wir nicht nur den Objektnamen durchsuchen, sondern andere Attribute mit einbeziehen - sein Künstlername ist nämlich „Barry White“

Diese Optionen finden sich folgendermaßen in der Konfiguration der Suche wieder:



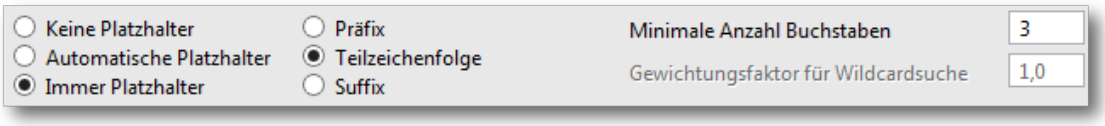
Konfiguration der einfachen Suchen mit (1) Angabe, welche Objekttypen durchsucht werden, (2) in welchen Attributen gesucht wird, (3) Groß- und Kleinschreibung und (4) Platzhaltern

1.3.2.1 Einfache Suche - Einstellungen im Detail

Platzhalter/Wildcard

Oft ist die Eingabe unvollständig oder wir wollen die Eingabe in längeren Attributfeldern

wiederfinden. Dazu können wir in der einfachen Suche Platzhalter benutzten. Folgende Einstellungen für Platzhalter finden sich in den einfachen Suchen:



- **Platzhalter hinten** (Präfix) findet bei der Eingabe "White" die [White Lies]
- **Platzhalter vorne** (Suffix) findet [Jack White]
- **Platzhalter hinten und vorne** (Teilzeichenfolge) findet [The White Stripes]
- Achtung! Platzhalter vorne ist langsam.

Die Option **Immer Platzhalter** funktioniert so als hätten wir tatsächlich einen Stern vorne oder/und hinten angehängt. Hinter **Automatische Platzhalter** steckt eine Eskalationsstrategie: Bei automatischen Platzhaltern wird erst mit der exakten Benutzereingabe gesucht. Falls diese keine Ergebnisse liefert, so wird mit Platzhalter gesucht, je nachdem welche Platzhalter eingestellt sind. Bei der Option **Präfix oder Teilzeichenfolge** gibt es noch einmal eine Reihenfolge: hier wird zuerst nach einem Präfix gesucht (durch Anhängen einer Wildcard) und, falls immer noch nichts gefunden wurde, nach einer Teilzeichenfolge (durch Voranstellen und Anhängen einer Wildcard).

Falls der Suche das Hinzufügen von Platzhaltern erlaubt ist, kann über das Feld **Minimale Anzahl Buchstaben** angegeben werden, wie viele Buchstaben die Sucheingabe mindestens aufweisen muss, damit die Platzhalter tatsächlich angefügt werden. Bei Eingabe von 0 ist diese Bedingung deaktiviert. Das ist vor allem wichtig, wenn wir eine Type-Ahead-Suche bauen.

Mit dem Gewichtungsfaktor für Wildcards läßt sich die Trefferqualität dahingehend anpassen, dass die Anwendung von Platzhaltern niedrigere Qualitäten ergibt. So können wir, wenn wir den Treffern ein Ranking mitgeben wollen, die Unsicherheit, die in den Platzhaltern steckt mit einem niedrigeren Ranking ausdrücken.

Falls die Option „keine Platzhalter“ gewählt ist, so wird die Sucheingabe nicht verändert. Die einzelnen Platzhalter-Einstellungen stehen dann nicht zur Verfügung.

Der Benutzer kann natürlich selbst Platzhalter in der Sucheingabe verwenden, die dann bei der Suche berücksichtigt werden.

Suchsyntax anwenden: Wenn kein Haken bei der Option „Suchsyntax anwenden“ gesetzt ist, so wird eine einfachere Form der Analyse der Sucheingabe verwendet, in der die steuernden Zeichenketten „|“ (Oder-Bedingung) , „&“ (Und-Bedingung) und „!“ (Negierung) keine Wirkung mehr haben. Um dennoch festlegen zu können, wie die Treffer für die einzelnen Tokens zusammengefasst werden sollen, lässt sich noch der Default-Operator auf „#and“ oder „#or“ schalten. Für alle Verknüpfungsoperatoren gilt, dass sie sich nicht auf Werte eines einzelnen Attributes beziehen, sondern auf die Ergebnisobjekte (je nach dem, ob „Treffer nur auf Attributen“ gesetzt ist). Ein Treffer für Online System liefert somit semantische Objekte, die sowohl für Online als auch für System ein passendes Attribut besitzen (welches nicht notwendigerweise dasselbe ist).

Filterung: Einfache Suchen, Volltextsuchen und auch einige der Spezialsuchen können nach Objekttypen gefiltert werden. Im Beispiel des letzten Abschnitts haben wir so dafür gesorgt, dass nur Personen und Bands als Suchergebnis zurückgeliefert werden. Attribute, die nicht zu einer eventuellen Filterung passen, werden im Suchkonfigurationsdialog fett und rot dargestellt. Das könnte in unserem Fall beispielsweise ein Attribut „Rezension“ sein, dass

nur für Alben definiert ist.

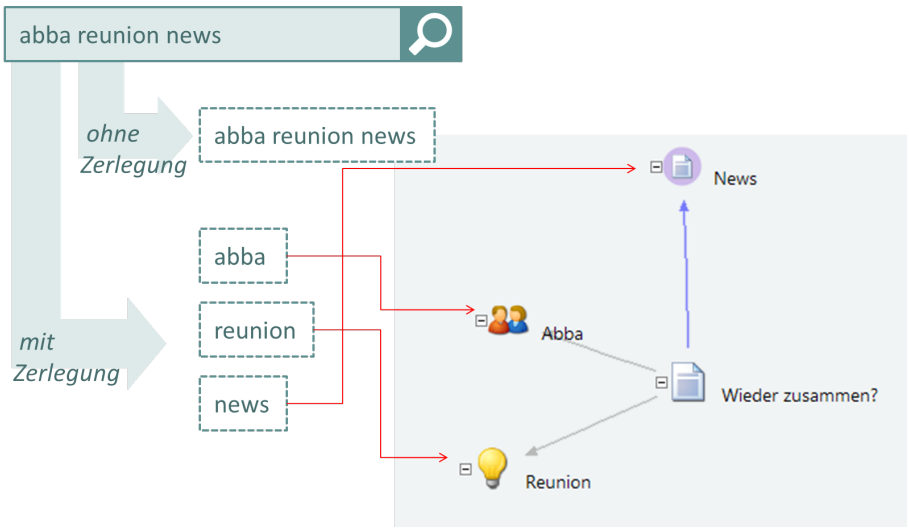
Übersetzte Attribute: Bei übersetzten Attributen können wir entweder eine Übersetzung wählen, oder die Sprache dynamisch bestimmen lassen. Mehrsprachige Attribute suchen dann in der aktiven Sprache oder in allen Sprachen, je nachdem, ob die Option „in allen Sprachen suchen“ gesetzt ist.

Ergebnismenge: Eine maximale Ergebnismenge kann durch die Eingabe der maximalen Anzahl im Feld „Ergebnisse“ bestimmt werden. Durch die Checkbox Ergebnismenge begrenzen kann dieser Mechanismus aktiviert oder deaktiviert werden. Durch Eingabe einer Zahl in Ergebnisse wird die Checkbox automatisch aktiviert. Achtung: Wird die Anzahl überschritten, werden keine Ergebnisse angezeigt!

Serverbasierte Suche: Es kann generell jede Suche auch als serverbasierte Suche ausgeführt werden. Voraussetzung ist, dass ein dazugehöriger Jobclient läuft. Diese Option kann eingesetzt werden, wenn abzusehen ist, dass sehr viele User Suchanfragen stellen werden. Durch die Auslagerung bestimmter Suchen auf externe Server wird der i-views-Server entlastet.

1.3.2.2 Mehrwort-Eingaben

Bei unseren Beispielen zu Abfragen haben die Nutzer bisher immer nur einen Suchbegriff eingegeben. Was aber, wenn der Nutzer z.B. "Abba Reunion News" eingibt und damit vielleicht einen News-Artikel finden möchte, der mit den Themen "Abba" und "Reunion" verschlagwortet ist? Diese Eingabe müssen wir zerlegen, auf den gesamten String wird keines unserer Objekte matchen oder zumindest nicht der gesuchte Artikel:



Unsere bisherigen Beispiele greifen aber nicht nur wegen Mehrwort-Sucheingaben zu kurz. Wir haben auch oft Suchsituationen, in denen es nicht sinnvoll ist, die Namen oder anderen Zeichenketten aus dem Netz, gegen die wir die Eingabe vergleichen, als Blöcke zu betrachten, z.B. weil wir eine Eingabe in einem längeren Text wiederfinden möchten. Hier sind die Wildcards irgendwann kein adäquates Mittel mehr: Wenn wir auch auf der Seite der Objekte und der durchsuchten Textattribute zerlegen möchten, dann wird besser die Volltextsuche verwendet.

1.3.2.3 Volltextsuche und Indexierung

Wenn wir längere Texte, z.B. Beschreibungsattribute, wortweise betrachten bzw. durchsuchen wollen, empfiehlt sich ein Volltextindex. Wie sieht so etwas aus?

Term	Vorkommen
aaliyah	Dok#155, Pos. 548644 / Dok#459, Pos. 934875/ Dok#935, Pos. 26526
abba	Dok#132, Pos. 43095 / Dok#459, Pos. 46795 / Dok#935, Pos. 534955 / Dok#343, Pos. 367773 / Dok#711, Pos. 92634
abitur	Dok#464, Pos. 94674 / Dok#4356, Pos. 298456 / Dok#437, Pos. 90743
abiturient	Dok#436, Pos. 54356
abnorm	Dok#103, Pos. 8234465
abonnement	Dok#462, Pos. 230756
abonnieren	Dok#356, Pos. 298456 / Dok#233, Pos. 26725

Der Volltextindex verzeichnet alle in einem Bestand an Texten vorkommenden Terme/Worte, so dass i-views schnell und einfach nachschlagen kann in welchen Texten (und an welcher Stelle im Text) ein bestimmtes Wort vorkommt.

„Texte" sind dabei in i-views i.d.R. nicht separate Dokumente, sondern die Zeichenketten-Attribute, die durchsucht werden sollen. Ihre Volltextindexierung ist Voraussetzung dafür, dass diese Attribute in der Suchkonfiguration angeboten werden.

Auch bei der Volltextindexierung geht es um die Abweichungen zwischen der genauen Zeichenfolge im Text und dem, was in den Index eingetragen und dementsprechend wiedergefunden wird. Beispiel: eine Nachricht aus der deutschen Musikszene:

In diesem Beispiel finden wir einen kleinen Teil der Filter- und Wortabgrenzungs-Operationen wieder, die typischerweise beim Aufbau eines Volltextindex zum Einsatz kommen:

Wortabgrenzung / Tokenizing: Im Deutschen werden Satzzeichen wie z.B. das Ausrufezeichen direkt ohne Leerzeichen an das letzte Wort des Satzes gesetzt. In den Volltextindex wollen wir aber den Eintrag {Tour}, nicht den Eintrag {Tour!} aufnehmen - nach letzterem wird wohl kaum jemand suchen. Dazu müssen wir beim Aufbau des Volltextindex angeben können, dass bestimmte Zeichen nicht zum Wort gehören. Nicht immer ist die Entscheidung so einfach: Bei einer Zeichenfolge wie „Kuschel-Klassik", die in einem Text vorkommt, müssen wir uns entscheiden ob wir diese als einen Eintrag in den Volltextindex aufnehmen wollen oder als {Kuschel} und {Klassik}. Im ersten Fall wird unsere Nachricht nur dann gefunden, wenn genau nach „Kuschel-Klassik" oder z.B. „*uschel-K*" gesucht wird, im zweiten Fall auch bei allen „Klassik"-Suchen.

Was wir trotz des Vorkommens von Satzzeichen wahrscheinlich zusammenhalten, d.h. vom Tokenizing ausnehmen wollen, sind Abkürzungen: wenn AC/DC nur a.d.D. nach Deutschland kommen (auf der Durchreise), dann ist es wahrscheinlich besser, die Abkürzung mit im Index zu haben statt der einzelnen Buchstaben.

Filtern: Durch Filteroperationen können wir sowohl Worte bei ihrer Aufnahme in den Volltextindex abwandeln als auch ihre Aufnahme komplett unterdrücken. Bekannt: **Stoppworte**, hier können wir Liste pflegen. Auch **einzelne Buchstaben** (Bela B.) wollen wir wahrscheinlich nicht so im Index stehen haben - die Verwechselungsgefahr ist zu groß. Mit anderen Filtern können wir **Worte auf ihre Grundformen** zurückführen oder **Ersetzungslisten für einzelne Zeichen** definieren (um z.B. Akzente zu eliminieren). Andere Filter wiederum bere-

inigen den Text um XML-Tags.

All das können wir im Admin-Tool unter „Indexkonfiguration“ einstellen. Diese Konfigurationen können wir dann (im KB oder im Admin-Tool) den Zeichenkettenattributen zuweisen. Die Indexkonfiguration ist so organisiert, dass eine Filterung vor der Wortabgrenzung und eine Filterung nach der Wortabgrenzung stattfinden kann.

Bei der Volltextsuche greift die Wildcard-Automatik der anderen Abfragen nicht, aber der Nutzer kann natürlich seine Eingabe mit Wildcards versehen.

1.3.3 Such-Pipeline

Such-Pipelines ermöglichen es, einzelne Komponenten zu komplexen Abfragen zu kombinieren. Die einzelnen Bausteine führen Operationen aus dabei z.B.:

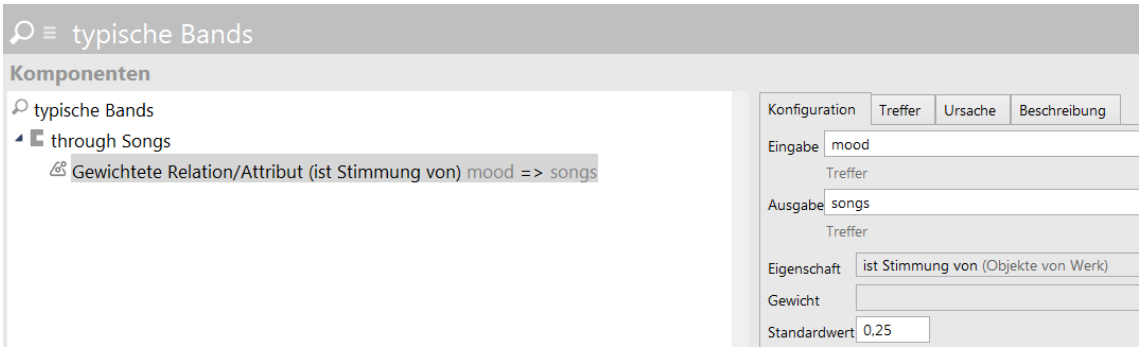
- Das Netz traversieren und dabei Gewichtung bestimmen
- Strukturabfragen und einfache Abfragen ausführen
- Treffermengen zusammenfassen

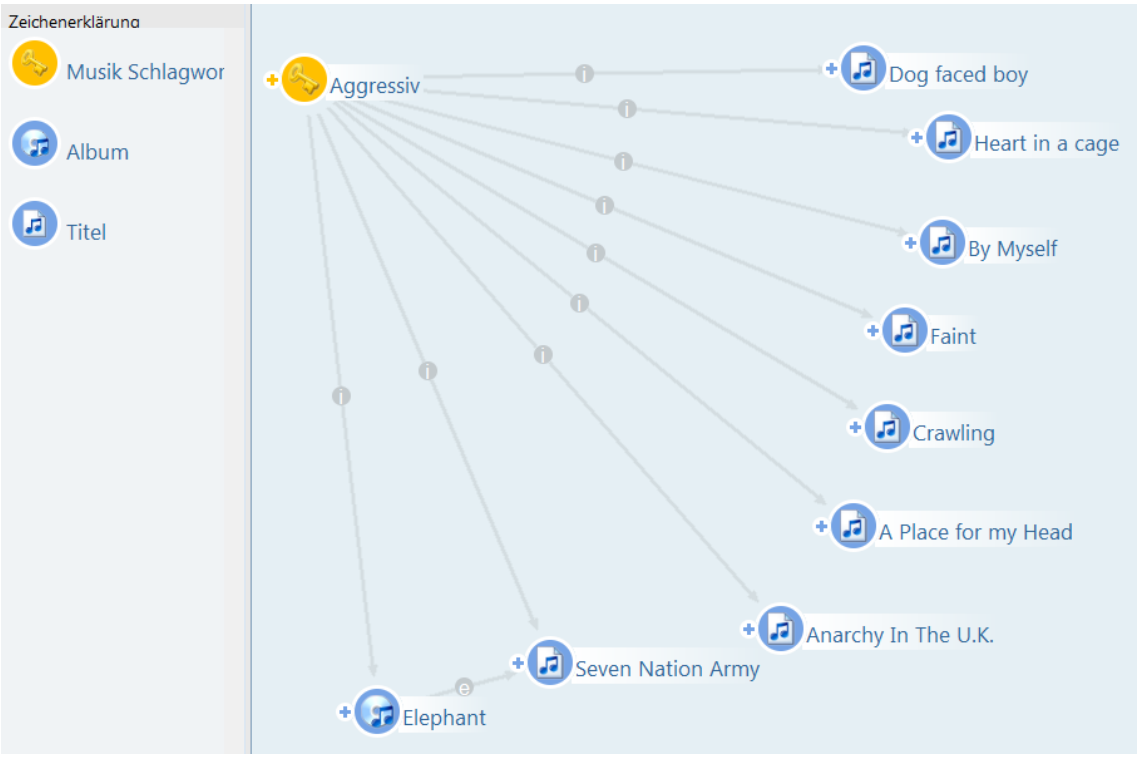
Jeder Abfrageschritt erzeugt eine Ergebnismenge (i.d.R. eine Menge von Objekten). Diese Ergebnismenge kann wiederum als Eingabe für nachfolgende Komponenten in der Pipeline verwendet werden.

Beispiel

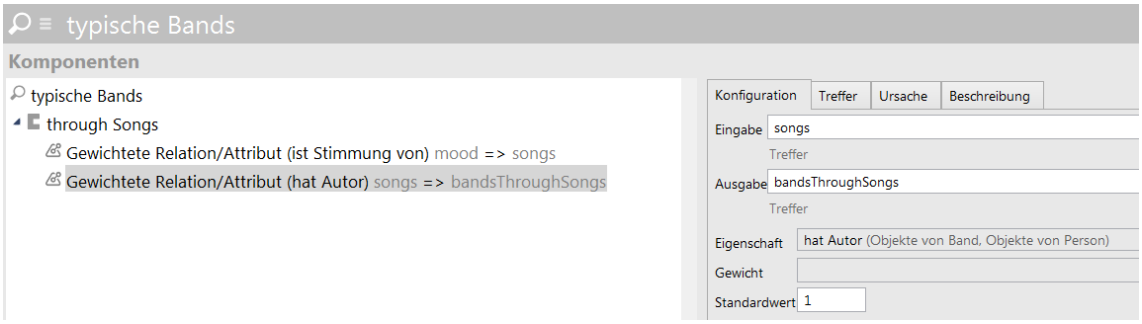
Nehmen wir an, Songs und Künstler unseres Musiknetzes sind mit Tags für ihre typische Stimmung charakterisiert. Ausgehend von einer bestimmten Stimmung wollen wir nun die Bands herausfinden, welche diese Stimmung am besten vertreten.

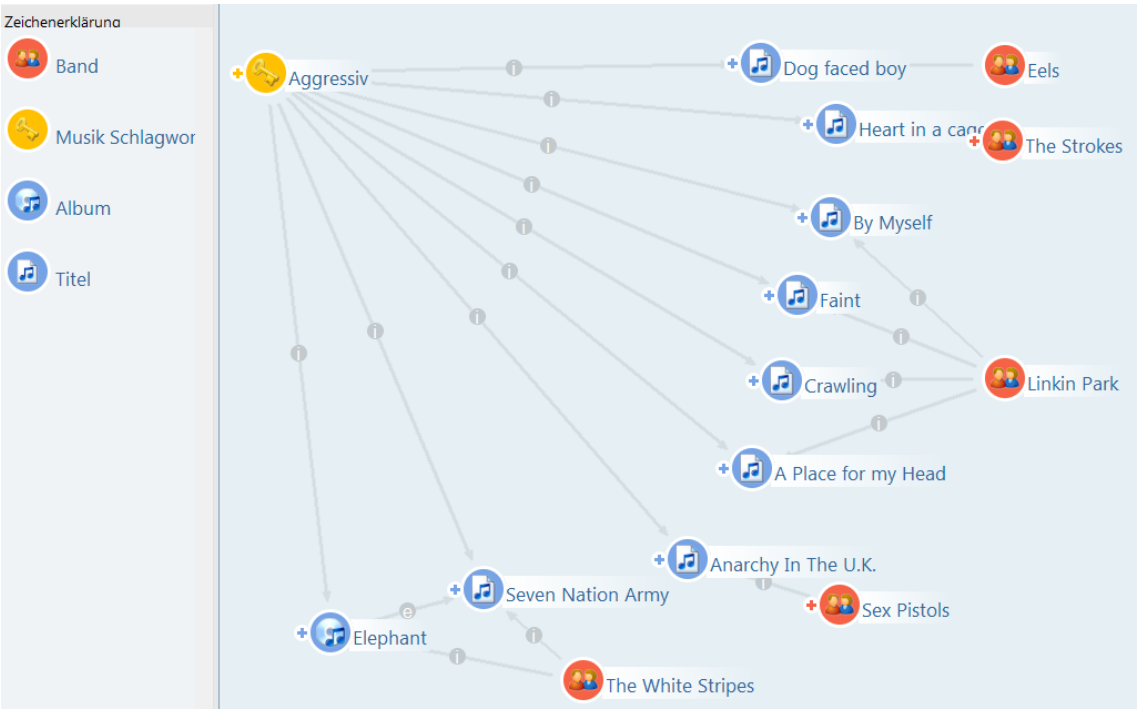
Schritt 0 unserer Such-Pipeline nimmt eine Ausgangsstimmung als Parameter entgegen und belegt damit die Variable namens "mood", in unserem Beispiel nehmen wir die Stimmung "Aggressiv" als Input und schauen uns jetzt an, wie wir mit Hilfe der Suchpipelines für diese Stimmung typische Bands ermitteln: Schritt 1 geht von der Ausgangsstimmung über die Relation *ist Stimmung von* zu den Songs, denen diese Stimmung zugeordnet ist:



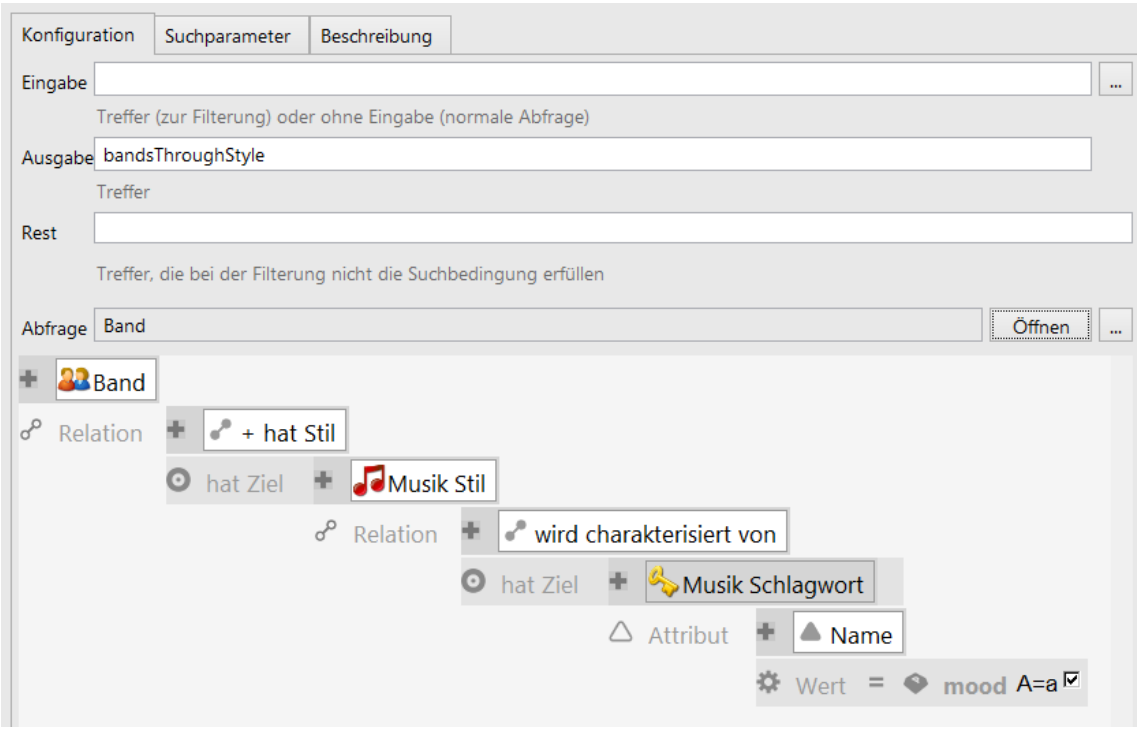


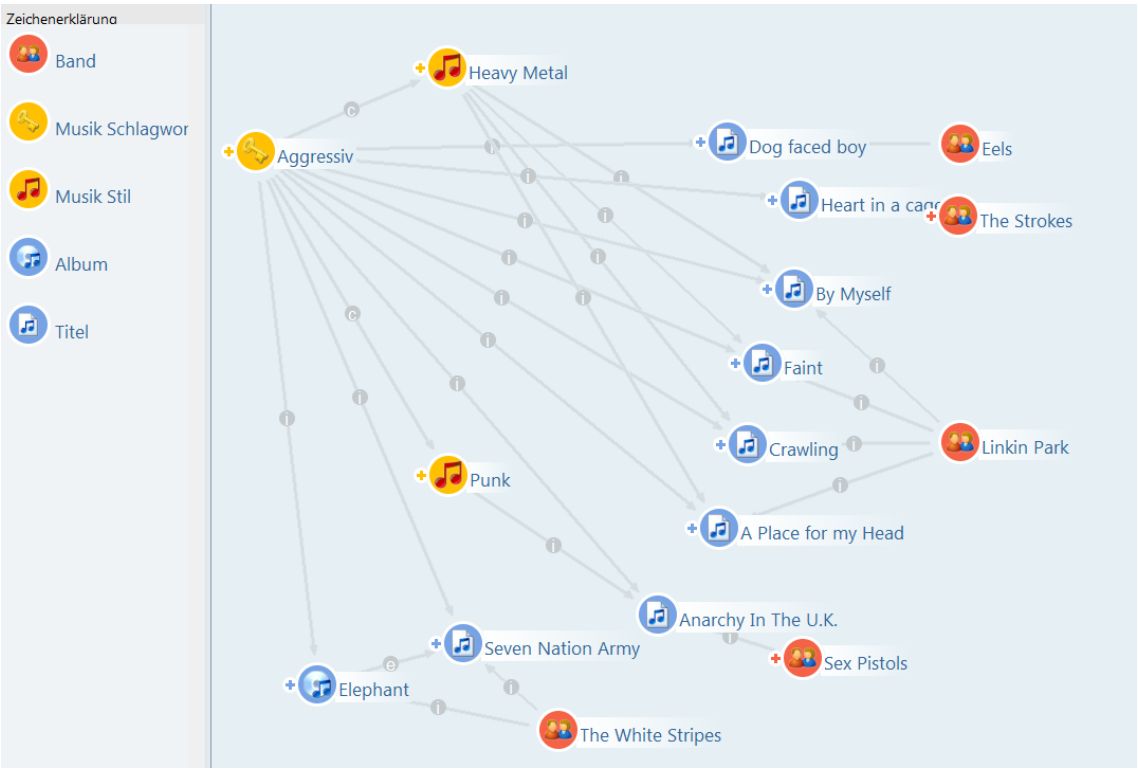
Im zweiten Schritt gehen wir von der Menge an ermittelten Songs zu den entsprechenden Bands über die Relation *hat Autor*:





Jetzt möchten wir noch einen zweiten Weg verfolgen: Vom Ausgangspunkt 'mood' „Aggressiv" zu Musikrichtungen, die durch Aggressivität charakterisiert werden. Von dieser Menge an relevanten Musikrichtungen ausgehend soll es wieder zu Bands gehen, die zu dieser Musikrichtung zählen. Wir könnten jetzt wieder wie oben Relationen aneinanderhängen, wir können dazu aber auch eine Strukturabfrage in die Suchpipeline einbauen:





Diesem zweiten Weg über die Musikrichtungen geben wir ein etwas niedrigeres Gewicht und fassen zum Schluss die Ergebnisse zusammen:

Name	Typ	Ursache	Suchtext	Qualität
Linkin Park	Band	A Place for my Head, By	·	100
The White Stripes	Band	Elephant, Seven Nation ,	·	38
Eels	Band	Dog faced boy	·	8
The Strokes	Band	Heart in a cage	·	6
Sex Pistols	Band	Anarchy In The U.K.	·	6

Die Abarbeitung der Schritte erfolgt sequenziell: Über die Eingabe und Ausgabe wird bestimmt, welcher Schritt mit welcher Treffermenge weiterarbeitet. So konnten wir z.B. bei unserem alternativen Weg wieder ganz vorne mit 'mood' beginnen.

Das Prinzip der Gewichtungen

Das Ziel war es, den Bands, die als Ergebnisse herauskommen, ein Ranking zu geben, das zeigt, wie groß ihre semantische „Nähe" zum mood *Aggressiv* ist. Das Ranking beeinflussen wir in dieser Suche vor allem an zwei Stellen: ganz zum Schluss gewichten wir in der Zusammenfassung Bands höher, die sowohl über ihre Musikrichtung als auch über Songs gefunden werden. Das trifft hier auf Linkin Park und auf Sex Pistols zu. Das höhere Ranking von Linkin Park resultiert daraus, dass immer wieder von verschiedenen Songs mit dem mood Agressiv zu Linkin Park führt. Da im Datenbestand mehr aggressive Songs von Linkin Park vorhanden sind, soll Linkin Park mit höherem Ranking 'belohnt' werden.

1.3.3.1 Zusammenstellung von Such-Pipelines

Die einzelnen Komponenten einer Such-Pipeline, werden im Hauptfenster im Feld *Komponente* in der Reihenfolge ihrer Ausführung dargestellt.

Mit der Schaltfläche *Hinzufügen* können wir eine neue Komponente am Ende der vorhandenen Komponenten anfügen.

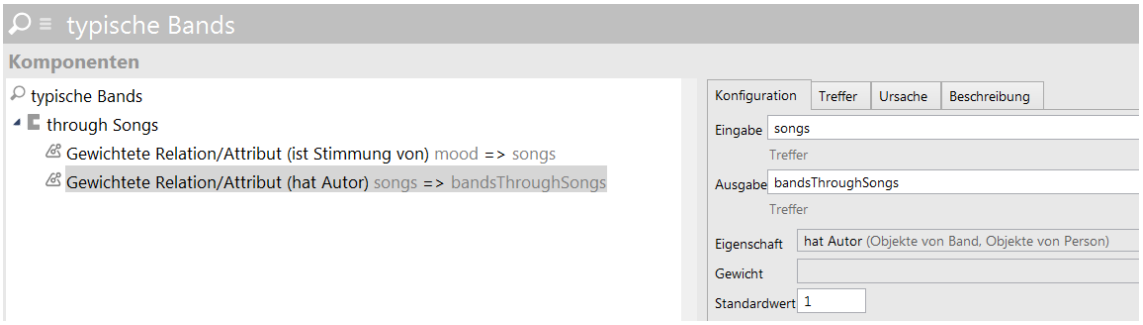
Die Gruppierung mit Blöcken dient allein der Übersichtlichkeit, etwa zur Zusammenfassung mehrerer Komponenten zu einem Funktionsbereich der Such-Pipeline.

Mit den Schaltflächen *Nach oben* und *Nach unten* oder mit Drag&Drop kann die Reihenfolge der Schritte geändert werden.

Mit *Entfernen* wird die ausgewählte Komponente entfernt, inklusive aller evtl. enthaltenen Unterkomponenten. Auf der rechten Seite im Hauptfenster wird die Konfiguration für die ausgewählte Komponente angezeigt.

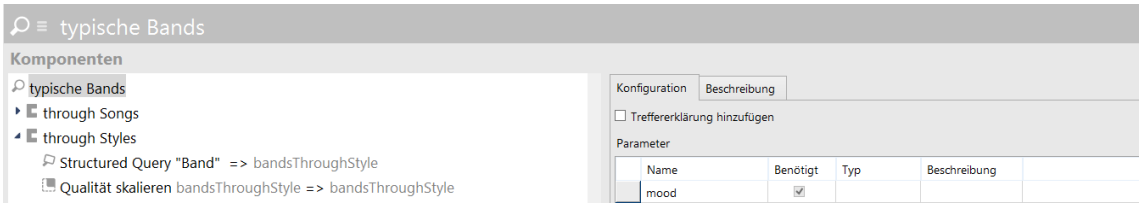
Konfiguration einer Komponente

Auf der rechten Seite des Hauptfensters lässt sich unter dem Reiter „Konfiguration“ eine gewählte Komponente konfigurieren: Die meisten Komponenten benötigen eine **Eingabe**. Diese kommt i.d.R. von einem vorangegangenen Schritt. So gibt die erste Komponente in unserem Beispiel ihr Ergebnis unter der Variable „songs“ an die nächsten Komponenten weiter, diese geht von da aus zu den Bands und gibt das Ergebnis wiederum als „bandsThroughSongs“ an die nächsten Schritte weiter:



Über die Eingabe- und Ausgabe-Variable können wir auch in späteren Schritten wieder neu bei den ersten Ergebnissen aufsetzen, wie wir schon im letzten Abschnitt gesehen haben.

Als globale Einstellungen für die Suche legen wir die Eingabeparameter fest. Unter dem Namen, den wir hier vergeben, können wir in den einzelnen Schritten unserer Such-Pipeline dann auf diese Eingaben zugreifen. In unserem Beispiel ist der Mood, zu dem wir typische Bands ermitteln, der Eingabeparameter.



Einige Komponenten erlauben eine Abweichungen von der Standard-Verarbeitungsreihenfolge:

Einzelverarbeitung: Elemente einer Menge wie z.B. Treffer einer Suche kann man auch einzeln verarbeiten. Dies ist z.B. praktisch, wenn man zu Suchtreffern eine individuelle Umgebung von benachbarten Objekten aufsammeln will. Bei der Einzelverarbeitung wird jedes Element in der bei Einzeltreffer konfigurierten Variable gespeichert und die Unterkomponenten ausgeführt.

Bedingung Parameter gesetzt: Diese Komponente führt weitere Unterkomponenten nur dann aus, wenn vorgegebene Parameter gesetzt sind. Der Wert ist dabei unerheblich. Mit

Hinzufügen kann man eine neue Unterkomponente hinzufügen.

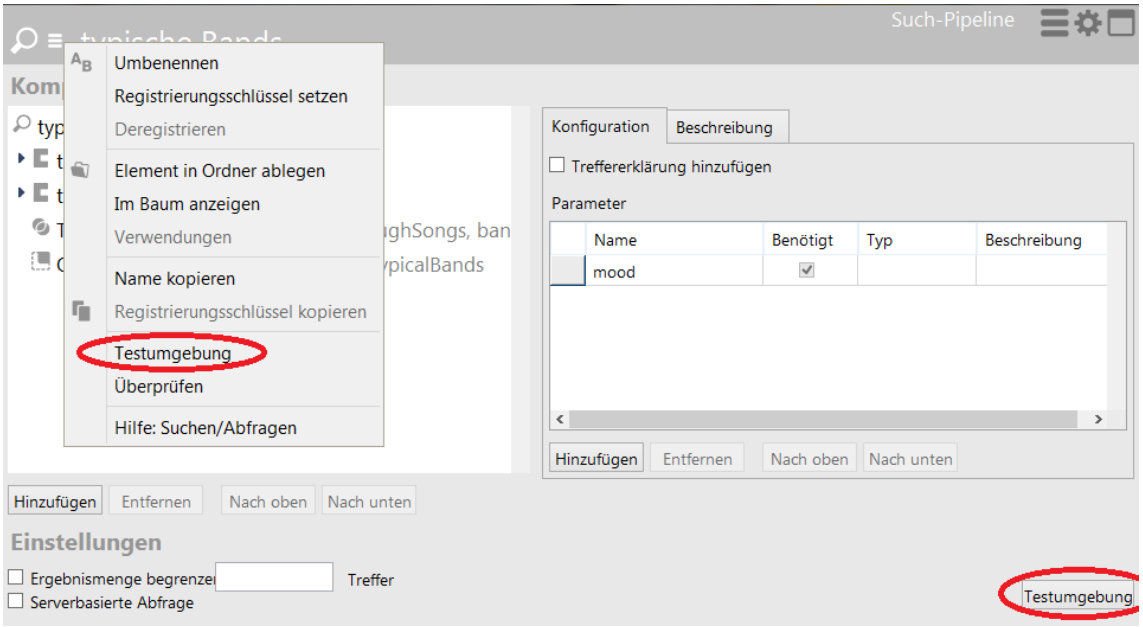
KPath-Bedingung: Mit einer KPath-Bedingung können wir bestimmen, dass die Unterkomponenten nur dann ausgeführt werden, wenn eine in KPath formulierte Bedingung erfüllt ist. Falls die Bedingung nicht erfüllt ist, wird die Eingabe übernommen. KPath ist im Handbuch zu KScript beschrieben.

Ergebnis: An beliebigen Stellen der Suche können wir die Suche abbrechen und ein Ergebnis zurückgeben. Diese Komponente ist unter anderem für das Testen der Such-Pipeline nützlich.

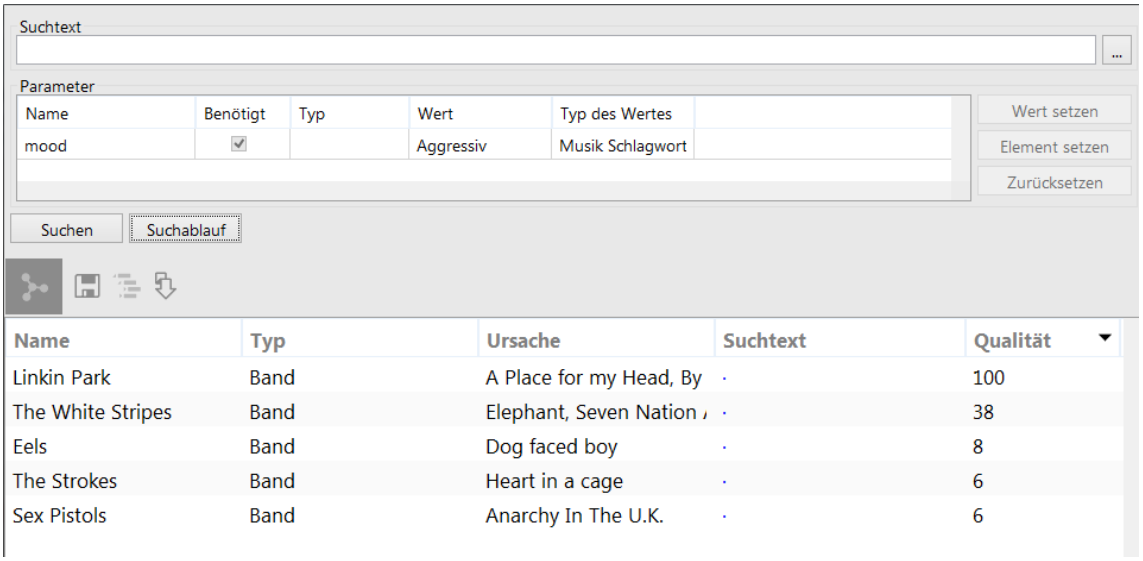
Die **Block-Komponente**, die wir auch in unserem Beispiel verwendet haben, gruppiert eine Menge von Einzelschritten. Um bei umfangreicheren Konfigurationen den Überblick zu wahren, können wir zudem auf dem Reiter „Beschreibung" den Namen der Komponente ändern sowie einen Kommentar hinzufügen. Weder die Block-Komponente noch die Beschreibung haben funktionale Auswirkungen. Beides dient lediglich der Lesbarkeit der Such-Pipeline.

Testumgebung

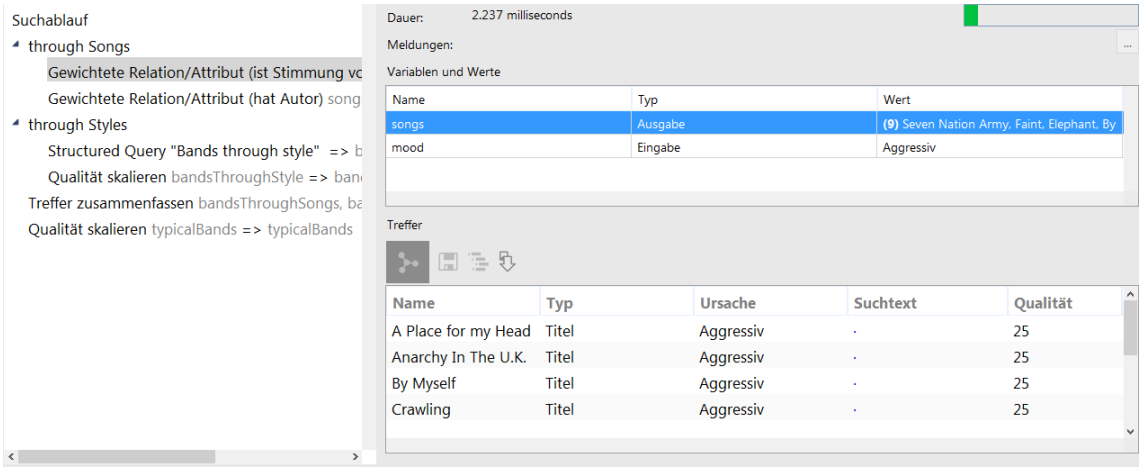
Auf die Testumgebung kann an folgenden Stellen zugegriffen werden:



Mit der Testumgebung können wir die Arbeitsweise der Suche analysieren. Der obere Teil enthält die Sucheingabe, der untere Teil das Ergebnis. Die Sucheingabe kann ein Suchtext oder ein Element der semantischen Graph-Datenbank sein, je nachdem welche geforderten und optionalen Eingabeparameter wir global in der Such-Pipeline definiert haben. Wenn wir ein Element der semantischen Graph-Datenbank als Startpunkt eingeben wollen, selektieren wir die entsprechende Parameter-Zeile und fügen, je nach Typ einen Attributwert oder ein Element der semantischen Graph-Datenbank hinzu.



Auf dem Reiter *Suchablauf* wird ein Protokoll der Suche angezeigt. Dieses besteht im wesentlichen aus der Belegung der Ausgabevariablen sowie der Dauer zur Ausführung der einzelnen Komponenten.



Verrechnungsmöglichkeiten

Bei einigen Komponenten ist es möglich, mehrere Qualitätswerte zu einem einzigen Qualitätswert zusammenzufassen - z.B. bei „Treffer zusammenfassen“, aber auch beim Traversieren von Relationen (siehe Beispiel oben). Dabei stehen folgende Berechnungsmethoden zur Verfügung:

- Addieren / Multiplizieren
- Arithmetischer Mittelwert / Median
- Minimum / Maximum
- Ranking

Die Option „Ranking“ passt immer dann, wenn wir aus Einzelhinweisen ein Gesamtbild zusammenfügen möchten, z.B. wenn wir viele, zumindest teilweise unabhängige Wege - am Ende noch mit unterschiedlicher Länge - zu einer „Gesamtnähe“ verrechnen wollen. Mit der Ranking-Verrechnung sorgen wir dafür, dass alle positiven Hinweise (alle unabhängigen

Wege) die Ähnlichkeit immer weiter erhöhen, ohne dass die 100% überschritten werden.

In der Such-Pipeline werden Qualitätswerte immer als Fließkommazahlen angegeben. Der Wert 1 entspricht dabei einer Qualität von 100%.

1.3.3.2 Die einzelnen Komponenten

Gewichtete Relationen und Attribute

Ausgehend von semantischen Objekten können wir mit diesem Schritt den Graph traversieren und Relationsziele oder Attribute aufsammeln. Dazu müssen wir den Typ der Relation oder des Attributs angeben.

Achtung: Ausgabe sind nur noch die aufgesammelten Ziele, nicht mehr die Ausgangsmenge. Wenn diese angezeigt werden sollen, müssen wir anschließend die Eingangs- und die Ausgangstreffer zusammenfassen.

Bei der Traversierung einer Relation kann die Gewichtung der Treffer beeinflusst werden. Nehmen wir an, wir wollen den „Ausgangs-mood" unserer Beispielsuche um „Unter-moods" semantisch erweitern. Aber diese Indirektion soll sich in einem Ranking niederschlagen: Verbindungen zu Bands, die über die Untermoods laufen, sollen nicht so stark zählen wie Verbindungen über einen Ausgangsmood. Zu diesem Zweck können wir das Relation-Entlanggehen mit einem pauschalen Wert - z.B. 0,5 - belegen, und mit dem Eingangsgewicht verrechnen, beispielsweise multiplizieren. Dann zählen die in diesem Schritt hinzugefügten Untermoods nur halb so viel wie die direkten.

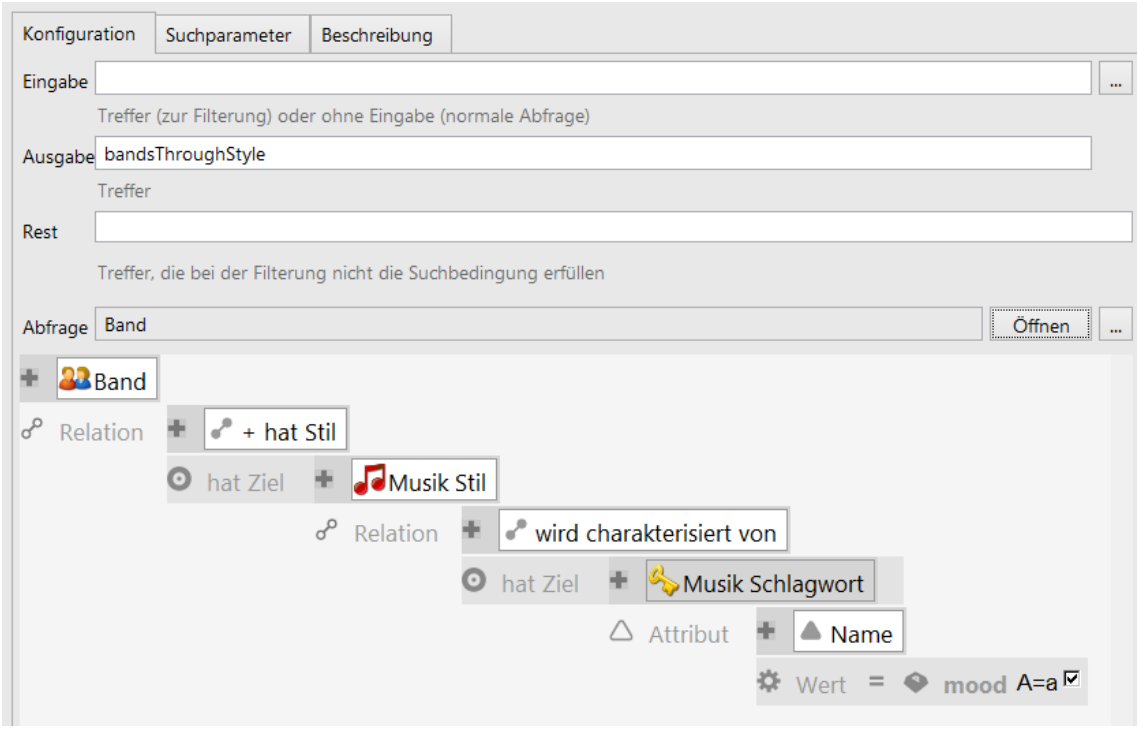
Statt eines pauschalen Gewichts für das Entlanggehen der Relation zu vergeben, können wir den Wert auch aus einer Metaeigenschaft des Basistyps Fließkommazahl der ausgewählten Relation auslesen. Falls dieses Attribut nicht vorhanden ist oder keines konfiguriert wurde, wird der Standardwert verwendet. Der Wert sollte zwischen 0 und 1 liegen. Die Treffererzeugung kann detaillierter konfiguriert werden: Bei Relationen kann optional auch für die Relationsquelle (statt für das Relationsziel) ein neuer Treffer erzeugt werden.

Wenn eine Relation als Eigenschaft ausgewählt wurde und Treffer für Relationsziele erzeugt werden, können wir optional auch die Relation transitiv verfolgen. Bei jedem Schritt verringert sich der Qualitätswert, bis der angegebene Schwellwert unterschritten wird. Falls ein Objekt mehr Relationen hat als bei maximaler Fanout angegeben, werden diese Relationen nicht verfolgt. Je höher der Dämpfungsfaktor ist, desto stärker wird der Qualitätswert verringert.

Strukturabfrage

Mit Strukturabfrage-Komponenten können wir entweder semantische Objekte suchen / von einer bestehende Menge zu anderen Objekten gehen (wie mit der gewichteten Relation) oder eine Menge filtern.

Wenn wir Objekte suchen, leiten wir unsere Ausgangsmenge von Treffern aus einem der vorhergehenden Schritte über den Parameternamen in die Suche ein. (Allgemein: Innerhalb der Expertensuche können Variablen der Such-Pipeline wie z.B. search-String über Parameter referenziert werden). Die Eingabe bleibt in diesem Fall leer.



Zur Filterung geben wir dagegen als Eingabe eine Menge von Objekten an. In der Ausgabe sind alle Objekte enthalten, auf die die Suchbedingung zutrifft. Objekte, die nicht zur Suchbedingung passen, können optional in einer weiteren Variable (Rest) gespeichert werden.

Wir können die Strukturabfrage entweder direkt in der Komponente ad hoc definieren oder eine bestehende Strukturabfrage verwenden.

Achtung: Wenn eine bestehende Suche ausgewählt wurde, wird keine Kopie angelegt, Änderungen, die wir an für Zwecke der Such-Pipeline an der Strukturabfrage vornehmen, ändern sie auch für alle anderen Verwendungen.

Abfrage

Mit der Komponente „Suche“ können einfache Suchen, Volltextsuchen und andere Such-Pipelines ausgeführt werden. Einfache Suchen und Volltextsuchen werden dabei mit einer Zeichenkette versehen, z.B. mit dem *searchString*: Das ist ein Parameter, der in allen Such-Pipelines zur Verfügung steht um die Nutzereingabe zu verarbeiten. Die Treffermenge der aufgerufenen Suche belegt die Ausgabe dieser Komponente.

Über das Einbinden von Such-Pipelines in anderen Such-Pipelines können wir häufiger auftretende Teilschritte ausfaktorisieren. Anderen Such-Pipelines können mehrere Parameter übergeben werden und ganze Treffermengen übergeben. Mit eingebundenen Such-Pipelines können wir auch mehrere Parameter austauschen, d.h. wir können in der eingebundenen Suche auf jede Teilschrittausgabe der umgebenden Suche zugreifen und umgekehrt. Wenn wir auf "ausgewählte Parameter gehen, können wir diese auch umbenennen, falls wir z.B. eine Treffermenge aus der eingebundenen Suche nutzen wollen, aber den Namen schon verwendet haben. Oder wir können, um solche Konflikte zu vermeiden, nur einen Teil der Parameter aus der eingebundenen Suche übernehmen.

Treffer zusammenfassen

Mit dieser Komponente können wir mehrere Treffermengen aus unterschiedlichen vorangegangenen Schritten zusammenfassen. Folgende Methoden zur Zusammenfassung stehen zur Verfügung:

Vereinigungsmenge: Alle Treffer, die in mindestens einer der Mengen vorkommen, werden als Ergebnis ausgegeben

Schnittmenge: Nur Treffer, die in allen Mengen vorkommen, werden als Ergebnis ausgegeben.

Bei Vereinigungsmengen und Schnittmengen werden kommt es vor, dass ein semantisches Objekt in mehreren Treffermengen vorkommt und zu einem Gesamttreffer mit neuer Trefferqualität verrechnet werden muss. Hier stehen wieder die erwähnten Verrechnungsmethoden zur Verfügung.

Differenz: Eine der Treffermengen muss als Ausgangsmenge definiert werden. Von dieser werden die anderen Mengen abgezogen.

Symmetrische Differenz: Die Ergebnismenge besteht aus den Objekten, die nur in genau einer Teilmenge enthalten sind (= alles außer dem Schnitt, bei zwei Mengen).

Es können drei unterschiedliche Arten von Gesamttreffern erzeugt werden. Die Auswahl ist insbesondere dann relevant, wenn die Teiltreffer zusätzliche Informationen tragen.

- Einheitliche Treffer erzeugen, ursprüngliche Treffer als Ursache merken: Es werden neue Treffer erzeugt, die den ursprünglichen Treffer als Ursache enthalten.
- Ursprüngliche Treffer erweitern: Der ursprüngliche Treffer wird kopiert und erhält einen neuen Qualitätswert. Falls mehrere Treffer für dasselbe semantische Objekt vorliegen, wird ein beliebiger Treffer gewählt.
- Einheitliche Treffer erzeugen: Es wird ein neuer Treffer erzeugt. Die Eigenschaften des ursprünglichen Treffers gehen verloren.

Teiltreffer zusammenfassen

Bei der Einzelverarbeitung ist es öfters notwendig, eine Gesamtmenge aus Teiltreffern zu erzeugen. Dieses ermöglicht die Komponente „Teiltreffer zusammenfassen“. Diese fasst alle Treffer einer oder mehrerer Teiltreffermengen zusammen. Der Unterschied zu Treffer zusammenfassen liegt darin, dass die Zusammenfassung erst am Ende erfolgt, nicht für jede einzelne Teiltreffermenge. Dies ist insbesondere bei der Berechnung der Qualität relevant, da Treffer zusammenfassen z.B. bei Median falsche Werte liefern würde.

Skript

Eine Such-Pipeline kann ein Skript (JavaScript oder KScript) enthalten. Dieses kann auf die Variablen der Such-Pipeline zugreifen. Außerdem kann ein Skript mehrere Parameter an die Such-Pipeline übergeben. Das Ergebnis des Skripts wird als Ergebnis der Komponente verwendet.

Die JavaScript-API sowie KScript sind in separaten Handbüchern beschrieben.

Qualität aus Attributwert übernehmen

Für Treffer können wir den Qualitätswert aus einem Attribut des semantischen Objekts

übernehmen. Falls das Objekt nicht genau ein solches Attribut besitzt, wird der Standardwert verwendet. Der Wert sollte zwischen 0 und 1 liegen.

Gesamtqualität aus gewichteten Qualitäten berechnen

Um die Qualität eines Suchtreffers anzupassen, kann es hilfreich sein, aus einzelnen Teilqualitäten einen Gesamtwert zu berechnen. Die Qualitäten müssen dabei als Zahlenwerte vorliegen. Aus diesen Werten wird eine neue Gesamtqualität berechnet.

Gesamtqualität einer Treffermenge berechnen

Aus den einzelnen Qualitätswerten einer Treffermenge kann man eine Gesamtqualität berechnen.

Qualität beschränken

Treffermengen können wir auf Treffer beschränken, deren Qualitätswert innerhalb vorgegebener Schranken (Minimum und Maximum) liegen. Im Normalfall möchten wir Treffer, die unterhalb einer bestimmten Qualitätsschwelle liegen, ausfiltern.

Trefferanzahl beschränken

Falls die Gesamtzahl einer Treffermenge begrenzt werden soll, können wir die Komponente „Trefferanzahl beschränken“ hinzufügen. Mit der Option „Treffer gleicher Qualität nicht zerteilen“ verhindern wir, dass bei mehreren Treffern mit gleicher Qualität eine willkürliche Auswahl erfolgt, um die Gesamtzahl einzuhalten. Wir erhalten dann mehr Treffer als vorgegeben.

Für einige sehr spezielle Fälle können wir die Treffer auch zufällig auswählen lassen, z.B. wenn wir eine große Menge an Treffern gleicher Qualität haben und eine Vorschau generieren wollen.

Qualität skalieren

Die Qualitätswerte einer Treffermenge kann skaliert werden. Es wird eine neue Treffermenge mit skalierten Qualitätswerten berechnet. Die Berechnung erfolgt ein zwei Schritten:

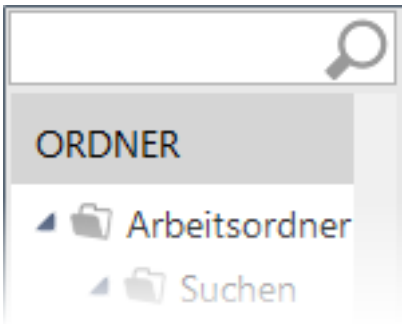
1. Die Qualitätswerte der Treffer werden begrenzt. Die Grenzwerte können entweder festgelegt oder berechnet werden. Bei der Berechnung werden der minimale und der maximale Wert der Treffer ermittelt. Falls die Grenzen vorgegeben werden und ein Treffer einen Qualitätswert außerhalb der Grenzwerte hat, wird der Wert auf den Grenzwert beschränkt. Falls man solche Treffer entfernen will, muss man die Komponente Qualität beschränken vorschalten. Beispiel: Abbilden von Prozentwerten auf Schulnoten. 30% ist Durchschnitt, über 90% ist Highscore. Die Werte können innerhalb von 30% bis 90% linear skaliert werden.
2. Anschließend werden die Qualitätswerte linear skaliert. Treffer mit dem minimalen/maximalen Eingangswert erhalten den minimalen/maximalen skalierten Wert.

Trefferqualität berechnen

Mit Hilfe eines KPath-Ausdrucks wird für einen Treffer ein neuer Treffer mit berechneter Qualität erzeugt. Der KPath-Ausdruck wird ausgehend von der Eingabe berechnet.

1.3.4 Die Suche im Knowledge-Builder

Bis auf die Strukturabfragen, die in den Ordnern angelegt und auch dort ausgeführt werden, können alle Suchen in der Kopfzeile des Knowledge-Builders für die interne Nutzung verfügbar gemacht werden.



Dazu müssen wir eine vorkonfigurierte Suche nur in das Suchfeld in der Knowledge-Builder-Kopfzeile hineinziehen. Stehen dort mehrere Suchen zur Auswahl, kann durch Klick auf das Lupensymbol aus einem PullDown-Menu die gewünschte Suche ausgewählt werden. Das Sucheingabefeld berücksichtigt dabei immer den zuletzt ausgeführten Suchmodus.

Entfernen können wir die Suche über die globalen Einstellungen. Hier können wir auch die Reihenfolge der verschiedenen Suchen im Menu ändern.

1.3.5 Spezialfälle

1.3.5.1 Volltextsuche Lucene

Die Volltextsuche lässt sich auch alternativ über den externen Indexer *Lucene* betreiben. Die Konfiguration der Suche ist dann analog zur normalen Volltextsuche, d.h. es können wiederum Attribute in die Suche konfiguriert werden, die an den Lucene-Index angeschlossen sind; der Suchvorgang erfolgt ebenfalls analog.

1.3.5.2 Suche mit regulären Ausdrücken

Reguläre Ausdrücke sind ein mächtiges Mittel, um je nach Aufgabe Datenbestände nach komplexen Suchausdrücken zu durchforsten.

Suche mit regulärem Ausdruck	Treffer
The [CF]all	The Call, The Fall
Car.	Cars
Car.*	Cars, Caravans, Carmen etc.
[∧R]oom	Doom, Loom etc. (aber nicht Room)

Als Sucheingaben unterstützt i-views dabei den auch aus Perl bekannten Standard, der z.B. im Wikipedia-Artikel zu regulären Ausdrücken beschrieben ist.

1.3.5.3 Suche in Ordnern

Die Suche in Ordnern sucht in Ordnernamen und -inhalten:

- Ordner, deren Name auf die Sucheingabe passt
- Ordner, die Objekte enthalten, die auf die Sucheingabe passen
- Expertensuchen, die Elemente enthalten, auf die die Sucheingabe passt
- Skripte, in denen die Sucheingabe auftaucht
- Rechte- und Triggerdefinitionen, die Elemente enthalten, auf die die Sucheingabe passt

Mit der Sucheingabe #obsolete kann gezielt nach Verwendungen von gelöschten Objekten (z.B. in Rechten, Triggern, Suchen) gesucht werden. Bei der Konfiguration der Suche kann die Menge der zu durchsuchenden Ordner eingeschränkt werden. Außerdem kann die Option „Nach Objektnamen in Ordnern suchen“ deaktiviert werden. Dies ist hilfreich, wenn man nicht nach semantischen Objekten in Ordnern suchen will, da bei umfangreichen Ordnern (z. B. gespeicherte Suchergebnisse) die Suche nach Objektnamen sehr lange dauern kann.

1.4 Ordner und Registrierung

Neben den Objekten und ihren Eigenschaften bauen wir in einem typischen Projekt auch diverse andere Elemente: wir definieren z.B. Abfragen und Importe/Exporte oder schreiben Skripte für spezielle Funktionen. Alles, was wir bauen und konfigurieren, können wir in Ordnern organisieren.

Die Ordner werden geteilt mit allen anderen, die am Projekt arbeiten. Wenn wir das nicht möchten, können wir Dinge im Privatordner ablegen, etwa für Testzwecke. Dieser ist nur für den jeweiligen Nutzer sichtbar.

Eine spezielle Form des Ordners ist die Sammlung semantischer Objekte, in der wir von Hand Objekte z.B. zur späteren Bearbeitung ablegen können. Dazu können wir sie einfach mit Drag&Drop in den Ordner schieben, zudem gibt es Operationen um z.B. Ergebnislisten in Ordnern festzuhalten. Die Sammlung semantischer Objekte hält lediglich Referenzen auf die Objekte: In dem Moment, in dem wir eines dieser Objekte löschen, ist wird es ebenfalls in der Sammlung gelöscht.

Registrierung

Abfragen, Skripte etc. können sich gegenseitig aufrufen (eine Abfrage kann in eine andere Abfrage oder in ein Skript eingebaut werden, umgekehrt kann ein Skript von einer Such-Pipeline aus aufgerufen werden). Zu diesem Zweck gibt es Registrierungsschlüssel, mit denen wir Abfragen, Import-/Export-Abbildungen, Skripte und sogar Sammlungen semantischer Objekte und Strukturordner ausstatten können, damit sie anderen Konfigurationen Funktionalität zur Verfügung stellen. Der Registrierungsschlüssel muss eindeutig sein. Alles, was einen Registrierungsschlüssel hat, wird automatisch in den Ordner „Registrierte Objekte“ aufgenommen bzw. in den seinem Typ entsprechenden Unterordner

Verschieben, Kopieren, Löschen

Nehmen wir an, wir haben in unserem Projekt einen Ordner namens „Playlist-Funktionen“ Dieser enthält vielleicht einen Export, einige Skripte und eine Strukturabfrage „ähnliche Songs“, die wir in einem REST-Services benutzen wollen. In dem Moment, in dem wir der Strukturabfrage einen Registrierungsschlüssel geben, wird sie im Ordner „Registrierte Objekte" (Abschnitt "Technik") aufgenommen. D.h. die Strukturabfrage "ähnliche Songs“ taucht im Ordner „Registrierte Objekt“ unter „Abfrage“ auf. Sie bleibt dort auch, wenn wir sie aus unserem Projekt-Unterordner „Playlist-Funktionen“ entfernen. Entfernen wir den Registrierungsschlüssel, fällt die Abfrage automatisch aus der Registratur heraus.

Das Grundprinzip beim Löschen bzw. Entfernen: Abfragen, Importe, Skripte können in ein oder mehreren Ordnern gleichzeitig und müssen in mindestens in einem Ordner enthalten sei. Erst wenn wir also z.B. unsere Abfrage aus dem letzten Ordner entfernen, wird sie tatsächlich gelöscht. Nur dann bittet i-views auch um eine Bestätigung der Löschaktion. Das gleiche gilt für das Entfernen des Registrierungsschlüssels.

Wollen wir in einem Schritt die Abfrage löschen, unabhängig davon, in wie vielen Ordnern sie sich findet, so können wir das nur von der Registratur aus.

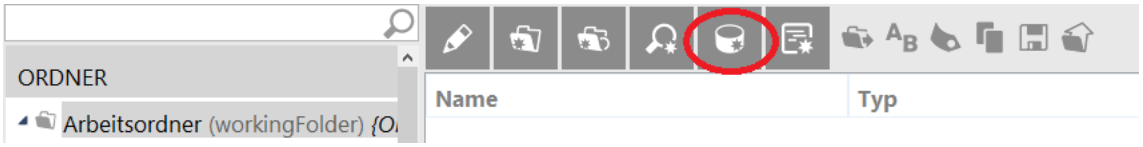
Ordnereinstellungen

In den Ordnereinstellungen können wir für Suchergebnisse, Ordner und Objektlisten (Liste der konkreten Objekte im Hauptfenster des Knowledge-Builders bei Selektion eines Objekttyps auf der linken Seite) Mengengrenzen definieren. Automatische Abfrage bis zur Anzahl Objekte gibt an, bis zu welcher Anzahl von Objekte der Ordnerinhalt oder die Objektliste ohne weitere Interaktion durch den Benutzer gezeigt wird. Ist die dort eingestellte Grenze überschritten, bleibt die Liste zunächst leer und in der Statuszeile erscheint die Meldung „Abfrage nicht ausgeführt“. Suche ausführen ohne Eingabe in die Eingabezeile zeigt alle Objekte. Zumindest bis die zweite Grenze erreicht ist: Maximale Anzahl der Abfrageergebnisse, Maximale Ergebnisanzahl in Objektlisten - hier hohe Werte - ab diesen Werten tatsächlich kein Ergebnis mehr, müssen Abfragen einschränken, z.B. in Objektlisten, indem wir zusätzlich Anfang des Namens im Eingabefeld.

1.5 Import + Export

Mit den Abbildungen von Datenquellen können wir Daten aus strukturierten Quellen in i-views importieren und Objekte und ihre Eigenschaften in strukturierter Form exportieren. Die Quellen können Excel/CSV-Tabellen, Datenbanken oder XML-Strukturen sein.

Die Funktionen für den Import und Export decken sich größtenteils und sind daher alle in einem Editor verfügbar. Um auf die Funktionen für den Import und Export zugreifen zu können, muss zunächst ein Ordner (z.B. der Arbeitsordner) ausgewählt werden. Dort kann über die Schaltfläche "Neue Abbildung einer Datenquelle" eine Datenquelle ausgewählt werden, aus der importiert, bzw. in die exportiert werden soll.



Alternativ findet man die Schaltfläche auch im Reiter "TECHNIK" unter "Registrierte Objekte" -> "Abbildungen von Datenquellen".

Folgende Schnittstellen und Dateiformate stehen für den Import und Export zur Verfügung:

- CSV/Excel-Datei

- XML-Datei
- MySQL-Schnittstelle
- ODBC-Schnittstelle
- Oracle-Schnittstelle
- PostgreSQL-Schnittstelle
- Für den Austausch von Benutzer-IDs ist eine Standard-LDAP-Schnittstelle implementiert.

Im Folgenden wird anhand einer CSV-Datei beschrieben, wie man einen tabellenorientierten Import/Export anlegt. Da bis auf den XML-Import/Export alle Importe/Exporte tabellenorientiert sind und sich die einzelnen Datenquellen ansonsten nur in ihrer Konfiguration unterscheiden, kann das Beispiel der Abbildung der CSV-Datei auch auf die Abbildungen der anderen Datenbanken und Dateiformate übertragen werden.

1.5.1 Abbildungen von Datenquellen

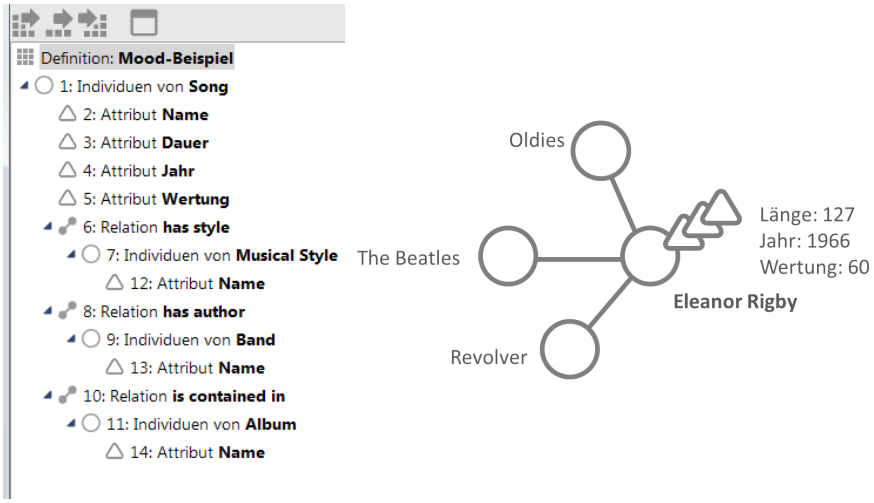
CSV-Dateien sind das Standard-Austauschformat von Tabellenkalkulationstools wie Excel. CSV-Dateien bestehen aus einzelnen Klartext-Zeilen, bei denen die Spalten durch ein fest vorgegebenes Zeichen wie z.B. ein Semikolon getrennt sind.

1.5.1.1 Funktionsprinzip

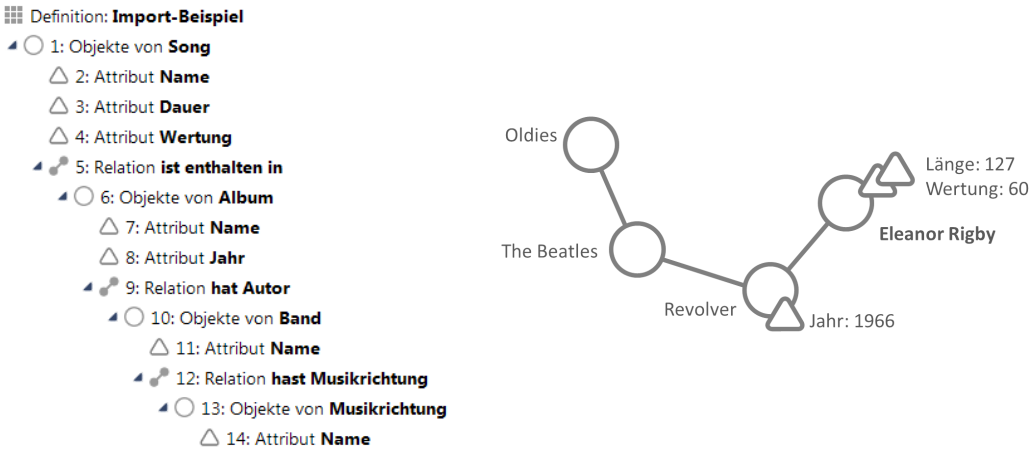
Nehmen wir als erstes Beispiel eine Tabelle mit Songs: Beim Import dieser Tabelle möchten wir für jede Zeile ein neues konkretes Objekt vom Typ Song anlegen. Aus den Inhalten der Spalten B bis G werden Attribute des Songs bzw. Relationen zu anderen Objekten:

	A	B	C	D	E	F	G	H
1	**Titelname**	**Interpret**	**Album**	**Genre**	**Dauer**	**Jahr**	**Meine Wertung**	
2	The Suburbs	Arcade Fire	The Suburbs	Postwave	315	2010	60	
3	Ready To Start	Arcade Fire	The Suburbs	Postwave	255	2010	80	
4	Modern Man	Arcade Fire	The Suburbs	Postwave	279	2010	60	
5	Rococo	Arcade Fire	The Suburbs	Postwave	236	2010	40	
6	Empty Room	Arcade Fire	The Suburbs	Postwave	171	2010	20	
7	City With No Children	Arcade Fire	The Suburbs	Postwave	191	2010	20	
8	Half Light I	Arcade Fire	The Suburbs	Postwave	253	2010	20	
9	Half Light II (No Celebration)	Arcade Fire	The Suburbs	Postwave	267	2010	40	
10	Suburban War	Arcade Fire	The Suburbs	Postwave	281	2010	80	
11	Month Of May	Arcade Fire	The Suburbs	Postwave	230	2010	20	
12	Wasted Hours	Arcade Fire	The Suburbs	Postwave	200	2010	40	
13	Deep Blue	Arcade Fire	The Suburbs	Postwave	268	2010	60	
14	We Used To Wait	Arcade Fire	The Suburbs	Postwave	301	2010	100	
15	Sprawl I (Flatland)	Arcade Fire	The Suburbs	Postwave	174	2010	40	
16	Sprawl II (Mountains Beyond Mountains)	Arcade Fire	The Suburbs	Postwave	318	2010	40	
17	The Suburbs (Continued)	Arcade Fire	The Suburbs	Postwave	87	2010	40	
18	Eleanor Rigby	The Beatles	Revolver	Oldies	127	1966	60	
19	For No One	The Beatles	Revolver	Oldies	121	1966	60	
20	Good Day Sunshine	The Beatles	Revolver	Oldies	129	1966	40	
21	Here There And Everywhere	The Beatles	Revolver	Oldies	145	1966	40	
22	I Want To Tell You	The Beatles	Revolver	Oldies	149	1966	40	
23	I'm Only Sleeping	The Beatles	Revolver	Oldies	181	1966	60	
24	Love To You	The Beatles	Revolver	Oldies	181	1966	20	
25	She Said She Said	The Beatles	Revolver	Oldies	157	1966	40	
26	Taxman	The Beatles	Revolver	Oldies	159	1966	20	
27	Tomorrow Never Knows	The Beatles	Revolver	Oldies	177	1966	20	
28	Yellow Submarine	The Beatles	Revolver	Oldies	160	1966	20	
29	About A Girl	Nirvana	MTV Unplugged in NY	Rock	217	1994	60	
30	Jesus Doesn't Want Me For A Su	Nirvana	MTV Unplugged in NY	Rock	277	1994	40	
31	The Man Who Sold The World	Nirvana	MTV Unplugged in NY	Rock	260	1994	80	
32	Pennyroyal Tea	Nirvana	MTV Unplugged in NY	Rock	220	1994	60	
33	Dumb	Nirvana	MTV Unplugged in NY	Rock	172	1994	40	
34	Polly	Nirvana	MTV Unplugged in NY	Rock	196	1994	60	

Ausgehend vom Song, bauen wir die Struktur von Attributen, Relationen und Zielobjekten auf, die durch den Import angelegt werden soll (linke Seite). So wird für die Zeile 18 beispielsweise ein Objekt vom Typ Song mit folgenden Attributen und Beziehungen angelegt:



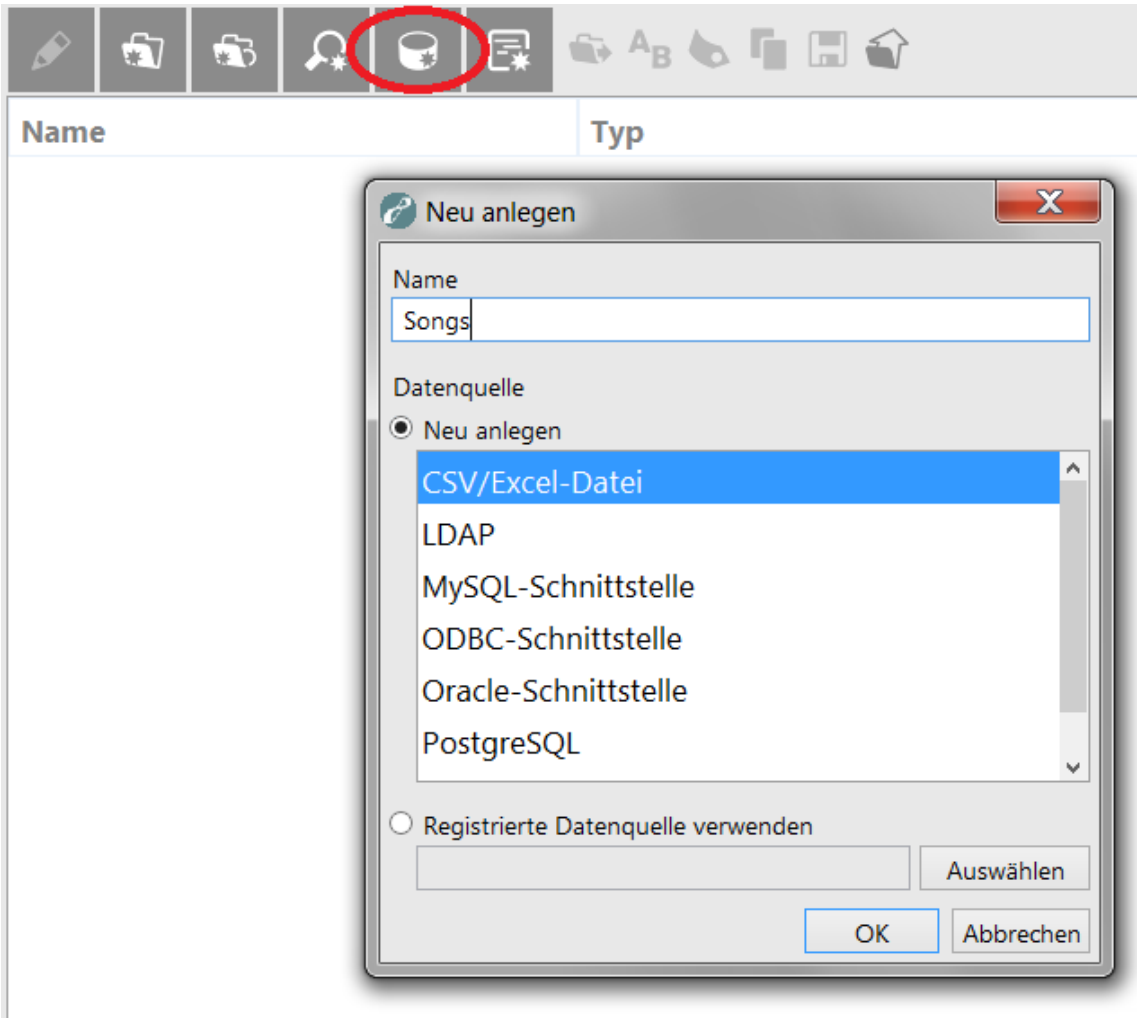
Wir können uns aber auch dafür entscheiden, die Angaben aus der Tabelle anders zu verteilen also z.B. Erscheinungsjahr und Interpret dem Album zuordnen und das Genre wiederum dem Interpreten. Eine Zeile bildet immer noch einen Kontext, muss deswegen aber nicht zu genau einem Objekt gehören:



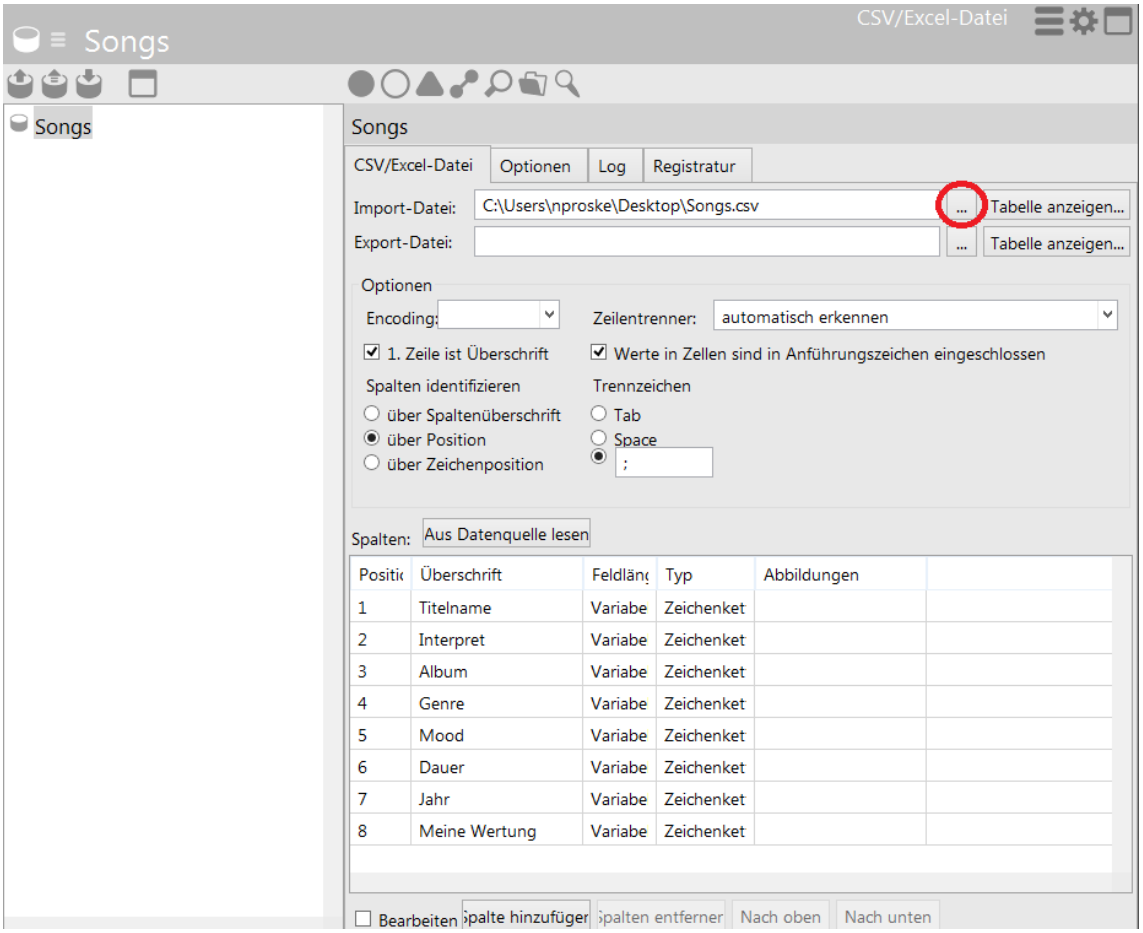
Überall, wo wir in unserem Beispiel neue konkrete Objekte als Relationsziele aufbauen, müssen wir immer mindestens ein Attribut zu diesem Objekt angeben, hier jeweils das Namensattribut, mit dem wir das entsprechende Objekt identifizieren können.

1.5.1.2 Datenquelle - Auswahl und Optionen

Nachdem wir, die Schaltfläche "*Neue Abbildung einer Datenquelle*" ausgewählt haben, öffnet sich ein Dialog, mit dem wir die Art der Datenquelle und den Namen der Abbildung angeben müssen. Haben wir die Datenquelle bereits in der semantischen Graph-Datenbank registriert, können wir diese im unteren Auswahlmenü finden.



Mit der Bestätigung auf "OK" öffnet sich der Editor für den Import und Export. Unter "Import-Datei" können wir den Pfad unserer zu importierenden Datei angeben. Alternativ können wir die Datei auch über den Button rechts daneben auswählen. Sobald die Datei ausgewählt wurde, werden die Spaltenüberschriften und ihre Positionen in der Tabelle ausgelesen und im Feld rechts unten angezeigt. Die Schaltfläche "*Aus Datenquelle lesen*" kann bei eventuellen Änderungen der Datenquelle die Spalten erneut auslesen. Die Spalte "Abbildungen" zeigt uns später jeweils auf welches Attribut die jeweilige Spalte der Tabelle abgebildet wird.



Die Struktur unserer Beispiel-Tabelle entspricht komplett den Standard-Einstellungen, sodass wir unter dem Menüpunkt *Optionen* nichts weiter berücksichtigen müssen. CSV-Dateien können jedoch sehr unterschiedliche Strukturen aufweisen, die mit folgenden Einstellungsmöglichkeiten berücksichtigt werden müssen:

Encoding: Hier wird die Zeichenkodierung der Import-Datei festgelegt. Zur Auswahl stehen ascii, ISO-8859-1, ISO-8859-15, UCS-2, UTF-16, UTF-8 und windows-1252. Ist nichts ausgewählt, wird die Standard-Einstellung übernommen, die der des laufenden Betriebssystems entspricht.

Zeilentrenner: In den meisten Fällen reicht die Einstellung "automatisch erkennen", die auch standardmäßig ausgewählt ist. Sollte man jedoch feststellen, dass Zeilenumbrüche nicht richtig erkannt werden, sollte man die entsprechende korrekte Einstellung manuell auswählen. Zur Auswahl stehen *CR* (*carriage return*, engl. für Wagenrücklauf), *LF* (*line feed*, engl. für Zeilenvorschub), *CR-LF* und *Keine*. Der Standard, der den Zeilenumbruch in einer Textdatei kodiert, ist bei Unix, Linux, Android, Mac OS X, AmigaOS, BSD und weiteren *LF*, bei Windows, DOS, OS/2, CP/M und TOS (Atari) *CR-LF* und bei Mac OS bis Version 9, Apple II und C64 *CR*.

1. Zeile ist Überschrift: Es kann vorkommen, dass die erste Zeile keine Überschrift enthält, was mit dem Entfernen des standardmäßig gesetzten Häkchens bei "*1. Zeile ist Überschrift*" dem System mitgeteilt werden muss.

Werte in Zellen sind in Anführungszeichen eingeschlossen wählt man aus, damit die Anführungszeichen nicht mit importiert werden, wenn man das nicht möchte.

Spalten identifizieren: Ob die Spalten über ihre Überschrift, die Position oder die Zeichenpo-

sition identifiziert werden, muss angegeben werden, da ansonsten die Tabelle nicht korrekt erfasst werden kann.

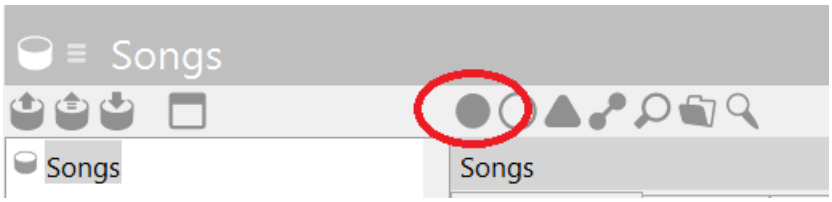
Trennzeichen: Falls ein anderes Trennzeichen als das standardmäßige Semikolon verwendet wird, muss dies ebenfalls angegeben werden, sofern die Spalte nicht über die Zeichenposition identifiziert wird.

Darüber hinaus gelten folgende Regeln: Falls ein Wert der Tabelle das Trennzeichen oder einen Zeilenumbruch enthält, muss der Wert in doppelte Anführungszeichen gestellt werden. Falls der Wert ein Anführungszeichen enthält, muss dieses verdoppelt (»""«) werden.

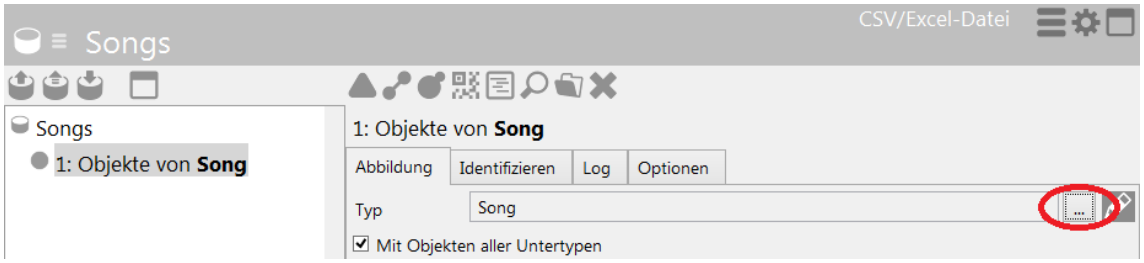
1.5.1.3 Definition von Zielstruktur und Abbildungen

1.5.1.3.1 Die Objektabbildung

Nun fangen wir an die Zielstruktur, die in der semantischen Graph-Datenbank entstehen soll, aufzubauen. In unserem Beispiel beginnen wir mit einer Objektabbildung der Songs. Um ein neues Objekt abzubilden müssen wir den Button "Neue Objektabbildung" bestätigen.

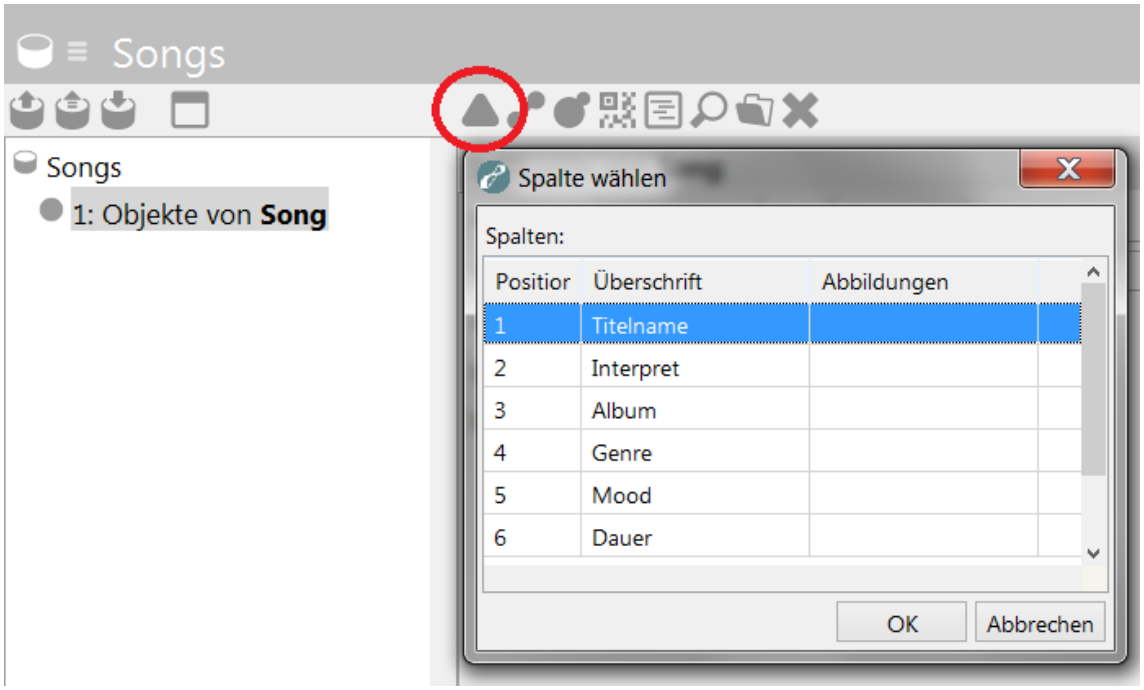


Als nächstes muss der Typ des zu importierenden Objektes angegeben werden. Ist der Haken bei "Mit Objekten aller Untertypen" gesetzt, werden beim Import auch Objekte aus allen Untertypen von "Song" berücksichtigt.

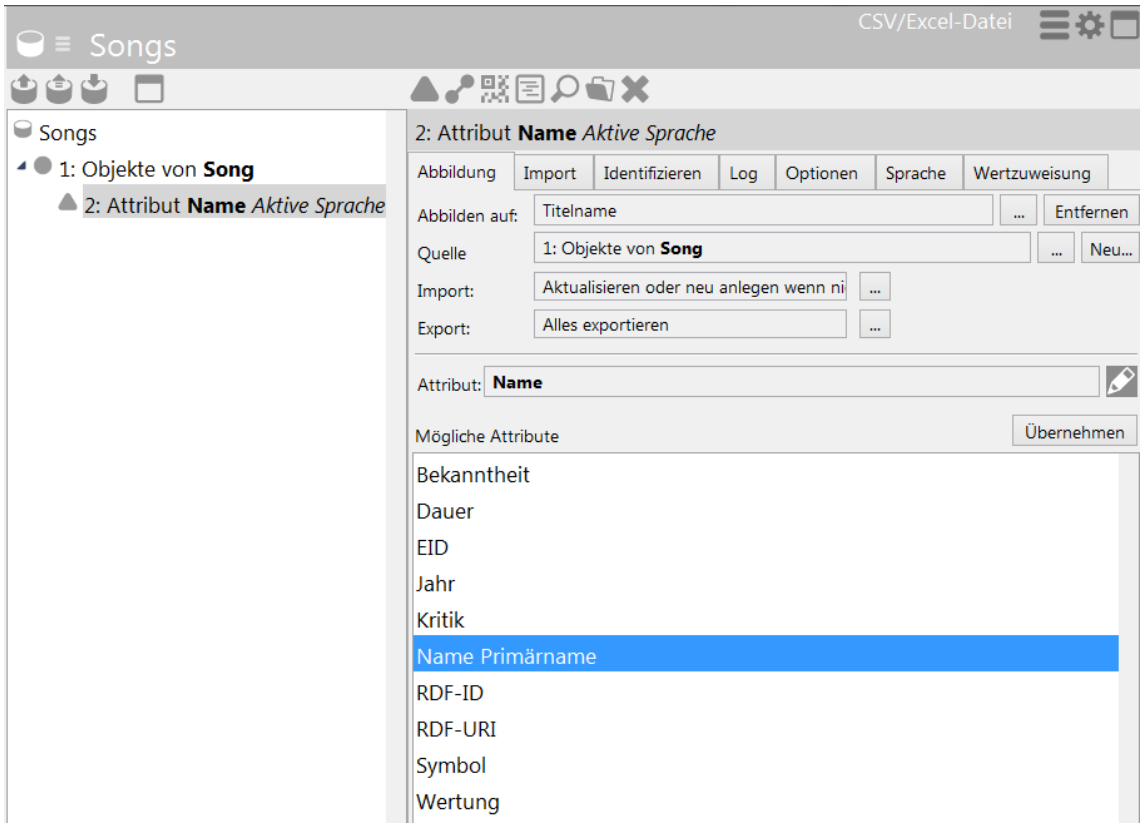


1.5.1.3.2 Die Attributabbildung / Identifizieren von Objekten

Nun wollen wir die in der Tabelle enthaltenen Informationen mit der Objektabbildung der Songs verknüpfen. Es sind sowohl Attribute zu den einzelnen Songs, als auch Relationen vertreten. Um zunächst den Titelnamen eines Songs in der Abbildung anzulegen, fügen wir der Objektabbildung von Song ein Attribut hinzu. Ein Klick auf die Schaltfläche "*Neue Attributabbildung*" öffnet einen Dialog, mit dem die entsprechende Spalte aus der zu importierenden Tabelle ausgewählt werden muss.



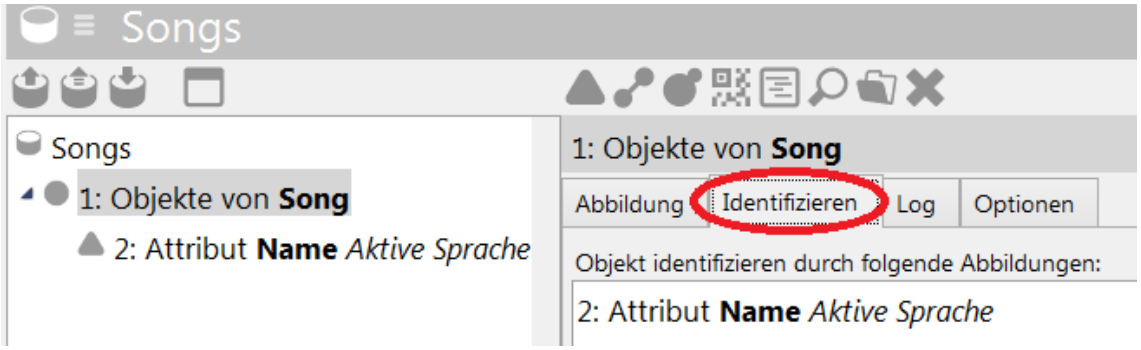
Da dieses Attribut, das erste ist, das wir zu der Objektabbildung von Songs angelegt haben, wird es daraufhin automatisch auf den Namen des Objekts abgebildet, da der Name in der Regel das meistgenutzte Attribut ist.



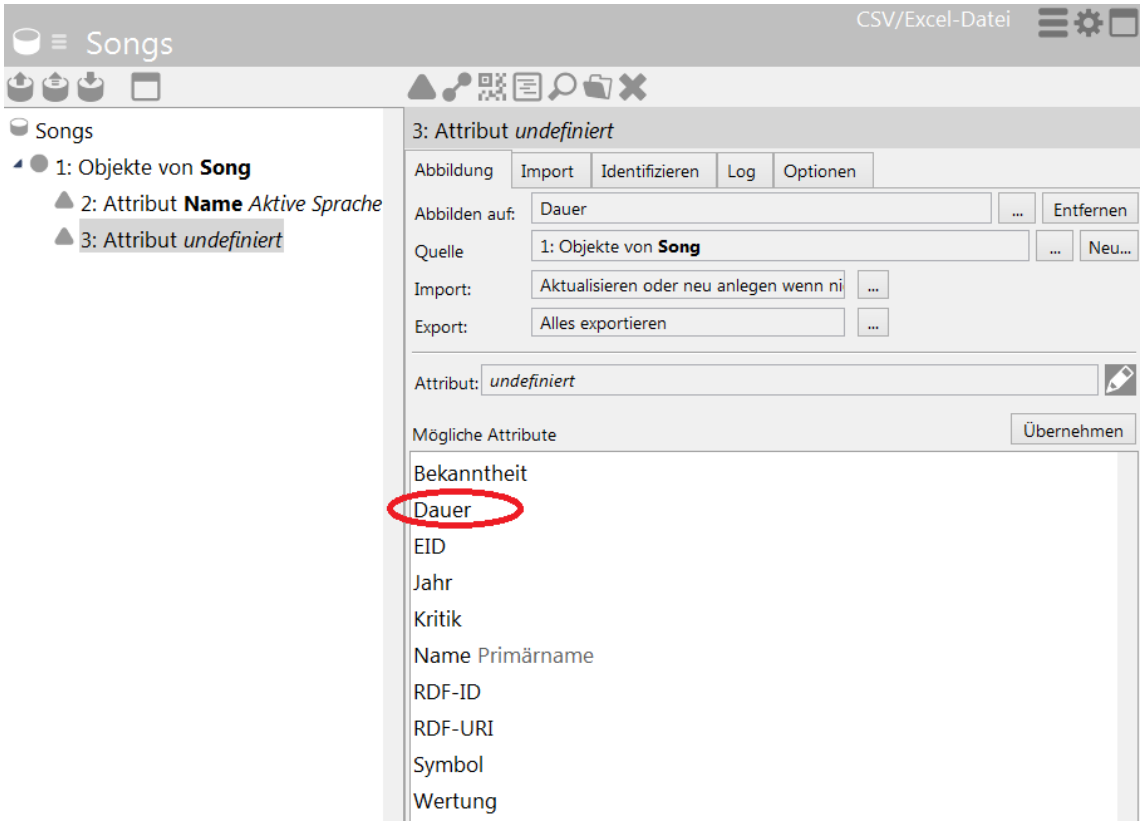
Das erste zu einem Objekt angelegte Attribut wird zudem automatisch zur **Identifizierung des Objektes** verwendet.

Ein Objekt muss über mindestens ein Attribut identifiziert werden - sei es über seinen Namen oder seine ID oder eine Kombination aus mehreren Attributen (wie bei Personen aus Vor- und Nachname und dem Geburtsdatum)-, damit es in der semantischen Graph-Datenbank eindeutig wiedergefunden werden kann, sollte es bereits vorhanden sein. So wird das ungewollte Anlegen von Dubletten beim Import vermieden.

Im Reiter "*Identifizieren*" kann das Attribut, das das Objekt identifiziert noch nachträglich geändert oder mehrere Attribute hinzugefügt werden. Zudem kann hier eingestellt werden, ob die Groß- und Kleinschreibung beim Abgleich der Werte beachtet werden soll und ob nach exakt gleichen Werten gesucht werden soll (ohne Indexfilter/Wildcards). Letzteres ist dann relevant, wenn im Index Filter oder Wildcards definiert sind, die z.B. festlegen, dass ein Bindestrich im Index entfallen soll. Der Begriff würde mit Bindestrich nicht gefunden werden, wenn nur über den Index gesucht wird, also müsste in diesem Fall hier der Haken gesetzt werden, sodass nach dem exakt gleichen Wert gesucht wird.

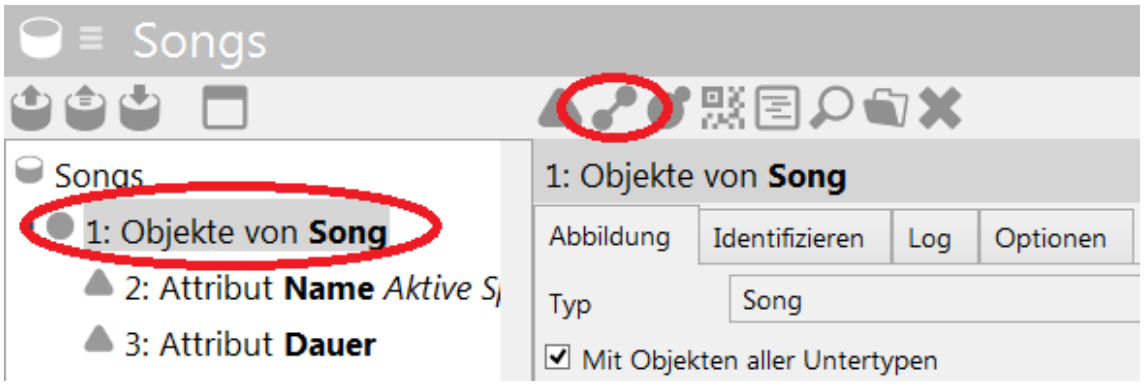


Jetzt können wir dem Objektmapping weitere Attribute hinzufügen, die nicht zur Identifizierung beitragen müssen, z.B. die Dauer eines Songs - dies geschieht wieder über die Schaltfläche "Neue Attributabbildung". (Achtung: Die Objektabbildung "Objekte von Song" muss zunächst wieder ausgewählt werden.) Jetzt wählen wir die Spalte "Dauer" aus der zu importierenden Tabelle aus. Dieses Mal müssen wir das Attribut, auf das die Spalte "Dauer" abgebildet werden soll, manuell auswählen. In dem Feld rechts unten befindet sich die Lister aller im Schema festgelegten möglichen Attribute, die uns für Objekte des Typs "Song" zur Auswahl stehen, darunter auch das Attribut "Dauer".

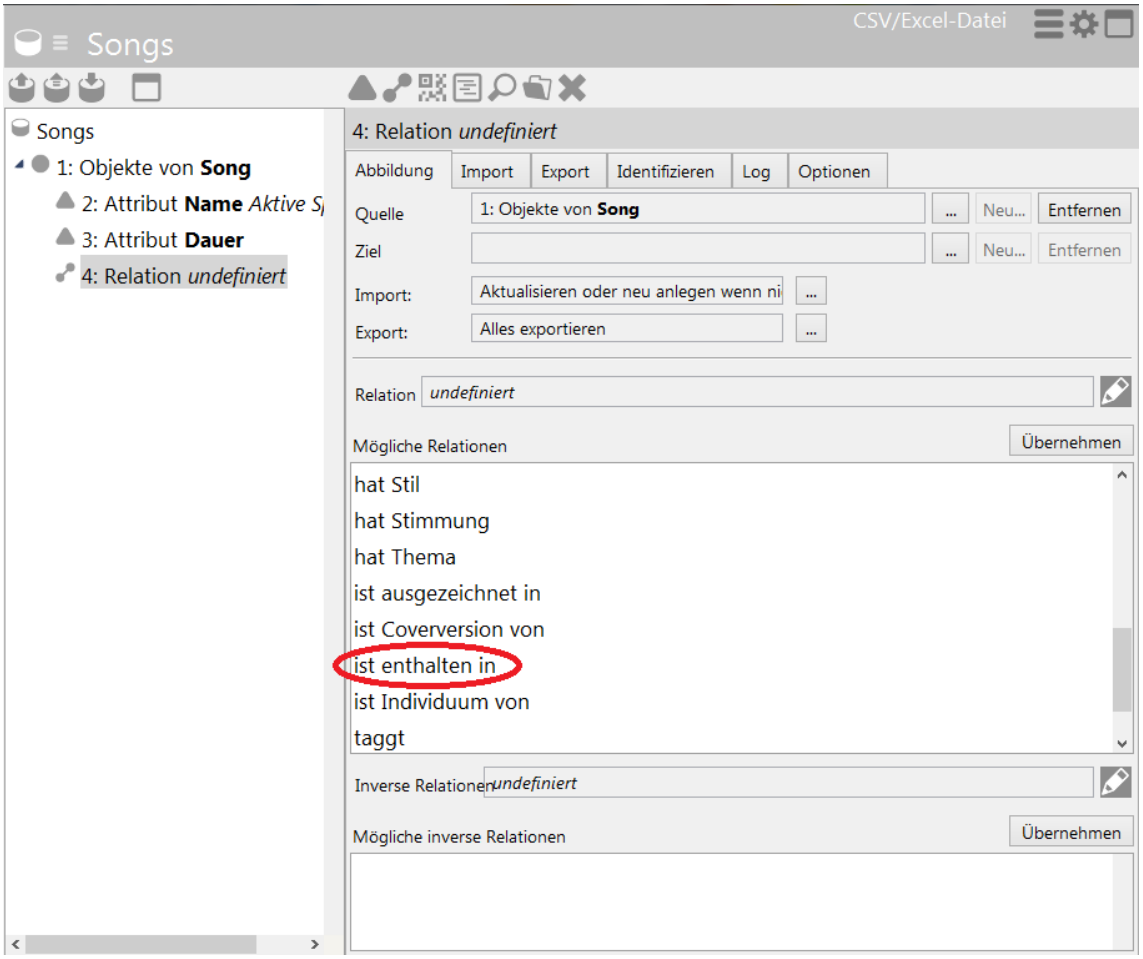


1.5.1.3.3 Die Relationsabbildung

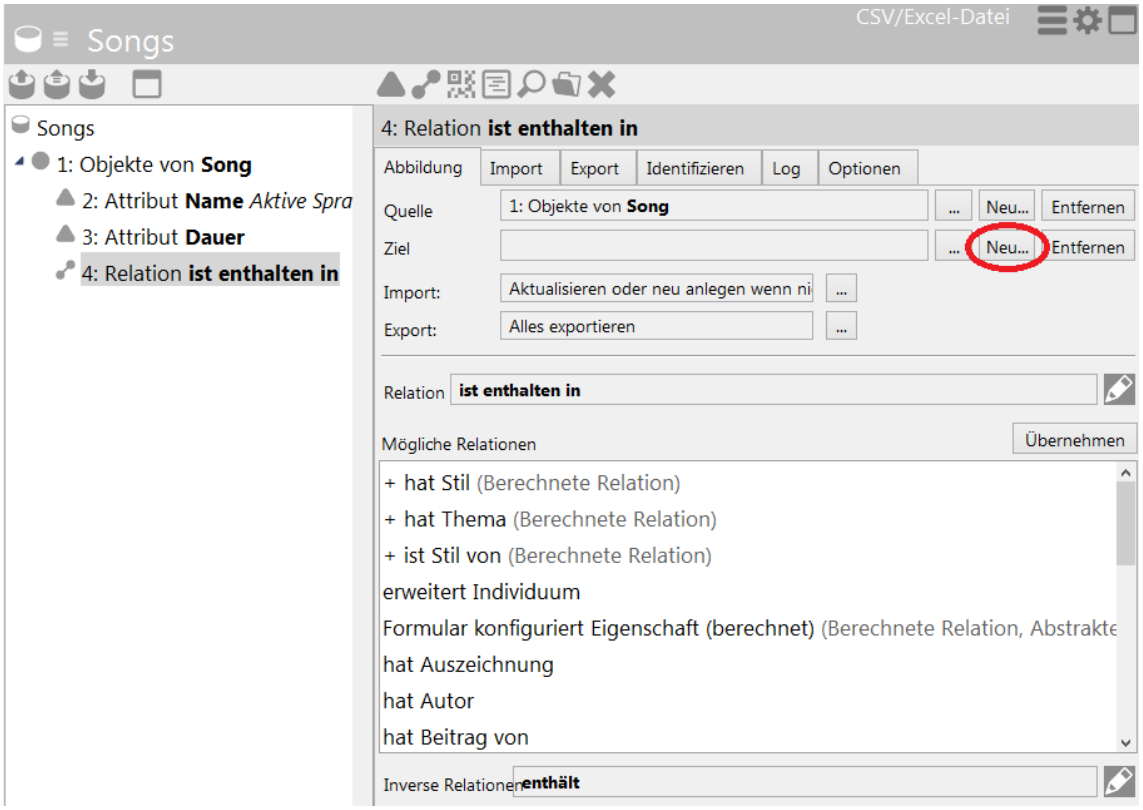
Als nächstes wollen wir das Album abbilden, auf dem der Song sich befindet. Da Alben eigene konkrete Objekte in der semantischen Graph-Datenbank sind, benötigen wir hierfür die Relation, die den Song und das Album verbindet. Um eine Relation abzubilden wählen wir zunächst das Objekt aus, für das die Relation definiert wird und klicken dann auf die Schaltfläche "*Neue Relationsabbildung*".



Daraufhin erhalten wir - wie bei den Attributen - eine Liste aller möglichen Relationen, auch die benötigte Relation "*ist enthalten in*" ist selbstverständlich dabei.



Nun müssen wir im nächsten Schritt festlegen, wo aus der Tabelle die Zielobjekte herkommen. Für das Ziel wird eine neue Objektabbildung gebraucht, die über die Schaltfläche "Neu" angelegt wird. Ist der Typ des Zielobjektes eindeutig im Schema definiert, wird dieser automatisch übernommen, ansonsten erscheint eine Liste der möglichen Objekttypen.

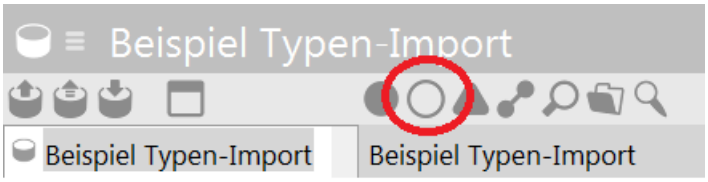


Bei der neuen Objektabbildung müssen wir anschließend wieder das Attribut auswählen, das das Zielobjekt identifiziert usw. So wird die Zielstruktur des Imports aufgebaut.

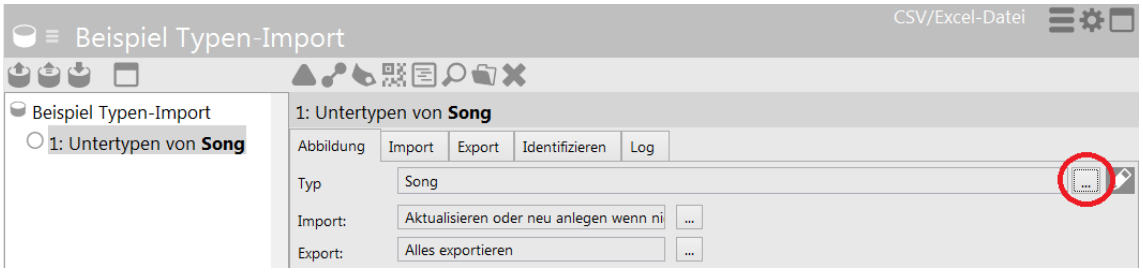
1.5.1.3.4 Die Typabbildung

Auch Typen können importiert und exportiert werden. Nehmen wir beispielhaft an, wir wollten die Genres der Songs als Typen importieren.

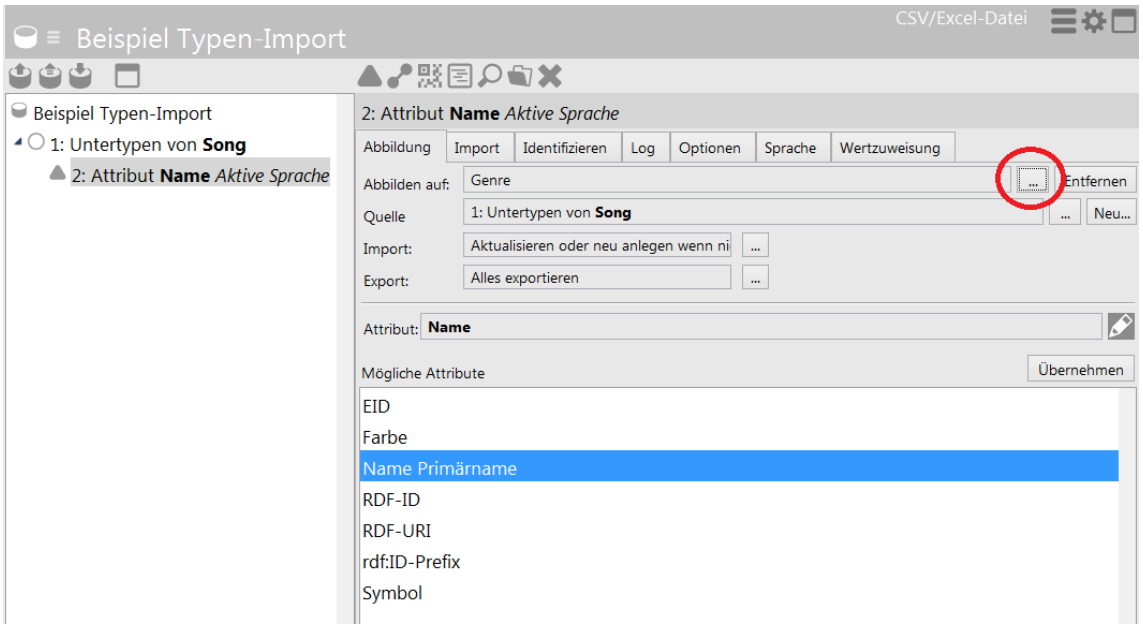
Um einen neuen Typen abzubilden, wählen wir den Button "Neue Typabbildung".



Daraufhin müssen wir den Obertyp der neu anzulegenden Typen angeben, in unserem Beispiel wäre der Obertyp "Song":

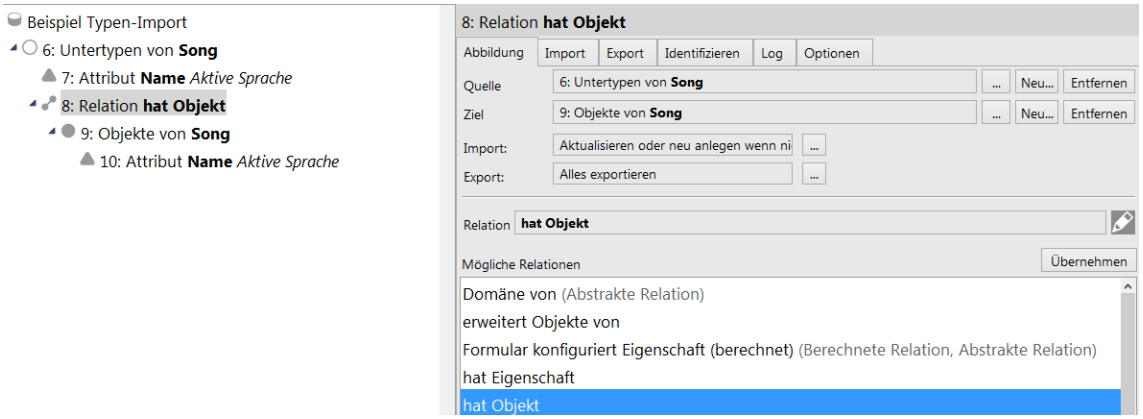


Anschließend müssen wir angeben, aus welcher Spalte der importierten Tabelle der Name unserer neuen Typen entnommen werden soll:



Schließlich müssen wir unter dem Reiter "Import" noch angeben, dass unsere neuen Typen nicht abstrakt sein sollen:

Wenn wir nun die entsprechenden Songs ihren neuen Typen zuordnen wollen, müssen wir die Systemrelation "hat Objekt" verwenden. In älteren Versionen von i-views heißt diese Relation "hat Individuum". Als Ziel wählen wir alle Objekte von Song (inkl. der Untertypen) aus, die sich über das Attribut Name entsprechend der Spalte Songtitel definieren.

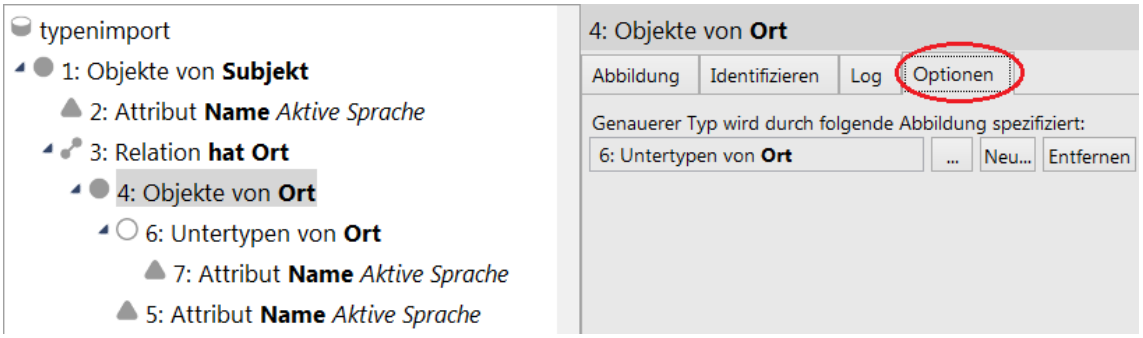


Importieren wir nun diese Abbildung erhalten wir das gewünschte Ergebnis. Die Songs, die bereits in der semantischen Graph-Datenbank vorhanden sind, werden durch die Import-Einstellung "Aktualisieren oder neu anlegen wenn noch nicht vorhanden" berücksichtigt und unter ihren neuen entsprechenden Typ geschoben, sodass kein Objekt doppelt angelegt wird (siehe Kapitel Einstellungen des Import-Verhaltens). Zur Erinnerung: Ein konkretes Objekt kann nicht mehreren Typen gleichzeitig angehören.

Es gibt noch einen Spezialfall. Angenommen, wir haben eine Tabelle, in der in einer Spalte verschiedene Typen vorkommen, dann können wir auch dies in unseren Importeinstellungen abbilden.

Person/Band	Herkunft	Typ des Ortes
Paul McCarntey	Liverpool	Stadt
The Beatles	Großbritannien	Land

Dazu wählen wir die Abbildung der Objekte aus, denen wir die Untertypen zuordnen wollen (in diesem Fall "Objekte von Ort") und wählen dann unter dem Reiter "Optionen" den entsprechenden Obertyp aus.



Wichtig ist auch hier wieder nicht zu vergessen, unter dem Reiter "Import" festzulegen, dass der Typ nicht abstrakt sein soll, damit konkrete Objekte angelegt werden können.

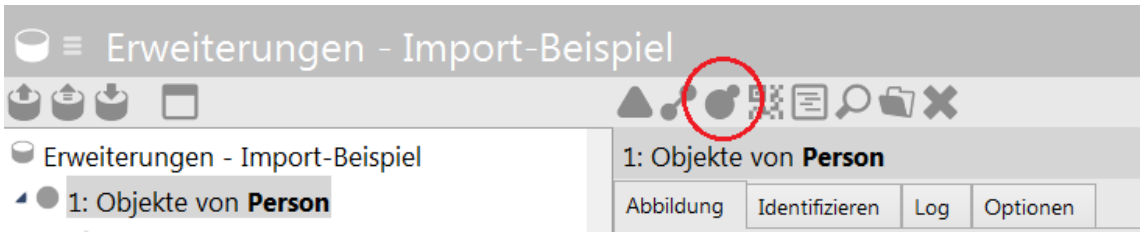
Vorsicht: Angenommen, Liverpool existiert bereits im Wissensnetz, ist jedoch dem Typ "Ort" zugeordnet, da dieser bis zu diesem Zeitpunkt noch keine Untertypen wie "Stadt" und "Land" besessen hat. In diesem Fall wird Liverpool **nicht** unter dem Typ Stadt neu angelegt. Begründung: die Objekte des Typs Ort werden lediglich über das Namensattribut identifiziert, nicht jedoch über den Untertyp.

1.5.1.3.5 Abbildung von Erweiterungen

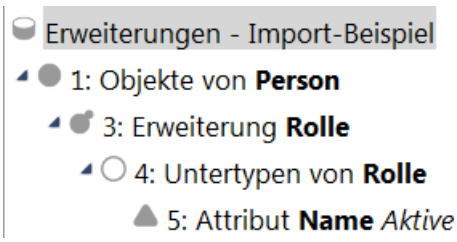
Auch Erweiterungen können importiert und exportiert werden. Angenommen, wir haben eine Tabelle, die die Rolle eines Bandmitglieds in einer Band zeigt:

Person	Band	Rolle
Ron Wood	Faces	Gitarrist
Ron Wood	Jeff Beck Group	Bassist
Ron Wood	Rolling Stones	Gitarrist

Ron Wood ist Gitarrist bei den Faces und den Rolling Stones aber Bassist bei der Jeff Beck Group. Um dies abzubilden müssen wir das Objekt auswählen, zu dem im Schema eine Erweiterung definiert wurde und dann den Button "Neue Erweiterungsabbildung" betätigen.

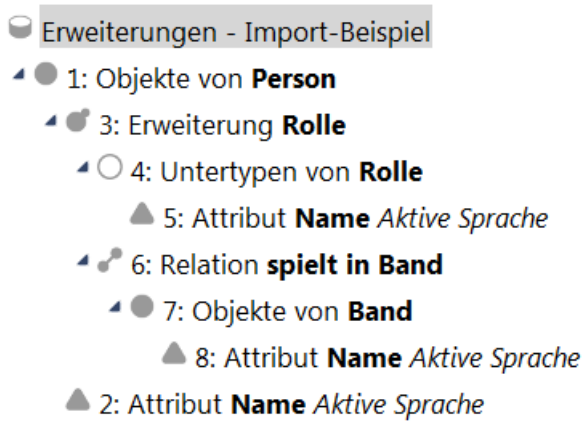


Die Abbildung einer Erweiterung fragt - wie eine Objektabbildung - einen zugehörigen Typen ab. Im Schema des Musik-Netzes ist der Typ "Rolle" ein abstrakter Typ. Deswegen muss in der Abbildung definiert werden, dass die Rolle auf Untertypen des Typs "Rolle" abgebildet weden sollen (siehe Kapitel Die Typabbildung).



Die Relation kann - wie auch bei Objekten und Typen - an der Erweiterung (bzw. an den

Untertypen einer Erweiterung) abgebildet werden.

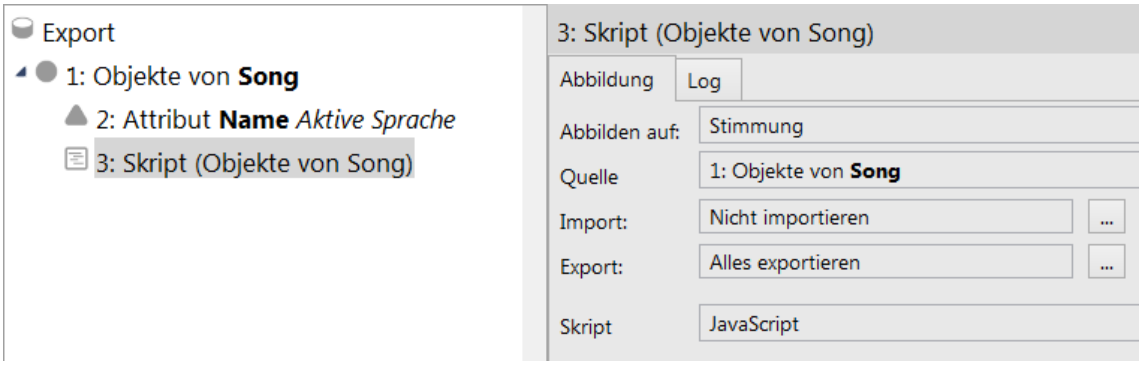


1.5.1.3.6 Die Skriptabbildung

Die Skriptabbildung kann ausschließlich beim Export verwendet werden. Das Skript kann entweder in JavaScript oder KScript geschrieben sein.

Die Skriptabbildung findet beispielsweise dann Verwendung, wenn wir drei Attribute aus der semantischen Graph-Datenbank zu einer ID zusammensetzen wollen. Allerdings kann es sein, dass der Export dann langsamer ist. (Bei einem Import könnte man dies einfacher über eine virtuelle Eigenschaft abbilden. Die Verwendung von virtuellen Eigenschaften wird im Kapitel Tabellenspalten erklärt.)

Der folgende Fall ist ein weiteres Beispiel für die Verwendung eines Skripts bei einem Export. Es zeigt wie mehrere Eigenschaften mit einem Trennzeichen in eine Zelle geschrieben werden können. In diesem Fall wollen wir eine Tabelle erzeugen, die in der ersten Spalte die Songnamen und in der zweiten Spalte alle Stimmungen der Songs mit Komma getrennt aufführt:



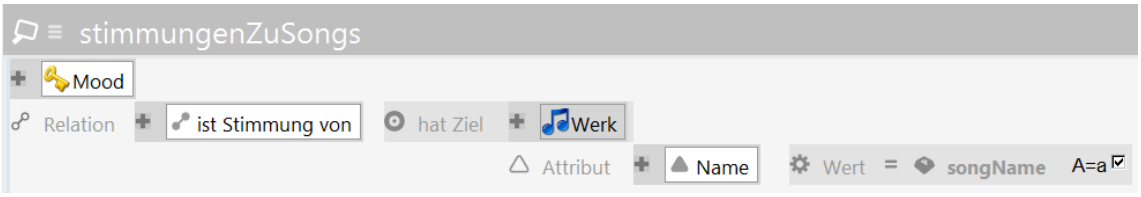
Um die zweite Spalte zu erzeugen benötigen wir folgendes Skript:

```
function exportValueOf(element)
{

    var stimmung = "";
    var relTargets = $k.Registry.query("stimmungenZuSongs").findElements({songName: element.attrib
    if(relTargets && relTargets.length > 0){
        for(var i = 0; i < (relTargets.length-1); i++){
            stimmung += relTargets[i].attributeValue("objektName") + ", ";
```

```
        }
        stimmung += relTargets[relTargets.length-1].attributeValue("objektName");
    }
    return stimmung;
}
```

Das Skript beinhaltet folgende Strukturabfrage (Registrierungsschlüssel: "stimmungenZuSongs"):



Über den Ausdruck "findElements" können wir auf einen Parameter (hier "songName") innerhalb der Abfrage zugreifen. "objektName" ist der interne Name des Namensattributs in diesem semantischen Modell.

Innerhalb der if-Anweisung sagen wir, dass wenn ein Element mehrere Relationsziele hat, diese durch ein Komma getrennt dargestellt werden sollen. Nach dem letzten Relationsziel, das die Schleife durchläuft, soll keine Komma mehr stehen. Auch wenn ein Element nur ein Relationsziel hat, wird dies demnach ohne Komma dargestellt.

Das Ergebnis ist eine Liste der Songs mit allen ihren Stimmungen, die durch Komma getrennt in der zweiten Spalte der Tabelle stehen:

Songtitel	Stimmung
Black Country Rock	
19th Nervous Breakdown	
A Manic Depressive Named Laughing Boy	
A Place for my Head	Aggressiv
All the Madmen	
Bipolar	
Bleed It Out	
Bleed Like Me	
Breaking the Habit	
By Myself	Aggressiv
Back To Black	Dramatisch, Bittersüß, Schwungvoll
China Girl (Bowie)	Melancholisch/Düster, Kalt
Climbing up the Walls	
Crawling	Aggressiv
Creep	Hymnisch, Elegisch, Dramatisch, Lethargisch, Melancholisch/Düster
Digging In The Dirt	

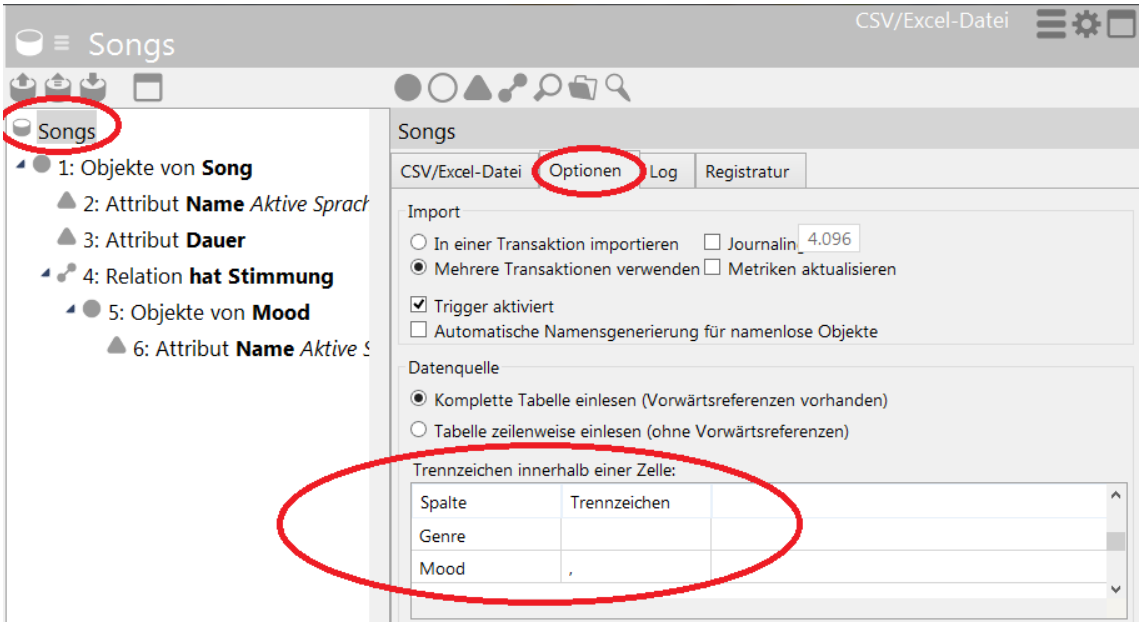
1.5.1.4 Abbildung von mehreren Werten für einen Objekttyp bei einem Objekt

Wenn für einen Objekttyp bei einem Objekt mehrere Werte angegeben sind (in unserem Beispiel etwa mehrere „Moods“ für jeden Song), dann gibt es drei Möglichkeiten, wie die Tabelle aussehen kann. Für zwei der drei Möglichkeiten muss der Import angepasst werden, was im Folgenden beschrieben ist.

Möglichkeit 1 - Trennzeichengetrennte Werte: Die einzelnen Werte befinden sich in einer Zelle und sind durch ein Trennzeichen (z.B. ein Komma) getrennt.

	A	B	C	D	E
1	**Titelname**	**Genre**	**Mood**	**Dauer**	**Jahr**
2	Eleanor Rigby	Oldies	Reflective, Dreamy	127	1966
3	For No One	Oldies	Acerbic	121	1966
4	I'm Only Sleeping	Oldies	Quirky, Mellow	181	1966
5	Yellow Submarine	Oldies	Spacey, Trippy, Playful	160	1966

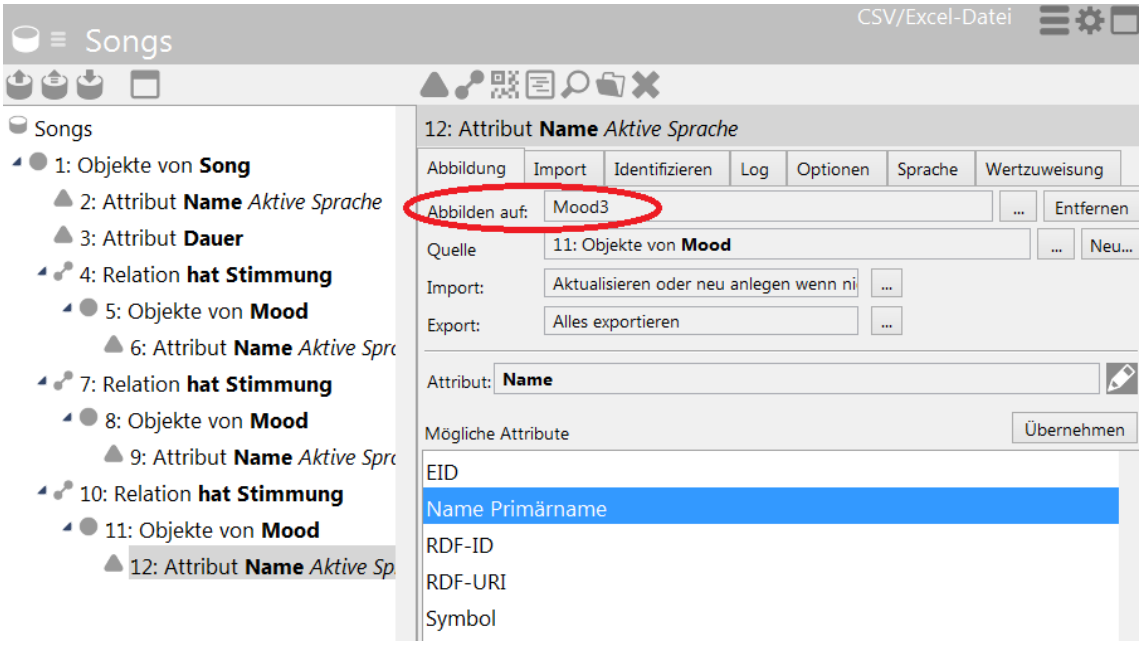
In diesem Fall gehen wir auf die Abbildung der Datenquelle, wo sich die allgemeinen Einstellungen befinden und dort auf den Reiter "*Optionen*". Hier finden wir im unteren Bereich die Einstellungsmöglichkeit, Trennzeichen innerhalb einer Zelle anzugeben. Nun müssen wir nur noch die entsprechende Spalte der zu importierenden Tabelle heraussuchen ("Mood") und in die Spalte "*Trennzeichen*" das verwendete Trennzeichen (",") eingeben.



Möglichkeit 2 - Mehrere Spalten: Die einzelnen Werte befinden sich jeweils in einer eigenen Spalte, wobei nicht jedes Feld ausgefüllt sein muss. Es werden so viele Spalten benötigt, wie maximal Moods pro Song vorhanden sind.

	A	B	C	D	E	F	G
1	**Titelname**	**Genre**	**Mood**	**Mood2**	**Mood3**	**Dauer**	**Jahr**
2	Eleanor Rigby	Oldies	Reflective	Dreamy		127	1966
3	For No One	Oldies	Acerbic			121	1966
4	I'm Only Sleeping	Oldies	Quirky	Mellow		181	1966
5	Yellow Submarine	Oldies	Spacey	Trippy	Playful	160	1966

In diesem Fall muss die entsprechende Relation so oft angelegt werden, wie Spalten vorhanden sind. In diesem Beispiel müssen demnach die erste Relation auf "Mood1", die zweite Relation auf "Mood2" und die dritte Relation auf "Mood3" abgebildet werden.



Möglichkeit 3 - Mehrere Zeilen: Die einzelnen Werte befinden sich jeweils in einer eigene Zeile. Achtung: Hierbei ist es zwingend nötig, dass die Attribute, die für die Identifizierung des Objektes benötigt werden (in diesem Fall der Titelname), in jeder Zeile auftreten, ansonsten würden die Zeilen als jeweils eigenes Objekt ohne Name gedeutet werden und ein korrekter Import wäre nicht möglich.

	A	B	C	D	E
1	**Titelname**	**Genre**	**Mood**	**Dauer**	**Jahr**
2	Eleanor Rigby	Oldies	Reflective	127	1966
3	Eleanor Rigby		Dreamy		
4	For No One	Oldies	Acerbic	121	1966
5	I'm Only Sleeping	Oldies	Quirky	181	1966
6	I'm Only Sleeping		Mellow		
7	Yellow Submarine	Oldies	Spacey	160	1966
8	Yellow Submarine		Trippy		
9	Yellow Submarine		Playful		

In diesem Fall sind keine besonderen Import-Einstellungen nötig, da das System über das identifizierende Attribut das Objekt erkennt und die Relationen korrekt zieht.

1.5.1.5 Einstellungen des Import-Verhaltens

Beim Import-Vorgang wird immer geprüft, ob ein Attribut bereits vorhanden ist. Das „Identifizieren“ schließt von Attributen auf die konkreten Objekte. Wenn wir nun im Folgenden von „bereits vorhandenen Attributen“ sprechen, dann sind das Attribute, die im Wert genau mit dem Wert aus der Spalte, auf die sie abgebildet sind, übereinstimmen. Wenn wir von bereits vorhandenen Objekten sprechen, dann sind das konkrete Objekte, die durch ein bereits vorhandenes Attribut identifiziert werden.

Beispiel: Wenn in unserem Netz bereits ein Song mit dem Namen „Eleanor Rigby“ existiert, dann ist das Namensattribut (abgebildet auf die Spalte „Titelname“ unserer Importtabelle) ein existierendes Attribut und folglich der Song ein existierender Song, solange der Song nur

über das Namensattribut identifiziert wird.

Mit den Einstellungen für das Importverhalten können wir steuern, wie der Import auf bereits vorhandene und neue Wissensnetzelemente reagieren soll. Folgende Tabelle zeigt eine Kurzbeschreibung der einzelnen Einstellungen, während die Unterkapitel dieses Kapitels ausführliche und anschauliche Beschreibungen beinhalten.

Einstellung	Kurzbeschreibung
Aktualisieren	Vorhandene Elemente werden überschrieben (aktualisiert), keine neuen Elemente werden angelegt.
Aktualisieren oder neu anlegen wenn nicht vorhanden	Vorhandene Elemente werden überschrieben, sollten keine vorhanden sein, werden sie neu angelegt.
Alle mit selbem Wert löschen (nur bei Eigenschaften verfügbar)	Alle Attributwerte, die mit dem importierten Wert übereinstimmen, werden für die jeweils entsprechenden Objekte gelöscht.
Alle vom selben Typ löschen	Alle Attributwerte des ausgewählten Typs werden für die entsprechenden Objekte gelöscht, unabhängig davon, ob die Werte übereinstimmen oder nicht.
Löschen	Wird verwendet, um genau das eine Element zu löschen.
Neu anlegen	Legt eine neue Eigenschaft/Objekt an, ohne zu beachten, ob der Attributwert oder das Objekt bereits vorhanden ist.
Neu anlegen wenn noch nicht vorhanden (nur bei Attributen verfügbar)	Nur, wenn noch kein Attribut des gewünschten Typs vorhanden ist, wird eines angelegt.
Nicht importieren	Kein Import.
Synchronisieren	Um die zu importierenden Inhalte mit den Inhalten der Datenbank zu synchronisieren, werden bei dieser Aktion alle Elemente, die noch nicht vorhanden sind, neu angelegt, alle die sich geändert haben, aktualisiert und alle, die nicht mehr vorhanden sind, gelöscht.

Bei einem Import müssen wir uns für jedes abgebildete Objekt, jede abgebildete Relation und jedes abgebildete Attribut einzeln entscheiden, welche Import-Einstellung wir jeweils verwenden wollen. Anders als in anderen Editoren des Knowledge-Buildes "vererbt" sich eine Einstellung nicht an die darunterliegenden Abbildungselemente. Auch die Import-Einstellung für ein Objekt "vererbt" sich nicht an seine Attribute.

1.5.1.5.1 Aktualisieren

Wird diese Einstellung bei einem **Attribut** angewendet, sorgt sie dafür, dass der Wert aus der Tabelle den Attributwert genau eines bereits existierenden Attributs überschreibt. Mit dieser Einstellung werden keine neuen Attribute angelegt. Falls das Objekt mehr als einen Attributwert des ausgewählten Typs hat, wird kein Wert importiert.

Verwendet man die Einstellung "Aktualisieren" bei einem identifizierenden Attribut, während man bei dem dazugehörigen Objekt die Einstellung "Aktualisieren oder neu anlegen, wenn nicht vorhanden" verwendet, erscheint die Fehlermeldung "Attribut nicht gefunden", wenn das identifizierende Attribut nicht in i-views vorhanden ist.

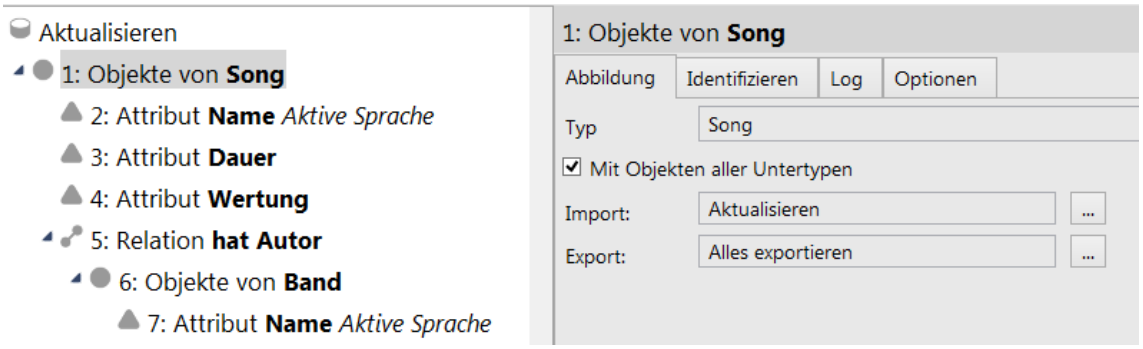
Wird "Aktualisieren" bei einem **Objekt** angewendet, sorgt die Einstellung dafür, dass alle Eigenschaften des Objekts durch den Import hinzugefügt, bzw. geändert werden können. Neue Objekte werden nicht angelegt.

Beispiel: Angenommen wir führen eine Datenbank mit unseren Lieblings-Songs. Nun haben wir eine Liste mit Songs, die neue Informationen, enthalten. Wir wollen diese Informationen in unsere Datenbank bringen, gleichzeitig aber nicht, dass Songs importiert werden, die nicht zu unseren Lieblings-Songs gehören. Hierfür verwenden wir die Einstellung "Aktualisieren".

Der Song "About A Girl" ist bereits im Knowledge-Builder vorhanden.

Song	Dauer	Wertung	Autor
About A Girl	168	5	Nirvana

Die Import-Tabelle enthält Angaben zu Dauer, Wertung und Autor des Songs.



Wir legen für Objekte von Song fest, dass sie aktualisiert werden sollen. Alle Attribute, Relationen und Relationsziele erhalten die Import-Einstellung "Aktualisieren oder neu anlegen wenn noch nicht vorhanden".

Das Ergebnis: Der Song wurde aktualisiert und hat nun neue Attribute und Relationen erhalten. Bereits vorhandene Eigenschaften wurden aktualisiert (Wertung).

1.5.1.5.2 Aktualisieren oder neu anlegen wenn nicht vorhanden

Diese Import-Einstellung wird in den meisten Fällen benötigt und ist darum als Standard-Einstellung gesetzt. Wenn Elemente bereits vorhanden sind, werden sie aktualisiert. Wenn Elemente noch nicht vorhanden sind, werden sie in der Datenbank neu angelegt.

1.5.1.5.3 Alle mit selben Wert löschen

Diese Import-Einstellung ist nur bei Eigenschaften (Relationen und Attributen) verfügbar und wird nur verwendet, wenn über die Import-Einstellung "Löschen" nicht gelöscht werden kann. Mit "Löschen" kann dann nicht gelöscht werden, wenn eine Relation oder ein Attribut bei einem Objekt mehrmals mit denselben Werten vorkommt. Versucht man es dennoch, erscheint eine Fehlermeldung. Zum Beispiel kann es sein, dass der Song "About A Girl" versehentlich zweimal mit der Band "Nirvana" über die Relation "hat Autor" verknüpft wurde.

In solchen Fällen greift die Import-Einstellung "Löschen" nicht, da sie bei Mehrfachvorkommen nicht weiß, welche der Relationen sie löschen soll. Hier muss also "Alle mit selben Wert löschen" verwendet werden.

1.5.1.5.4 Alle vom selben Typ löschen

Diese Import-Einstellung wird verwendet, wenn alle Attribute, Objekte oder Relationen eines Typs gelöscht werden sollen, unabhängig von den vorhandenen Werten. Im Gegensatz dazu berücksichtigen die Einstellungen "Löschen" und "Alle mit selben Wert löschen" die vorhandenen Werte. Es werden nur die Elemente der Objekte gelöscht, die in der Import-Tabelle vorkommen.

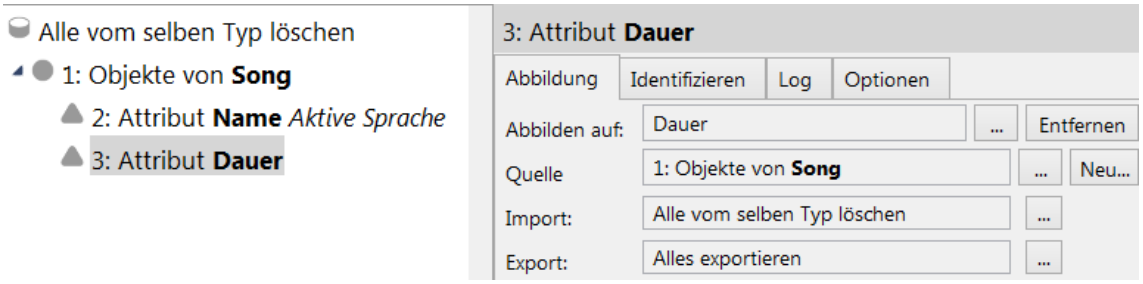
Beispiel: Wir haben eine Import-Tabelle mit Songs und der Dauer der Songs. Wir sehen, dass sich die Dauer in vielen Fällen unterscheidet und beschließen, die Dauer für diese Songs zu löschen, damit wir keinesfalls falsche Angaben haben.

Song	Dauer
19th Nervous Breakdow	120
A Manic Depressive Nan	306
A Place for my Head	239
About A Girl	168

Die Dauer in der Import-Tabelle unterscheidet sich bei den meisten Songs...

Name	Dauer
19th Nervous Breakdown	113
A Manic Depressive Named Laughing Boy	300
A Place for my Head	249
About A Girl	168

... von der Dauer der Songs in der Datenbank.



Beim Attribut "Dauer" verwenden wir die Import-Einstellung "Alle vom selben Typ löschen".

Name	Dauer
19th Nervous Breakdown	
A Manic Depressive Named Laughing Boy	
A Place for my Head	
About A Girl	

Nach dem Import, sind alle Attributwerte des Attributtyps Dauer für diese 4 Songs gelöscht.

1.5.1.5.5 Löschen

Die Import-Einstellung "Löschen" wird verwendet, um genau das eine Objekt / genau die eine Relation / genau den einen Attributwert zu löschen. Falls keine oder mehrere Objekte / Relationen / Attributwerte mit den zu importierenden Elementen übereinstimmen, erscheint diesbezüglich eine Fehlermeldung und die betroffenen Elemente werden nicht gelöscht.

1.5.1.5.6 Neu anlegen

Diese Import-Einstellung legt eine neue Eigenschaft / ein neues Objekt an, ohne zu beachten, ob der Attributwert oder das Objekt bereits vorhanden ist. Einzige Ausnahme: Sollte eine Eigenschaft nur einmal vorkommen (man beachte die Einstellung "kann mehrfach vorkommen" bei der Attributdefinition), so wird das neue Attribut nicht angelegt und eine Fehlermeldung erscheint, die dies mitteilt.

Es sind folgende Fehler aufgetreten:

Zeile	Schema	Wert	Abbildung	Beschreibung	Kategorie
2	Dauer	120	3: Attribut **Dauer**	Attribut "Dauer" kann nicht bei '19th Nervous Breakdown' angelegt werden	Fehler
3	Dauer	306	3: Attribut **Dauer**	Attribut "Dauer" kann nicht bei 'A Manic Depressive Named Laughing Boy' angelegt werden	Fehler
4	Dauer	239	3: Attribut **Dauer**	Attribut "Dauer" kann nicht bei 'A Place for my Head' angelegt werden	Fehler
5	Dauer	168	3: Attribut **Dauer**	Attribut "Dauer" kann nicht bei 'About A Girl' angelegt werden	Fehler

1.5.1.5.7 Neu anlegen wenn noch nicht vorhanden

Diese Import-Einstellung ist nur bei Attributen verfügbar. Ein neuer Attributwert wird nur angelegt, wenn das entsprechende Attribut noch keinen Wert hat. Die Werte müssen nicht gleich sein, es geht nur um das Vorhandensein, bzw. Nichtvorhandensein irgendeines Wertes des entsprechenden Attributtyps. Der Import mehrerer Attributwerte gleichzeitig auf einen Attributtyp ist nicht möglich, da hier nicht entschieden werden kann welcher der Attributwerte verwendet werden soll.

Beispiel: Angenommen, wir haben eine Import-Tabelle, die Musiker mit ihren alias-Namen beinhaltet. Einige Musiker haben auch mehrere alias-Namen. Hier können wir die Einstellung "Neu anlegen wenn noch nicht vorhanden" nicht verwenden, da dann alle Musiker mit

mehreren alias-Namen keinen erhalten würden.

1.5.1.5.8 Nicht importieren

Mit der Import-Einstellung "Nicht importieren" können wir sagen, dass ein Objekt oder eine Eigenschaft nicht importiert werden soll. Dies ist dann nützlich, wenn wir bereits eine Abbildung definiert haben und diese wiederverwenden wollen, jedoch bestimmte Objekte und Eigenschaften nicht noch einmal importieren wollen.

1.5.1.5.9 Synchronisieren

Die Import-Einstellung "Synchronisieren" ist mit Vorsicht zu genießen, denn sie betrifft als einzige Import-Einstellung nicht nur die Objekte und Eigenschaften in i-views, die in ihren Werten mit denen der Import-Tabelle übereinstimmen, sondern darüber hinaus alle Elemente des gleichen Typs in i-views. Wenn man eine Import-Tabelle mit i-views synchronisiert, bedeutet das prinzipiell, dass das Ergebnis in i-views nach dem Import exakt so aussehen soll wie in der Tabelle.

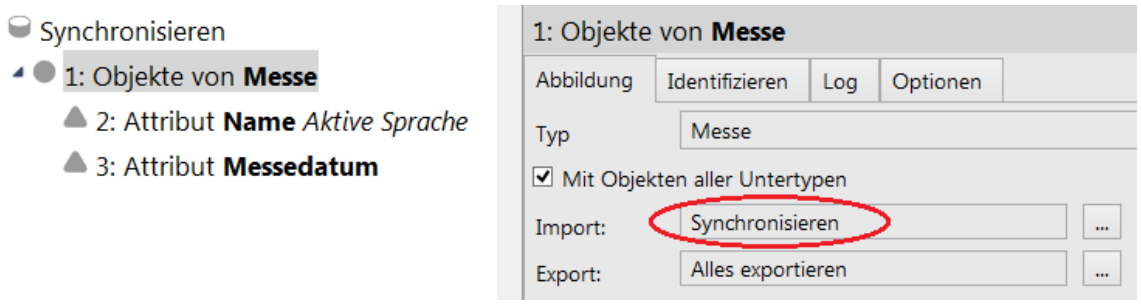
Wenn Objekte eines Typs synchronisiert werden, werden alle Objekte dieses Typs gelöscht, die nicht in der Import-Tabelle vorkommen. Die Objekte, die vorkommen, werden aktualisiert und die Objekte, die nicht in i-views vorkommen, werden neu angelegt.

Beispiel: Wir wollen die Musikmessen in i-views (links) mit einer Tabelle mit den Messen und ihrem Datum (rechts) synchronisieren:

Name	Messedatum
CamJam Europe	26. - 27.09.2015
chor.com Messe	01. - 04.10.2015
Musikmesse 2015	15. - 18.04.2015
Musikmesse 2016	07. - 09.04.2016

Name	Messedatum
CamJam Europe	26. - 27.09.2015
	15. - 18.04.2015
Musikmesse 2016	07. - 10.04.2016

Für die Objekte des Typs Messe wählen wir die Import-Einstellung "Synchronisieren", für die einzelnen Attribute *Name* und *Messedatum* wird die Import-Einstellung "Aktualisieren oder neu anlegen, wenn nicht vorhanden" verwendet:



Das Attribut Name ist das identifizierende Attribut von Messe. Für das Objekt Musikmesse 2015 fehlt der Name in der Import-Tabelle. Importieren wir die Tabelle so, erhalten wir dazu eine Fehlermeldung:

Zeile	Schema	Wert	Abbildung	Beschreibung	Kategorie
4			1: Objekte von **Messe**	Keine identifizierenden Eigenschaften in der Datenquelle vorhanden	Fehler

Nach dem Import sehen wir nun, dass durch den Import zwei Objekte entfallen sind, die keine Entsprechung in der Import-Tabelle hatten. Das Datum bei Musikmesse 2016 wurde

aktualisiert:

Name	Messedatum
CamJam Europe	26. - 27.09.2015
Musikmesse 2016	07. - 10.04.2016

Wenn **Attribute** synchronisiert werden gilt folgendes: Wenn ein bestehendes Attribut durch einen Import keinen Wert erhält, wird es für das entsprechende Objekt der Import-Tabelle gelöscht. Wenn das bestehende Attribut einen anderen Wert hat als in der Import-Tabelle, wird es aktualisiert, auch dann, wenn es mehrmals vorkommen darf. Wenn das Attribut noch nicht vorhanden ist, wird es neu angelegt.

Wenn **Relationen** synchronisiert werden, und sie erhalten keinen Wert werden sie für das entsprechende Objekt gelöscht. Wenn die bestehende Relation einen anderen Wert hat, als in der Import-Tabelle, wird sie aktualisiert. Sollte es das Zielobjekt noch nicht in der Datenbank geben, wird es neu angelegt, vorausgesetzt, das Zielobjekt hat eine entsprechende Import-Einstellung zugewiesen bekommen. Kann das Zielobjekt nicht neu angelegt werden, da hier beispielsweise die Import-Einstellung "Aktualisieren" zugewiesen wurde erscheint eine Fehlermeldung, die uns mitteilt, dass das Zielobjekt nicht gefunden wurde und es wird nicht neu angelegt.

1.5.1.6 Tabellenspalten

Bei Abbildungen von Datenbank-Queries sind die Spalten, die zum Import zur Verfügung stehen, durch die Datenbanktabellen bzw. durch das Select-Statement vorgegeben. Beim Abbilden von Dateien können die Spalten mit der Schaltfläche "Aus Datenquelle lesen" aus der Datei übernommen werden. Man kann sie aber auch von Hand angeben. Dann hat man die Wahl, ob man eine Standard-Spalte oder eine virtuelle Eigenschaft anlegen möchte.

Will man aus der semantischen Graph-Datenbank exportieren, muss man die Spalten von Hand eingeben. Es können nur Standard-Spalten, nicht jedoch virtuelle Spalten exportiert werden.

Virtuelle Tabellenspalte / Virtuelle Eigenschaft
Virtuelle Spalten sind zusätzliche Spalten, die es erlauben die Inhalte, die wir in einer Spalte der zu importierenden Tabelle vorfinden, mit regulären Ausdrücken zu transformieren. Beispiel: Nehmen wir an in unserer Import-Tabelle steht bei den Jahreszahlen immer ein a.d. dahinter. Das können wir bereinigen, indem wir eine virtuelle Spalte anlegen, die aus der Spalte Jahr nur die ersten 4 Zeichen übernimmt.

Auch beim Export können wir virtuelle Eigenschaften definieren.

Den reguläre Ausdruck schreiben wir dabei einfach in die Spaltenüberschrift (in den Namen der Spalte). Dabei werden Teilzeichenketten, die in spitze Klammern <...> eingeschlossen sind, nach den folgenden Regeln ersetzt, wobei *n, n1, n2, ...* für die Inhalte anderer Tabellenspalten stehen.

Ausdruck	Beschreibung	Beispiel	Eingabe	Ausgabe
<n**p**>	Druckausgabe des Inhalts von Spalte n	Treffer: <1p>	1 (integer) 'keine' (String)	Treffer: 1 Treffer: 'keine'

<n**s**>	Ausgabe der Zeichenkette in Spalte n	Hallo <1s>!	'Peter'	Hallo Peter!
<n**u**>	Ausgabe der Zeichenkette in Spalte n in Großbuchstaben	Hallo <1u>!	'Peter'	Hallo PETER!
<n**l**>	Ausgabe der Zeichenkette in Spalte n in Kleinbuchstaben	Hallo <1l>!	'Peter'	Hallo peter!
<n**c**start-stop>	Teilzeichenkette von Position start bis stop aus Spalte n	<1c3-6> <1c3> <1c3->	'Spalten'	alte ten alten
<n**m**regex>	Test, ob der Inhalt von Spalte n den regulären Ausdruck regex matcht. Die folgenden Ausdrücke werden nur ausgewertet, wenn der reguläre Ausdruck zutrifft.	<1m0[0-9]>hi <1m$>test	01 123 (leer) 123	hi (leer) test (leer)
<n**x**regex>	Test, ob der Inhalt von Spalte n den regulären Ausdruck regex matcht. Die folgenden Ausdrücke werden nur ausgewertet, wenn der reguläre Ausdruck **nicht** zutrifft.	<1x0[0-9]>hallo	01 123	(leer) hallo
<n**e**regex>	Selektiert alle Treffer von regex aus dem Inhalt von Spalte n. Einzeltreffer sind im Ergebnis durch Komma voneinander getrennt.	<1eL+> <1e\d\d\d\d>	HELLO WORLD 02.10.2001	LL,L 2001
<n**r**regex>	Entfernt alle Treffer von regex aus dem Inhalt von Spalte n	<1rL>	HELLO WORLD	HEO WORD
<n**g**regex>	Überträgt den Inhalt aller Gruppen des regulären Ausdrucks	<1g\+(\d+)\->	+42-13	42

<n**f**format>	Formatiert Zahlen, Datums- und Zeitangaben aus Spalten gemäß der Formatangabe 'format'	<1f#,0.00> <1fd/m/y> <1fdd/mmm>	3,1412 1234,5 1. Mai 1935 1. Mai 1935	3,14 1.234,50 1/5/1935 01/Mai

Beispiele

Angenommen wir haben eine Import-Tabelle, in der konkrete Objekte ohne Namen vorkommen. In unserem Datenmodell sollen diese Objekte jedoch als eigene Objekte modelliert werden. Ein Beispiel: zu einem Lastpunkt steht in Spalte 88 sein Hauptwert, der Drehmoment. Als Definition unserer virtuellen Spalte, die für den Namen dieses Lastpunktes stehen soll, geben wir also den Ausdruck *Lastpunkt <88s>* ein. Der daraus entstehende Name für einen Lastpunkt mit dem Drehmoment von 850 wäre demnach "Lastpunkt 850".

Wir können die virtuelle Eigenschaft auch nutzen, um einen Usernamen herzustellen, der aus den ersten 4 Buchstaben des Vornamens und des Nachnamens zusammengesetzt ist. Heißt die Person Maximilian Mustermann und wir definieren die virtuelle Spalte mit dem entsprechenden Ausdruck <1c1-4><2c1-4>, erhalten wir das Ergebnis "MaxiMust".

Die virtuelle Eigenschaft kann auch dazu genutzt werden, einem User beim Import ein initiales Passwort anzulegen. Der Ausdruck könnte *Pass4<2s>* lauten. Das daraus resultierende Passwort für Maximilian Mustermann wäre "Pass4Mustermann".

Ein etwas umfangreicheres Beispiel zeigt, wie die virtuelle Eigenschaft dazu genutzt werden kann, Objekten die korrekte direkte Obergruppe zuzuordnen:

Tabelle

#	Gruppe	Nomenklatur mdı	Bezeichnung deu	Ergaenzung_DE	Bezeichnung engl	Ergaenzung_EN	<1mUG><2c1-3>000	<1m><2c1-4>00	Heimtextil 2016
2	HG	010000	floor		floor				Heimtextil 2016
3	UG	010100	Teppiche		Carpets		010000		Heimtextil 2016
4		010101	Handknüpfteppicl		Handwoven carpe			010100	Heimtextil 2016
5		010102	Webteppiche abg		Handwoven and f			010100	Heimtextil 2016
6		010103	Teppiche, handge		Carpets, handtuft			010100	Heimtextil 2016
7		010104	Antike Teppiche		Antique carpets			010100	Heimtextil 2016
8		010105	Sonstige Verfahre		Other processes			010100	Heimtextil 2016
9		010106	Brücken, Vorlager		Rugs			010100	Heimtextil 2016
10		010107	Läufer, Bettumrar		Runners, stair-car			010100	Heimtextil 2016
11		010109	Teppiche		Carpets			010100	Heimtextil 2016
12		010110	Teppichunterlage		Carpet underlays			010100	Heimtextil 2016
13	UG	010200	Teppichböden		Carpeting		010000		Heimtextil 2016
14		010201	Teppichböden, ge		Carpetings, tuftec			010200	Heimtextil 2016

Die drei rechten Spalten sind virtuelle Spalten.

<1mUG>: In die erste der virtuellen Spalten wird die Nummer der Obergruppe des Objekts nur geschrieben, wenn der Begriff "UG" (für Untergruppe) in der ersten Spalte für das Objekt vorkommt.

<2c1-3>000: Die Nummer die in die Spalte geschrieben werden soll, setzt sich aus den ersten drei Zeichen der zweiten Spalte zusammen und drei Nullen.

<1m>: Nur wenn die erste Spalte für das Objekt leer ist, also keinen Wert enthält, wird die Nummer der Obergruppe des Objekts in die Spalte geschrieben.

<2c1-4>00: Die Nummer, die in die Spalte geschrieben werden soll, setzt sich aus den ersten

vier Zeichen der zweiten Spalte zusammen.

Heimtextil 2016: Dieser Ausdruck wird für alle Objekte in die Spalte geschrieben.

1.5.1.7 Konfiguration weiterer tabellenorientierter Datenquellen

Datenbanken

In einem Mapping für eine PostgreSQL-, Oracle- oder ODBC-Schnittstelle müssen die Datenbank, der Benutzer und das Passwort angegeben werden.

Angabe der Datenbank

Die Angabe für die Datenbank setzt sich aus Name des Host, dem Port und dem Namen der Datenbank zusammen. Die Syntax lautet.

```
host:port_databaseName
```

Benutzername und Passwort konfigurieren

Benutzername und Passwort werden so angegeben, wie sie in der Datenbank abgelegt sind. Unter dem Punkt Tabelle kann die Tabelle angegeben werden, die importiert werden soll. Für den Import besteht aber auch die Möglichkeit, dass unter dem Punkt "Query" eine Query formuliert wird, die angibt, welche Daten importiert werden sollen.

Encoding

Handelt es sich um ein PostgreSQL-Mapping, dann kann auf dem Reiter "Encoding" das Encoding angegeben werden.

Spezielle Anforderungen der Oracle-Schnittstelle

Die Funktion zum direkten Import aus einer Oracle Datenbank setzt voraus, dass auf dem importierenden Rechner bestimmte Laufzeit-Bibliotheken installiert sind.

Direkt benötigt wird das "Oracle Call Interface" (OCI) und zwar in einer Version, die laut Oracle zu dem Datenbankserver passt, der angesprochen werden soll. D.h., um eine Oracle 11i Datenbank anzusprechen, sollte auf dem importierenden Rechner das OCI in Version 11 installiert sein. Das OCI lässt sich am einfachsten installieren, wenn man den "Oracle Database Instant Client" installiert. Die Package Version "Basic" ist ausreichend. Der Client ist entweder vom Serverbetreiber zu bekommen oder von Oracle nach Registrierung unter http://www.oracle.com/technology/tech/oci/index.html ladbar.

Nach der Installation ist sicher zu stellen, dass die Bibliothek für den importierenden Client auffindbar ist, entweder indem sie im gleichen Verzeichnis liegt oder für das entsprechende Betriebssystem passende Umgebungsvariablen definiert werden (ist beim OCI dokumentiert).

Je nach Betriebssystem auf dem der Import stattfinden soll, sind weitere Bibliotheken notwendig, die nicht immer installiert sind.

- MS Windows: neben der benötigten "oci.dll" sind noch zwei weitere Bibliotheken notwendig: advapi32.dll (Erweitertes Windows 32 Base-API) und mscvr71.dll (Microsoft C Runtime Library)

Bis auf den XML-Import/Export sind alle Importe/Exporte tabellenorientiert und unterscheiden sich nur in der Konfiguration der Quelle. Für die Beschreibung einer tabellenorientierten Abbildung kann das Beispiel der CSV-Datei herangezogen werden.

1.5.1.8 Abbildung einer XML-Datei

Das Prinzip von XML-Dateien ist, die unterschiedlichen Angaben zu einem Datensatz über Tags (<>) explizit zu machen (nicht über Tabellenspalten). Dementsprechend sind Tags auch die Grundlage der Abbildung beim Import von XML-Strukturen in i-views.

Ein Beispiel: Nehmen wir an, unsere Liste von Songs liegt als XML-Datei vor:

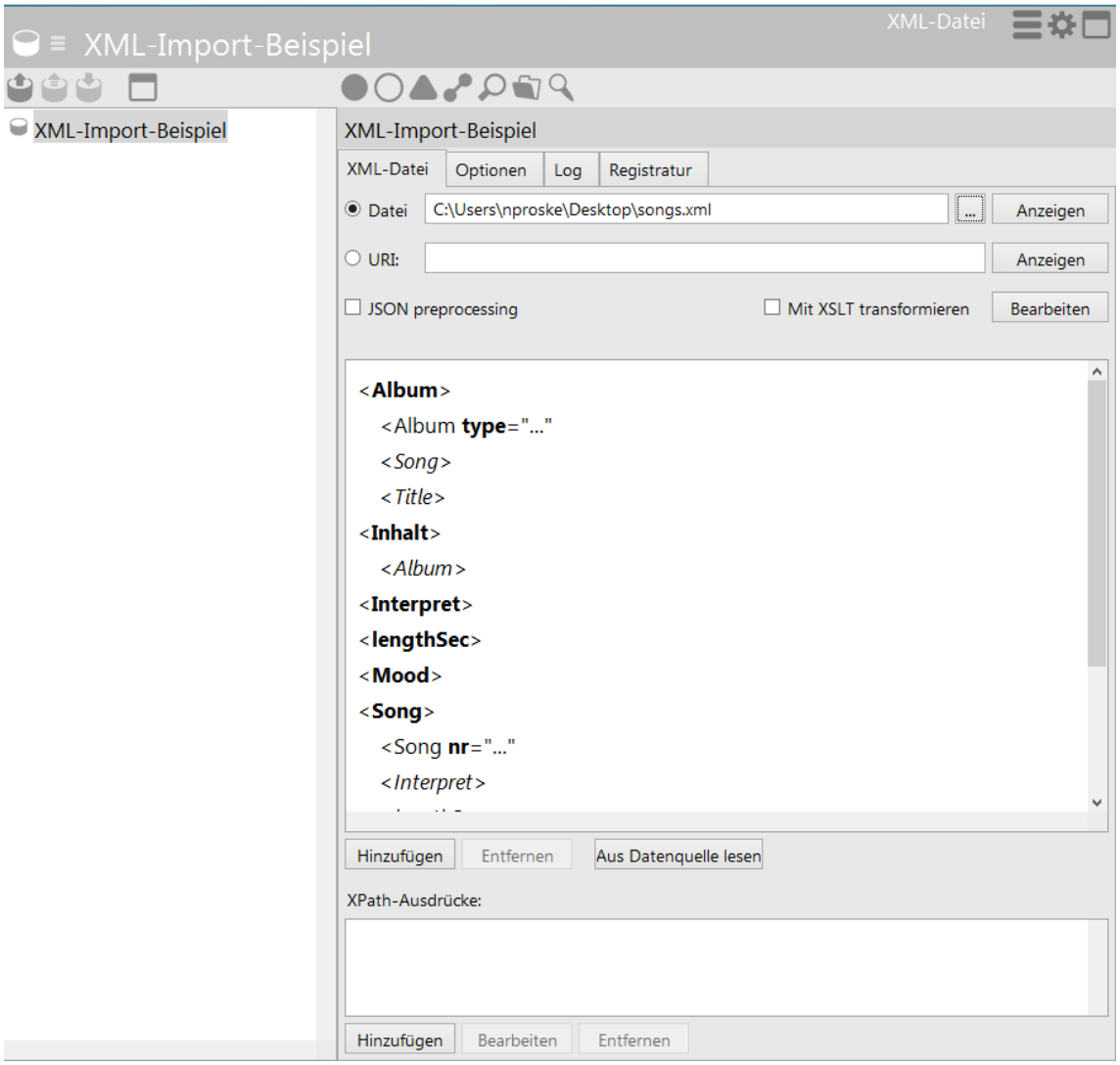
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Inhalt>
    <Album type="Oldie">
        <Title>Revolver</Title>
        <Song nr="1">
            <Title>Eleanor Rigby</Title>
            <lengthSec>127</lengthSec>
            <Interpret>The Beatles</Interpret>
            <Thema>Mental illness</Thema>
            <Mood>Dreamy</Mood>
            <Mood>Reflective</Mood>
        </Song>
        [...]
    </Album>
    [...]
</Inhalt>
```

Wenn wir nun diese XML-Datei importieren wollen, wählen wir bei der Auswahl des Typs der Datenquelle "XML-Datei" aus, wodurch sich der Editor für den Im- und Export von XML-Dateien öffnet. Bereits in der Angabe des Dateistandortes gibt es Unterschiede zum Editor für CSV-Dateien. Wir können nun zwischen einem lokalen Dateipfad und der Angabe einer URI wählen.

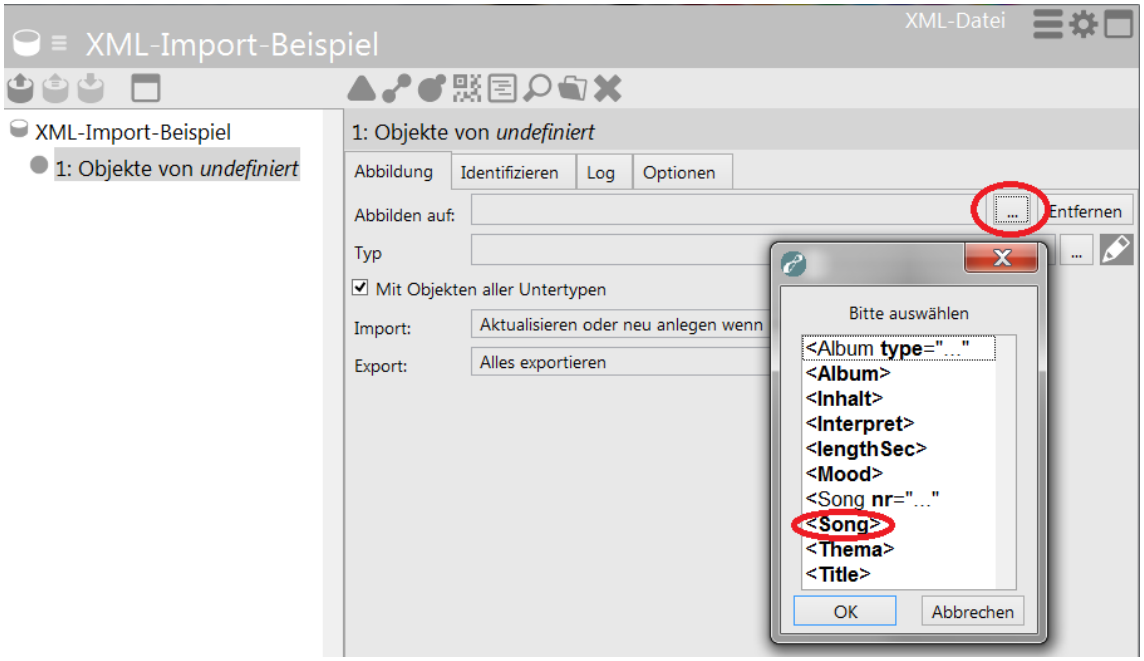
JSON preprocessing ermöglicht das Umwandeln einer JSON-Datei in XML vor dem eigentlichen Import.

Mit XSTL transformieren kann man auswählen, wenn man die XML-Daten aus der ausgewählten XML-Datei noch vor dem Import in andere XML-Daten umwandeln möchte, um beispielsweise die Struktur zu ändern oder einzelne Werte weiter aufzutrennen. Über die Schaltfläche "Bearbeiten" öffnet sich die XML-Datei, in der man die Änderungen mittels XSLT definieren kann.

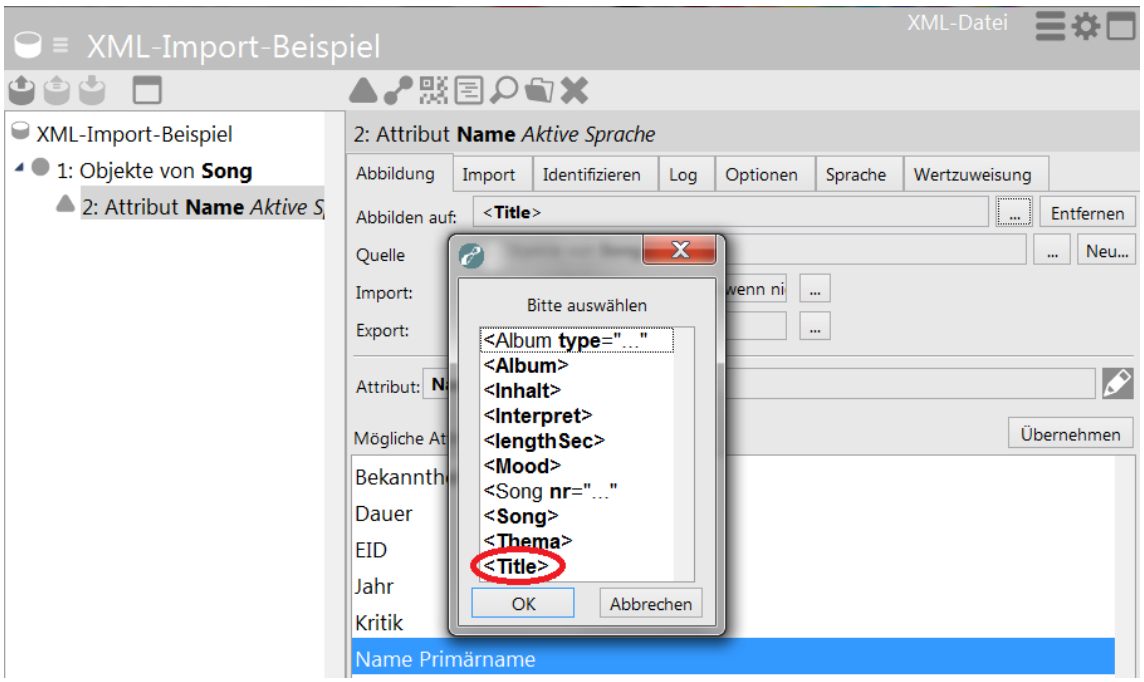
Ist die Datei ausgewählt, können wir mit dem Button "Aus Datenquelle lesen" die XML-Struktur auslesen lassen, die uns daraufhin in rechten Fenster angezeigt wird.



Wir wollen die einzelnen Songs unserer Liste importieren. Darum legen wir eine neue Objektabbildung an und wählen über den Button bei "Abbilden auf" den Tag <*Song*> aus. Im Gegensatz zum CSV-Import, bei dem nur Attributwerte eine Entsprechung in der CSV-Tabelle finden und eine einzelne Zeile für ein Objekt steht, sodass auch nur die Attributwerte abgebildet werden müssen, erfolgt das Mapping der semantischen Objekte hier über die XML-Struktur. Darum müssen auch für alle abzubildenden Objekte jeweils ein entsprechendes Tag der XML-Datei angegeben werden.



Wie in unserem Beispiel, sind die Tags ohne Kontext nicht immer eindeutig: <Title> wird sowohl für Titel von Alben als auch für Songtitel verwendet. Erst in Kombination mit dem umgebenden Tag wird der Objekttyp klar. Oft laufen der Kontext der XML-Struktur und der Kontext der Abbildungshierarchie synchron: Da wir nun bereits festgelegt haben, dass die Objekte auf den Tag <Song> abgebildet werden sollen, ist durch die XML-Struktur klar, welcher <Title>-Tag nun gemeint ist, wenn wir <Title> mit dem Namensattribut von Songs mappen. Dort, wo Abbildungshierarchie und Tagstruktur nicht parallel laufen, können wir im XML-Import zusätzlich zu den in der XML-Datei vorkommenden Tags Ketten bilden - mit XPath.

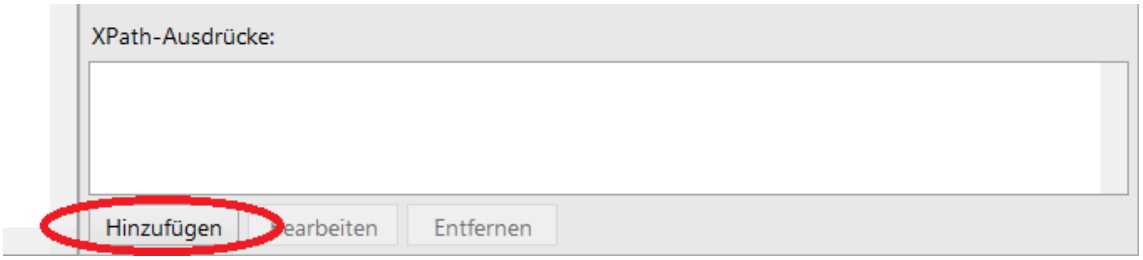


Wie auch beim CSV-Import muss über den Reiter "Identifizieren" bei der Objektabbildung festgelegt werden, durch welche Attributwerte das Objekt in der semantischen Graph-Datenbank

identifiziert werden soll. Das erste angelegte Attribut für ein Objekt wird auch hier wieder automatisch als identifizierendes Attribut verwendet.

Möglichkeiten mit XPath-Ausdrücken

Angenommen wir würden nur Songs aus Alben des Musik-Stils "Oldie" importieren wollen. In unserem XML-Dokument ist die Information über den Musik-Stil direkt im Album-Tag angegeben unter *type="..."*. Im Editor müssen wir also einen XPath-Ausdruck definieren, der den Pfad im XML-Dokument beschreibt, der nur diejenigen Songs enthält, die aus Oldie-Alben stammen. Im rechten unteren Bereich des Editors finden wir ein Feld zum hinzufügen von XPath-Ausdrücken.



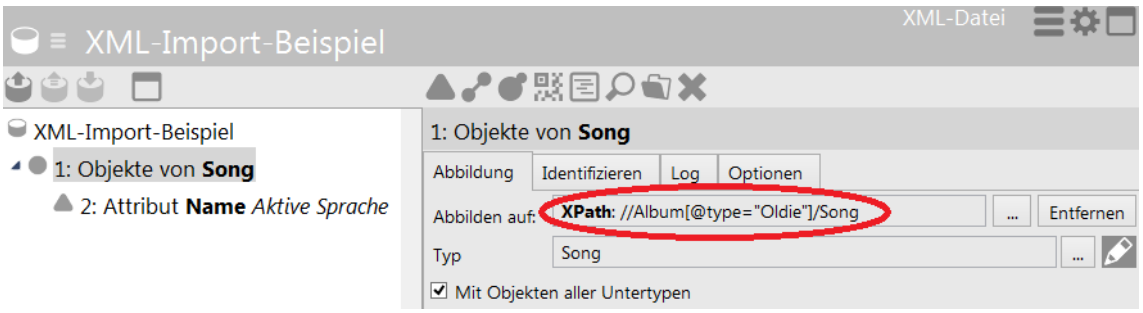
Der passende XPath-Ausdruck lautet:

```
//Album[@type="Oldie"]/Song
```

Erklärung im Einzelnen:

//Album	Selektiert alle Alben, wobei es keine Rolle spielt, wo sie sich im Dokument befinden.
Album[@type="Oldie"]	Selektiert alle Alben vom Typ "Oldie"
Album/Song	Selektiert alle Songs, die Subelemente von Alben sind.

Diesen Ausdruck können wir nun verwenden, um eine Entsprechung für die Objektabbildung der Songs zu definieren.

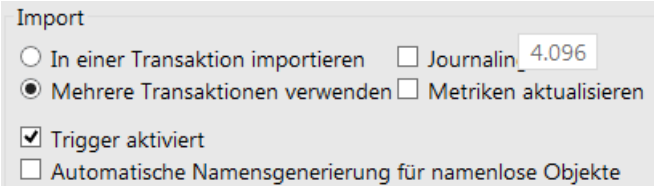


Mit XPath stehen uns außerdem viele weitere nützliche Selektions-Funktionen zur Verfügung. So können wir beispielsweise Elemente über ihre Position im Dokument selektieren, Vergleichsoperatoren einsetzen, sowie alternative Pfade angeben.

1.5.1.9 Weitere Optionen, Log und Registratur

1.5.1.9.1 Weitere Optionen beim Import

Im Reiter "Optionen" stehen uns folgende Funktionen zur Auswahl, die unabhängig von der Datenquelle sind:



In einer Transaktion importieren: Ist langsamer als ein Import mit mehreren Transaktionen und sollte darum nur dann verwendet werden, wenn es beim Import mit mehreren Transaktionen ansonsten zum Konfliktfall kommen kann, wenn viele Personen zur selben Zeit im Knowledge-Builder arbeiten oder wenn man Daten importieren will, bei denen es eine Rolle spielt, dass sie nicht getrennt voneinander betrachtet werden. Beispiel 1: Es erfolgt stündlich ein Import mit dem Status der Auslastung von Maschinen. Die addierten Werte der Auslastung dürfen einen gewissen Wert nicht übersteigen, da es ansonsten eventuell zum Stromausfall kommen kann. Damit diese Regel (z.B. mithilfe eines Skripts) berücksichtigt werden kann, müssen alle Werte gemeinsam betrachtet und dann importiert werden. Beispiel 2: Es erfolgt ein Import mit Personen, von denen höchstens eine den Hauptschlüssel haben kann, weil nur ein Hauptschlüssel existiert. Auch hier muss der Import in einer Transaktion erfolgen, da bei mehreren Transaktionen der Fehler übersehen werden könnte, dass bei zwei Personen das Attribut für den Hauptschlüsselbesitz gesetzt wurde.

Mehrere Transaktionen verwenden: Standardeinstellung für einen schnellen Import.

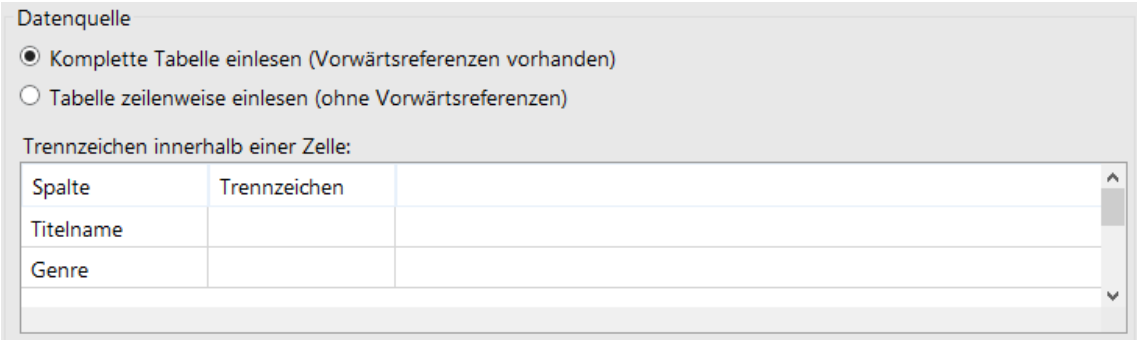
Journaling: Das Journaling sollte verwendet werden, wenn extrem viele Daten mit einem Import gelöscht oder geändert werden. Erst nach jeweils 4.096 Einträgen (die Zahl ist variabel), sollen die Änderungen, bzw. Löschungen für diese Einträge auch am Index vorgenommen werden. Dadurch wird der Import beschleunigt, da nicht für jede einzelne Änderung/Löschung der Index herangezogen werden muss, sondern spätestens nach 4.096 Veränderungen diese gemeinsam in den Index übernommen werden.

Metriken aktualisieren: Die Metriken sollten aktualisiert werden, wenn der Import eine große Auswirkung auf die Menge von Objekttypen oder Eigenschaftstypen hat, wenn also sehr viele Objekte oder Eigenschaften eines Typs der semantischen Graph-Datenbank hinzugefügt werden. Würden die Metriken nicht aktualisiert, könnte dies negative Auswirkungen auf die Performance von Suchen haben, in denen die entsprechenden Typen eine Rolle spielen.

Trigger aktiviert: Ob Trigger beim Import aktiviert sein sollen oder nicht kann hier über das Häkchen bestimmt werden. Falls es gewünscht ist, dass ein Trigger greift und ein anderer nicht, müssen zwei verschiedene Abbildungen mit den entsprechenden semantischen Elementen definiert werden. Informationen zu Triggern stehen im Kapitel Trigger zur Verfügung.

Automatische Namensgenerierung für namenlose Objekte: Ermöglicht die automatische Namensgenerierung für namenlose Objekte.

Liegt eine tabellenorientierte Quelle vor, können wir folgende Einstellungen vornehmen:



Komplette Tabelle einlesen: Obwohl es länger dauern kann, die komplette Tabelle auf einmal einzulesen, macht es Sinn diese Option auszuwählen, wenn Vorwärtsreferenzen vorhanden sind, d.h., wenn Relationen zwischen den zu importierenden Objekten gezogen werden sollen. In diesem Fall müssen nämlich beide Objekte bereits vorhanden sein, was beim zeilenweisen Einlesen der Tabelle nicht der Fall ist. Zudem ist die Fortschrittsanzeige genauer als beim zeilenweisen Einlesen.

Tabelle zeilenweise einlesen: Das zeilenweise Einlesen der Tabelle sollte immer dann verwendet werden, wenn keine Querreferenzen in der Tabelle vorhanden sind, da der Import so schneller geht.

Trennzeichen innerhalb einer Zelle: siehe Kapitel Abbildungen von mehreren Werten für einen Objekttyp bei einem Objekt.

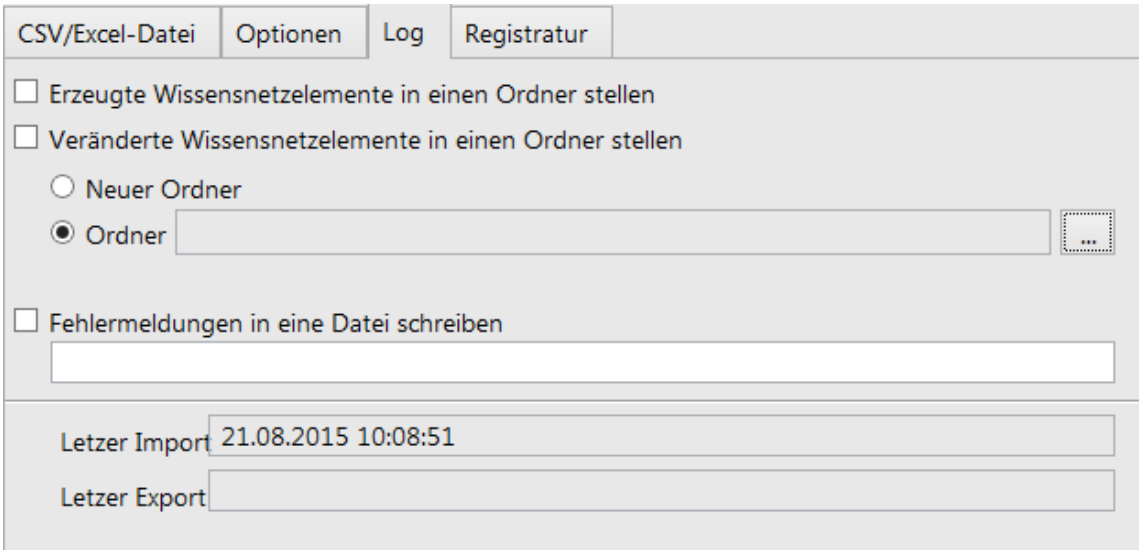
Liegt eine XML-basierte Datenquelle vor, stehen uns folgende Funktionen zur Verfügung:

Inkrementeller XML-Import: Der XML-Import erfolgt schrittweise**.** Die Schritte werden durch das partitionierende Element festgelegt.

DTD einlesen: Liest die Dokumenttypdefinition (DTD) ein**.**

1.5.1.9.2 Log

Die Funktionen im Reiter "Log" ermöglichen Änderungen, die beim Import vorgenommen werden, verfolgen zu können.

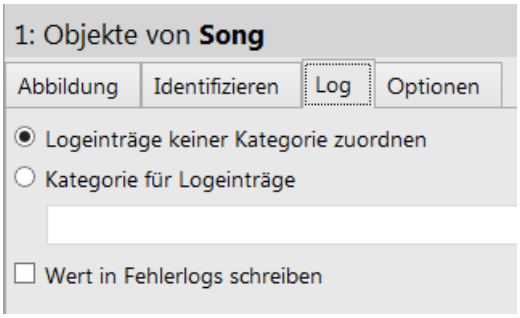


Erzeugte Wissensnetzelemente in einen Ordner stellen: Werden neue Objekte, Typen oder Eigenschaften durch den Import erzeugt, können diese in einen Ordner in der semantischen Graph-Datenbank gestellt werden.

Veränderte Wissensnetzelemente in einen Ordner stellen: Alle Eigenschaften oder Objekte, deren Eigenschaften sich durch den Import geändert haben, können in einen Ordner gestellt werden.

Fehlermeldungen in eine Datei schreiben: Beim Import können Fehler auftreten (z.B. kann es sein, dass ein identifizierendes Attribut für mehrere Objekte vorkam und daher das Objekt nicht eindeutig identifiziert werden konnte). Diese Fehler werden standardmäßig nach einem Import in einem Fenster angezeigt und man hat dann die Möglichkeit den Fehlerbericht zu speichern. Wenn dies automatisch passieren soll, kann hier der Haken gesetzt und eine Datei angegeben werden.

Letzter Import / Letzter Export: Hier werden das Datum und die Uhrzeit des zuletzt vorgenommen Imports und des zuletzt vorgenommen Export angezeigt.



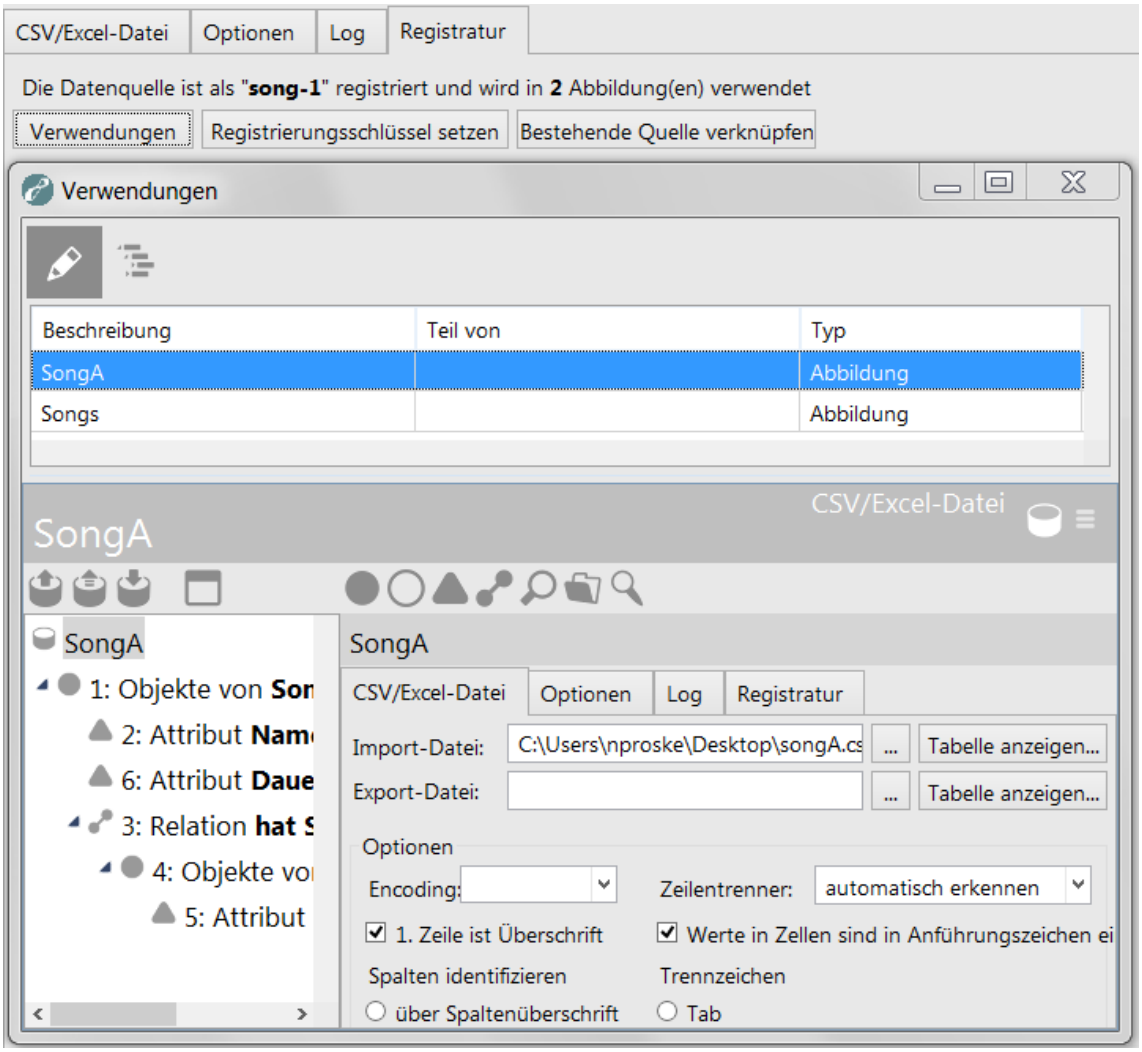
Auch bei den einzelnen Abbildungs-Objekten ist der Reiter "Log" verfügbar. Hier kann bei Bedarf eine Kategorie für Logeinträge eingetragen werden. Zudem kann festgelegt werden, dass der Wert des entsprechenden Objekts / der entsprechenden Eigenschaft in den Fehlerlog geschrieben werden soll. Dies ist standardmäßig nicht aktiviert, um das Offenlegen sensibler Daten (z.B. Passwörter) zu vermeiden.

1.5.1.9.3 Registratur

Unter dem Reiter "Registratur" findet man die Funktion "Registrierungsschlüssel setzen" mit der man die Datenquelle für andere Importe und Exporte registrieren kann.

Die Funktion "Bestehende Quelle verknüpfen" ermöglicht die Wiederverwendung einer registrierten Quelle.

Unter "Verwendungen" kann man einsehen, wo eine Datenquelle noch verwendet wird:



1.5.2 Attributtypen und -formate

Eine häufig auftretende Aufgabe einer Attributabbildung ist der Import bestimmter Daten von konkreten Objekten, beispielsweise von Personen: Telefonnummer, Geburtsdatum etc.

Beim Import von Attributen, für die i-views ein bestimmtes Format verwendet (z.B. Datum), müssen die Einträge der zu importierenden Spalte in einer Form vorliegen, die von i-views unterstützt wird. Beispielsweise kann in ein Attributfeld vom Typ Datum keine Zeichenkette der Form abcde... importiert werden; in einem solchen Fall wird für das entsprechende Objekt kein Wert importiert.

Die folgende Tabelle listet die von i-views unterstützten Formate beim Import von Attributen auf. Ein Tabellenwert ja oder 1 wird also beispielsweise korrekt als boolescher Attributwert (bei einem entsprechend definierten Attribut) importiert, ein Wert wie ein oder ähnliches hingegen nicht.

Attribut	Unterstützte Werte-Formate
Auswahl	Die Abbildung der Import- auf die Attributwerte kann über den Reiter "Wertzuweisung" konfiguriert werden.
Boolesch	Die Abbildung der Import- auf die Attributwerte kann über den Reiter "Wertzuweisung" konfiguriert werden.
Datei	Import nicht möglich.
Datum	• <day> <monthName> <year>, z. B. 5 April 1982, 5-APR-1982 • <monthName> <day> <year>, z. B. April 5, 1982 • <monthNumber> <day> <year>, z. B. 4/5/1982 Das Trennzeichen zwischen <day>, <monthName> und <year> kann z.B. ein Leerzeichen, ein Komma oder ein Bindestrich sein (es sind aber noch weitere Zeichen möglich). Gültige Monatsnamen sind: • 'Januar', 'Februar', 'März', 'April', 'Mai', 'Juni', 'Juli', 'August', 'September', 'Oktober', 'November', 'Dezember' • 'Jan', 'Feb', 'Mrz', 'Mär', 'Apr', 'Mai', 'Jun', 'Jul', 'Aug', 'Sep', 'Okt', 'Nov', 'Dez'. Achtung: Zweistellige Jahreszahlen xy werden zu 20xy expandiert (aus 4/5/82 wird also 4/5/2082). Wenn das Mapping auf "Frei definierbares Format" eingestellt ist, können folgende Tokens verwendet werden: YYYY und YY (Jahr), MM und M (Monatsnummer), MMMM (Monatsname), MMM (abgekürzter Monatsname), DD und D (Tag)
Datum und Uhrzeit	Für Datum und Uhrzeit siehe die jeweiligen Attribute. Das Datum muss vor der Uhrzeit stehen. Wenn die Uhrzeit weggelassen wird, wird 0:00 verwendet.
Farbe	Import nicht möglich.
Festkommazahl	Import möglich.
Ganzzahl	• Ganzzahlen beliebiger Größe • Fließkommazahlen (mit Punkt getrennt), z.B. 1.82. Die Zahlen werden beim Import gerundet.
Internet-Verknüpfung	Jede beliebige URL möglich.
Uhrzeit	<hour>: <minute>: <second> <am/pm>, z.B. 8:23 pm (wird zu 20:23:00) <minute>, <second> und <am/pm> können weggelassen werden. Wenn das Mapping auf "Frei definiertes Format" eingestellt ist, könne folgende Tokens verwendet werden: hh und h (Stunde), mm und m (Minute), ss und s (Sekunde), mmm (Millisekunde)

Zeichenkette	Jede beliebige Zeichenkette. Es wird keine Dekodierung vorgenommen.

Boolesche Attribute und Auswahlattribute

Auswahl- oder boolesche Attribute können nur Werte aus einer vorgegebenen Menge annehmen; bei Auswahlattributen ist dies eine vorgegebene Liste, bei booleschen Attributen das Wertepaar ja/nein in Form eines Klickfelds. Beim Import dieser Attribute kann angegeben werden, wie die Werte aus der Import-Tabelle in Attributwerte der semantischen Graph-Datenbank übersetzt werden. Zum einen können die Werte so, wie sie in der Tabelle stehen, übernommen werden; entsprechen sie keiner der in der semantischen Graph-Datenbank definierten möglichen Werte des Attributs, werden sie nicht importiert. Zum anderen können Wertzuweisungen zwischen Tabellenwerten und Attributwerten, die dann importiert werden, festgelegt werden.

1.5.3 Konfiguration des Exports

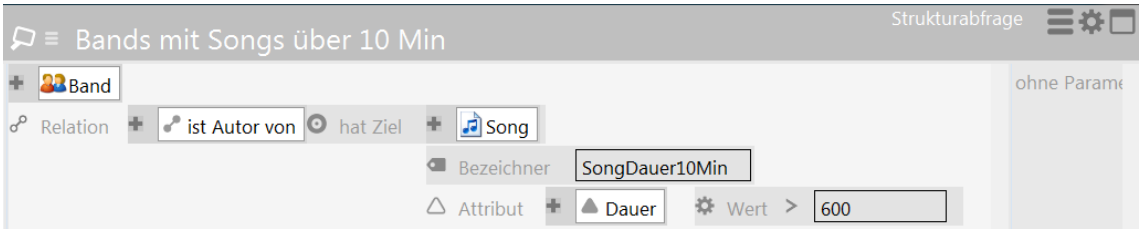
Der Export von Daten aus einer semantischen Graph-Datenbank in eine Tabelle wird in demselben Editor wie der Import und ganz analog vorbereitet:

1. Im einem Tabellen-Mapping-Ordner im Hauptfenster wird ein neues Mapping angelegt.
2. Im Tabellen-Mapping-Editor wird die zu erzeugende Datei angegeben.

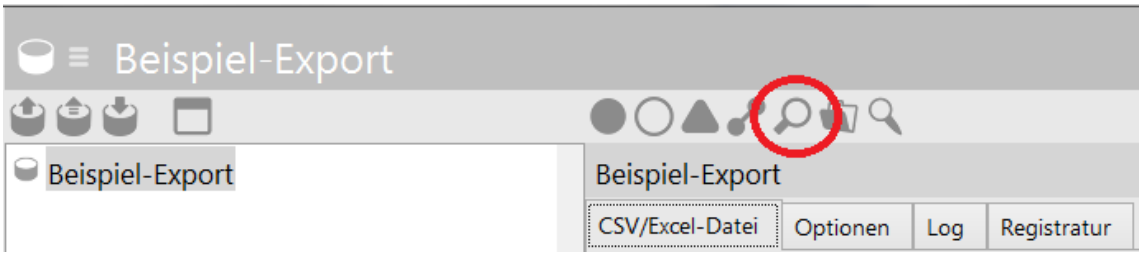
Der Unterschied zum Import liegt darin, dass die Spalten jetzt nicht aus der Tabelle eingelesen werden, sondern im Tabellen-Mapping-Editor angelegt werden müssen. Da der Import- und der Export-Editor derselbe sind, muss man beim Anlegen einer neuen Spalte zunächst auswählen, ob es sich um eine *Standard*-Spalte oder eine *Virtuelle Eigenschaft* handelt. Virtuelle Eigenschaften sind bei einem Export jedoch nicht verwendbar.

Export von Strukturabfragen

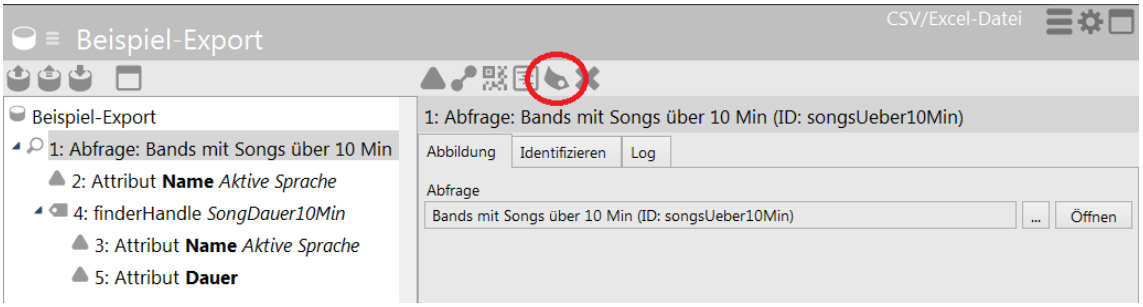
Es besteht die Möglichkeit das Ergebnis einer Strukturabfrage zu exportieren. Diese Vorgehensweise bietet sich an, wenn nur bestimmte Objekte, die durch eine Suche eingeschränkt werden, exportiert werden sollen. Nehmen wir als Beispiel an, wir wollten alle Bands, die Songs geschrieben haben, die länger als 10 min. dauern, exportieren. Dafür müssen wir zunächst eine Strukturabfrage definieren, die die gewünschten Objekte zusammenstellt.



Auf diese Strukturabfrage greifen wir dann von der Konfiguration des Exports aus zu. Dazu wählen wir im Kopf der Mapping-Konfiguration anstelle einer Objekt-Abbildung die Abbildung einer Abfrage. Die Strukturabfrage benötigt einen Registrierungsschlüssel, um auf sie zugreifen zu können.

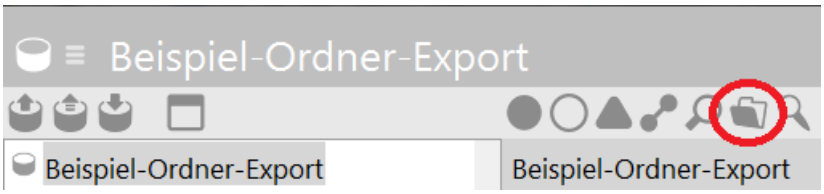


Damit werden nur noch die Ergebnisse der Strukturabfrage exportiert. Für diese Objekte können wir jetzt wieder Eigenschaften angeben, die in den Export mit aufgenommen werden sollen: z.B. Gründungsjahr der Band, Mitglieder und Songs. Jetzt kann es aber vorkommen, das wir von den Bands, die wir so zusammengestellt haben, nicht alle Songs exportieren wollen, sondern gerade nur die, die auch dem Suchkriterium entsprechen, in unserem Beispiel die Songs über 10 min. Dazu können wir die einzelnen Suchbedingungen in der Strukturabfrage mit Bezeichnern belegen. Diese Bezeichner können dann wiederum in der Export-Definition angesprochen werden.



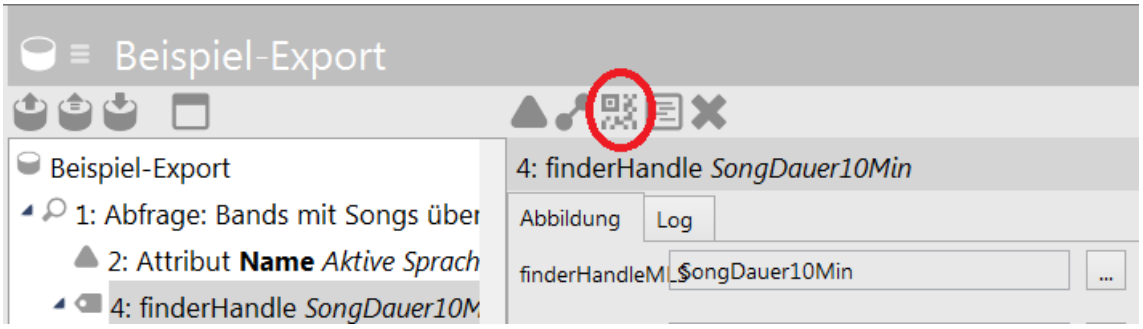
Export von Sammlungen semantischer Objekte

Auch Sammlungen semantischer Objekte können exportiert werden. Diese brauchen ebenfalls einen Registrierungsschlüssel, den man unter TECHNIK -> Strukturordner setzen kann.



Export der Frame-ID

Die Abbildung der Frame-ID ermöglicht es uns, die in der semantischen Graph-Datenbank für ein Wissensnetzelement vergebene ID, zu exportieren. Hierzu wählen wir einfach das Objekt, den Typ oder die Eigenschaft aus, für die wir die ID brauchen und wählen dann den Button "Neue Abbildung der Frame-ID":



Wir können außerdem entscheiden, ob wir die ID im Format eines Strings wollen (ID123_456) oder, ob wir sie als 64 Bit Integer ausgegeben haben wollen.

Export mithilfe von Skripten

Schließlich steht uns beim Export noch ein weiteres mächtiges Werkzeug zur Verfügung: die Skriptabbildung. Informationen hierzu sind im Kapitel Die Skriptabbildung verfügbar.

Export-Aktionen bei Datenbankexporten

Die Abbildung der Eigenschaften eines Objekts wird für einen Export in eine Datenbank genauso vorgenommen wie für einen Import und wie für alle anderen Mappings. Einzig ist für den Export die Export-Aktion zu bestimmen. Diese gibt an, welche Art von Query in der Datenbank ausgeführt werden soll. Es stehen drei Export-Aktionen zur Verfügung:

Folgende Aktionen stehen in dem sich öffnenden Auswahldialog zur Verfügung:

- **Datensätze in Tabelle neu anlegen**: Es werden neue Datensätze in der Datenbanktabelle hinzugefügt. Diese Aktion entspricht einem INSERT.
- **Existierende Datensätze aktualisieren**: Die Datensätze werden über eine ID in der Tabelle identifiziert. Sie werden nur überschrieben, wenn der Wert sich geändert hat. Gibt es keinen passenden Datensatz, dann wird ein neuer hinzugefügt. Diese Aktion entspricht einem UPDATE.
- **Tabelleninhalt beim Export überschreiben**: Alle Datensätze werden erst gelöscht und dann neu geschrieben. Diese Aktion entspricht einem DELETE auf der ganzen Tabelle mit folgendem INSERT.

1.5.4 RDF-Import und -Export

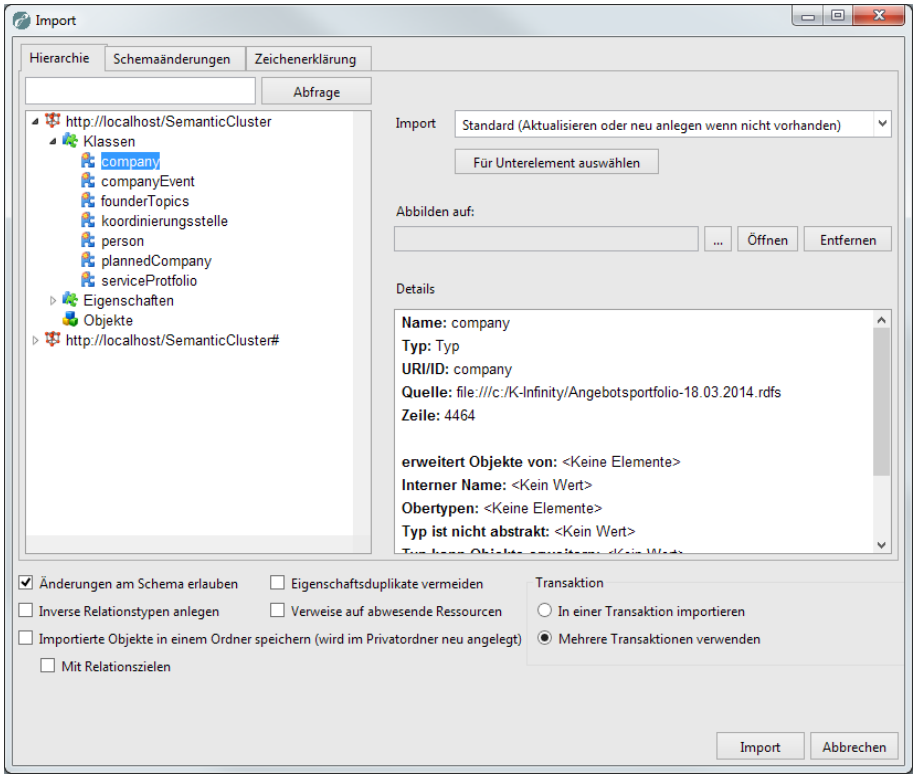
RDF ist ein Standardformat für semantische Datenmodelle. Mit dem RDF-Import und -Export können wir die Daten der semantischen Graph-Datenbank mit anderen Anwendungen austauschen, aber auch Daten von einem i-views-Wissensnetz in ein anderes transportieren.

Beim RDF-Export wird das gesamte Wissensnetz in eine RDF-Datei ausgeleitet. Der RDF-Import dagegen ist interaktiv und selektiv. D.h. wir können sowohl auf Schema-Ebene als auch bei den einzelnen Objekten und Eigenschaften angeben, was importiert werden soll und was nicht.

Abgleich der Objekte aus RDF mit den existierenden Objekten im semantischen Netz

Wenn die RDF-Daten aus demselben Schema stammen, wie das Netz in das sie importiert werden - z.B. aus einer Sicherungskopie - dann ordnet der RDF-Import Objekte und Objekttypen automatisch anhand ihrer ID zu. In den Import-Einstellungen können wir wie bei den Tabellen- und XML-Importen nun bestimmen z.B. existierende Objekte durch den Import aktualisiert werden sollen, ob neue angelegt werden etc.

Wenn die Daten aus einer anderen Quelle kommen, ist die Default-Einstellung der Import in ein eigenes Subnetz. Wir können aber auch diese externen Informationen in unseren Bestand integrieren - durch manuelle Zuordnungen mit der Abbilden-auf-Funktion im Mapping-Interface.



Daneben gibt es einige globale Einstellungen: Möchten wir überhaupt Änderungen am Schema erlauben? Erlauben wir es, dass Eigenschaften mehrfach angelegt werden? Schließlich werden alle Schema-Änderungen auf einem separaten Reiter angezeigt.

1.6 Zugriffsrechte und Trigger

In diesem Abschnitt wird die Prüfung von Zugriffsrechten und Trigger behandelt:

- **Zugriffsrechte** regeln, welche Operationen am semantischen Modell bestimmte Nutzergruppen durchführen dürfen. Sie werden in i-views im Rechtesystem definiert. Das Rechtesystem befindet sich im Bereich *Technik* > *Rechte*.
- **Trigger** sind automatische Operationen, die bei einem bestimmten Ereignis ausgelöst werden und die zugehörigen Aktionen ausführen. Der Bereich Trigger befindet sich unter *Technik* > *Trigger*.

Das Rechtesystem und Trigger sind in einer neu angelegten semantischen Graph-Datenbank initial noch nicht aktiviert. Diese Bereiche müssen erst aktiviert werden, bevor sie eingesetzt werden können.

Bei der Erstellung von Rechten und Triggern ist die grundsätzliche Vorgehensweise identisch: Es werden Filter benötigt, die prüfen ob bestimmte Bedingen erfüllt sind oder nicht. Sind diese Bedingen erfüllt, wird beim Rechtesystem ein Zugriffsrecht oder -verbot erteilt sowie bei Triggern ein Log eingetragen oder ein Script ausgeführt. Im Rechtesystem wird die Anordnung der Filter als Rechtebaum und bei Triggern als Triggerbaum bezeichnet.

1.6.1 Die Prüfung von Zugriffsrechten

Mit Rechten regeln wir den Zugriff von Nutzern auf die Daten im semantischen Netz. Die zwei grundsätzlichen Ziele, deren Erreichung mit dem sogenannten Rechtesystem ermöglicht werden, sind:

- **Schutz von sensiblen Daten:** Es werden Nutzern oder Nutzergruppen, nur die Daten angezeigt, die sie auch lesen dürfen. Damit werden Geheimhaltungs- und Vertraulichkeitsbeschränkungen gewährleistet.
- **Arbeitsspezifische Übersicht:** Bestimmte Nutzer benötigen für ihre Arbeit mit dem System häufig nur einen Ausschnitt der Daten des Modells. Mit Hilfe des Rechtesystems ist es möglich ihnen nur die Elemente anzuzeigen, die sie für das Erledigen ihrer Aufgaben brauchen.

Das Rechtesystem von i-views zeichnet sich durch einen hohen Grad an Flexibilität aus. Es kann auf verschiedene Erfordernisse eines Projektes zielgenau konfiguriert werden. Durch die Definition von Regeln im Rechtebaum bestehend aus einzelnen Filtern und Entscheidern, entsteht ein netzspezifische Konfiguration des Rechtesystems. Es gibt vielfältige Möglichkeiten diese Regeln für das Rechtesystem zusammenzusetzen, wodurch hoch differenzierte Rechte erzeugt werden. Es ist nicht möglich, alle möglichen Kombinationen von Konfigurationen aufzulisten; hier muss eine Beratung für den Einzelfall stattfinden.

Wie funktioniert das Rechtesystem?

Zugriffsrechte im System werden immer dann geprüft, wenn durch einen Nutzer eine Operation auf die Daten vorgenommen wird. Die grundsätzlichen Operationen sind:

- *Lesen*: Ein Element soll angezeigt werden.
- *Modifizieren*: Ein Element soll geändert werden.
- *Erzeugen*: Ein neues Element soll erstellt werden.
- *Löschen*: Ein Element soll gelöscht werden.

Soll in einer bestimmten Zugriffssituation das Zugriffsrecht geprüft werden, wird der **Rechtebaum** abgearbeitet, bis eine Entscheidung für oder gegen den Zugriff in dieser Situation getroffen werden kann. Der Rechtebaum besteht aus Bedingungen, gegen die die Zugriffsituation geprüft wird. Um die Bedingungen zu prüfen, werden **Filter** verwendet, die Elemente des Wissensnetzes und Operationen filtern. Am Ende eines Teilbaumes aus Filtern im Rechtebaum befinden sich die **Entscheider**. Von diesen wird der Zugriff entweder erlaubt oder abgewiesen.

In Bezug auf die Zugriffssituation werden Aspekte ausgewählt, die als Bedingung für die Erlaubnis oder das Verbot des Zugriffes eingesetzt werden. In Zugriffssituationen werden häufig folgende Aspekte für die Entscheidung herangezogen:

- die Operation (Erzeugen, Lesen, Löschen oder Modifizieren)
- das Element, auf das zugegriffen werden soll
- der aktuelle Nutzer

Es kann sein, dass nur ein Aspekt der Zugriffsituation als Bedingung ausgewählt wird, aber es kann auch eine Kombination der aufgeführten Aspekte abgefragt werden. Beispiel: "Paul [Nutzer] darf keine Beschreibungen [Element] löschen [Operation]".

1.6.1.1 Die Aktivierung des Rechtesystems

In einem neu angelegten Wissensnetz ist das Rechtesystem standardmäßig deaktiviert. Damit es genutzt werden kann, muss es in den Einstellungen des Knowledge-Builders aktiviert werden.

Anleitung für die Aktivierung des Rechtesystems

1. Rufen Sie im Knowledge-Builder das Menü *Einstellungen* auf und wählen Sie den Reiter *System* aus. Wählen Sie dort das Feld *Rechte*.
2. Setzten Sie im Feld *Rechtesystem aktiviert* einen Haken.
3. Geben Sie im Feld Benutzertyp den Objekttyp an, dessen Objekte die Benutzer des Rechtesystems sind. Das ist i.d.R. der Objekttyp "Person". (Typ darf nicht abstrakt sein.)
4. Wenn Sie das Knowledge-Portal von i-views angebunden haben, geben Sie in dem Feld *Standard Web-Benutzer* einen Benutzer an (Objekt des zuvor definierten Personenobjekttyps).

Vor der Aktivierung des Rechtesystems heißt der Ordner *Rechte (deaktiviert)*. Wurde das Rechtesystem aktiviert heißt der Ordner *Rechte*. Durch eine Deaktivierung des Rechtesystem, werden keine Prüfungen der Zugriffsrechte mehr durchgeführt. Die definierten Regeln im Rechtebaum bleiben aber erhalten und werden bei einer erneuten Aktivierung des Rechtesystems wieder verwendet.

Beachte: Greift man vom Web-Frontend aus ohne spezielle Anmeldung auf ein Element zu, wird die unter *Standard Web-Benutzer* angegebene Person verwendet. Gewöhnlich legt man hier eine Scheinperson namens "anonymous" oder "Gast" an.

Damit das Rechtesystem auch im Knowledge-Builder funktioniert, muss der Benutzer-Accounts des Knowledge-Builders mit einem Objekt aus dem semantischen Modell verknüpft werden. Der Benutzer-Account kann nur mit Objekten des Typs verknüpft werden, der bei der Aktivierung des Rechtesystems im Feld Benutzertyp angegeben wurde.

Die Verknüpfung ist für die Verwendung des Operationsparameter *Benutzer* bei Suchfiltern bzw. bei für die Verwendung des Zugriffsparameters *Benutzer* bei Strukturabfragen generell notwendig, wenn das Rechtesystem bzw. die Suche nicht in einer Anwendung sondern im Knowledge-Builder selbst ausgeführt.

Anleitung für die Verknüpfung von Knowledge-Builder Nutzern mit Objekten des Personen Typs

1. Im Knowledge-Builder das Menü *Einstellungen* aufrufen und den Reiter *System* wählen. Dort das Feld *Benutzer* auswählen.
2. Den Nutzer auswählen, der verknüpft werden soll. Über *Verknüpfen* kann der Benutzer mit einem Personenobjekt verknüpft werden, das noch mit keinem Knowledge-Builder-Account verknüpft ist.
   Die Funktion *Verknüpfung aufheben* führt dazu, dass die Verknüpfung des Knowledge-Builder-Account mit dem Personenobjekt aufgehoben wird.

Beachte: Der aktuell angemeldete Nutzer kann nicht verknüpft werden.

Benutzer mit Administratorenrechten dürfen generell alle Operationen durchführen, unab-

hängig davon welche Rechte im Rechtesystem definiert wurden. Die Definition als Administrator wird ebenfalls im Menü *Einstellungen* auf dem Reiter *System* im Feld *Benutzer* durchgeführt.

1.6.1.2 Der Rechtebaum

Traversierung des Rechtebaumes

Der Rechtebaum besteht aus Regeln, die in einem Baum definiert sind. Die Äste des Baumes, auch als Teilbaum bezeichnet, bestehen aus den Bedingungen, die geprüft werden sollen. Die Bedingungen werden im System als Filter definiert, die ineinander geschachtelt werden. Bei der Auswertung läuft das System den Baum von oben nach unten ab. Wenn eine Bedingung auf die Zugriffssituation passt, dann geht die Prüfung zum nächsten Filter des Teilbaumes. Dieser Filter wird wiederum geprüft. Dies wird bis zum Ende des Teilbaumes durchgeführt, dort steht ein Zugriffsrecht oder -verbot. Trifft eine Bedingung nicht auf die Zugriffssituation zu, wird zum nächsten Teilbaum gewechselt. Wenn das System bei der Abarbeitung des Rechtebaumes auf ein Zugriffsrecht oder -verbot stößt, wird die Rechteprüfung mit diesem Ergebnis beendet. Die Äste (Teilbäume) des Baumes werden also nacheinander abgearbeitet, der Baum wird "traversiert", bis eine Entscheidung getroffen werden kann.

Filter und Entscheider werden in Form von Ordnern ineinander geschachtelt, so dass ein Baumkonstrukt entsteht, das aus verschiedenen Teilbäumen besteht. Ein Ordner kann mehrere Unterordner haben (mehrere Nachfolgefilter auf einer Ebene), wodurch Verzweigungen im Rechtebaum entstehen. Ordner, die auf einer Ebene definiert sind, werden nacheinander abgearbeitet (von oben nach unten).

Gestaltung des Rechtebaumes

Bei der Erstellung des Rechtebaumes ist es wichtig die Regeln sinnvoll zu gruppieren, denn wenn eine Entscheidung für eine Zugriffserlaubnis oder ein Zugriffsverbot getroffen wurde, werden keine weiteren Regeln mehr geprüft. Deswegen sollten Ausnahmen vor den globalen Regeln definiert werden.

Die zwei grundlegenden Fälle, die man unterscheiden muss, sind:

- **Negativ-Konfiguration**: Im untersten Teilbaum wird pauschal alles erlaubt, darüber werden Verbote formuliert.
- **Positiv-Konfiguration**: Unten ist pauschal alles verboten, außer dem, was weiter oben erlaubt ist.

Die Reihenfolge der Teilbäume ist also ausschlaggebend bei der Erstellung des Rechtebaumes. Die Reihenfolge der Bedingungen in einem Teilbaum dagegen (ob wir zuerst die Operation und dann die Eigenschaft prüfen oder umgekehrt) ist beliebig.

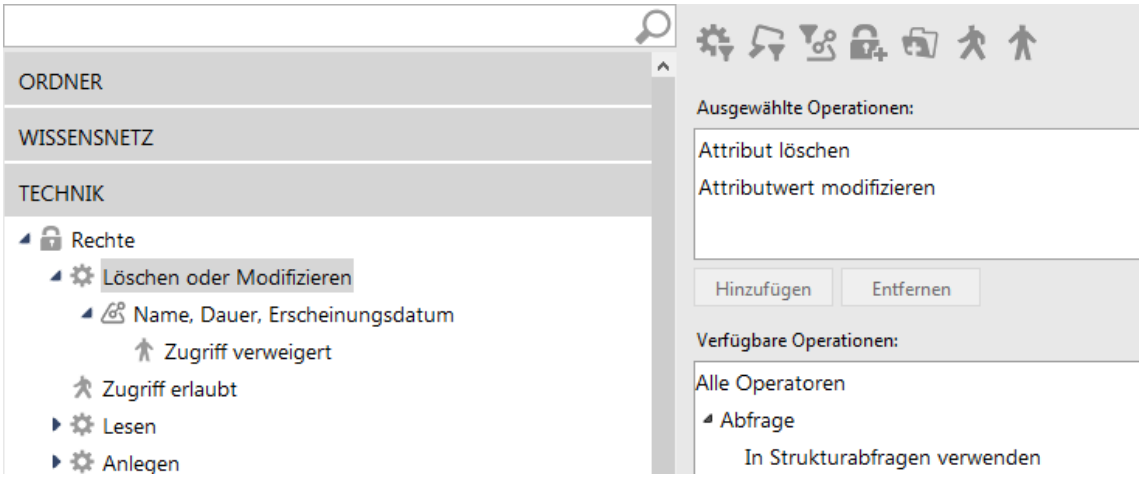
Für die Definition eines Teilbaumes des Rechtebaumes, ist es nicht unbedingt notwendig alle Filterarten zu verwenden. Ein Teilbaum besteht aus mindestens einem Filter und einem Entscheider. Eine Ausnahme ist der letzte Teilbaum der i.d.R. nur aus einem Entscheider besteht, der alle restlichen Operationen erlaubt (die vorher im Rechtebaum nicht verboten wurden) bzw. alle restlichen Operationen verbietet (die vorher im Rechtebaum nicht erlaubt wurden).

Beispiel: Rechtebaum

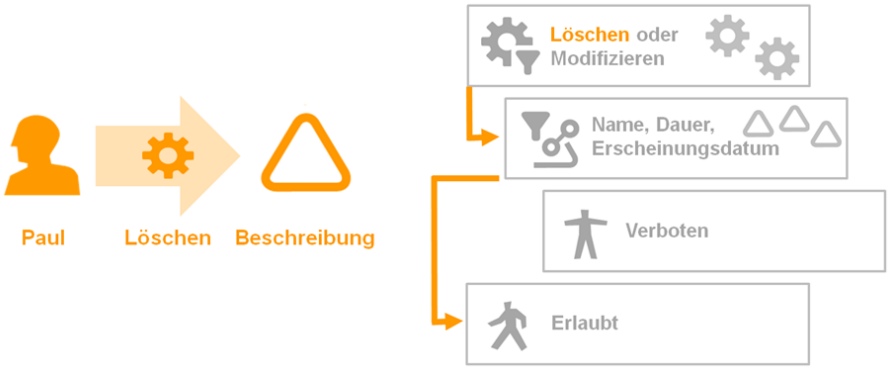
In dieses einfache Beispiel zeigt einen Rechtebaum bestehend aus einem Rechteteilbaum und einen Default-Entscheider, der alles erlaubt:

Im dem Rechteast wird das Löschen oder Modifizieren von den Attributen Name, Dauer und Erscheinungsdatum verboten. Dafür wird ein Operationsfilter verwendet, der die Operationen Löschen oder Modifizieren als Bedingung hat. Nur diese Operationen werden von diesem Operationsfilter durchgelassen. Der nächste Filter ist ein Eigenschaftsfilter, der auf bestimmte Eigenschaften filtert. In diesem Fall werden die Attribute Name, Dauer und Erscheinungsdatum gefiltert unabhängig davon, an welchem Objekt oder an welcher Eigenschaft diese gespeichert sind. Der letzte Knoten des Rechteast ist der Entscheider Verboten, der jede Zugriffsoperation verbietet, die auf die beiden vorgestellten Filter passt. Trifft eine der beiden Bedingungen nicht auf die Zugriffsituation zu, wird der Default Entscheider Erlaubt ausgeführt.

Dieser einfache Rechtebaum würde in i-views folgender Maßen aussehen:



Prüfung einer Operation anhand des Rechtebaum Beispiels:



Die linke Seite zeigt die zu prüfende Operation: Der Nutzer Paul möchte das Attribut Beschreibung löschen. Auf der rechten Seite ist der Rechtebaum abgebildet. Die Prüfung der Bedingung des

ersten Filters fällt positiv aus, da Paul die Operation Löschen durchführen möchte. Im Rechtebaum wird der nächste Filter des Rechteteilbaumes ausgeführt. Dies ist der Eigenschaftsfilter der Attribute Name, Dauer und Erscheinungsdatum. Die Prüfung des Filters fällt negativ aus, da die Beschreibung keine der gefilterten Eigenschaften ist. Die Abarbeitung des Teilbaumes wird angebrochen. Es wird zum nächsten Teilbaum des Rechtebaumes gewechselt. Dies ist bereits der Default-Entscheider Erlaubt, der alles erlaubt, was nicht im Rechtebaum explizit verboten ist.

1.6.1.3 Entscheider im Rechtebaum

Entscheider stehen immer an der letzten Stelle eines Rechteteilbaumes. Durch die Kombination mit Filtern werden Zugriffsituationen bestimmt in denen der Zugriff explizit erlaubt bzw. verboten ist. Wenn bei der Traversierung des Rechtebaumes ein Entscheider erreicht wird, dann wird mit dieser Entscheidung die Rechteprüfung beantwortet. Die zu prüfende Operation wird dann entweder erlaubt oder abgewiesen. Der Rechtebaum wird dann nicht weiter geprüft.

Symbol	Zugriffsrecht	Beschreibung
	Zugriff gewähren	Der Zugriff wird in der zu prüfenden Zugriffssituation erlaubt.
	Zugriff verweigern	Der Zugriff wird in der zu prüfenden Zugriffssituation nicht erlaubt.

Es gibt grundsätzlich zwei verschiedene Entscheider einen positiven - Zugriff erlaubt und einen negativen - Zugriff verboten.

Anleitung zum Anlegen eines Entscheiders

1. Wählen Sie im Rechtebaum die Stelle aus, an der sie einen Entscheider anlegen wollen.
2. Über die Buttons und werden neue Entscheider als Unterordner des aktuell ausgewählten Ordners angelegt.
3. Geben Sie dem Ordner einen Namen.

1.6.1.4 Zusammensetzen von Rechten

Für die Definition von Rechten werden Filter und Entscheider im Rechtebaum miteinander kombiniert. Im Kapitel Filter werden die verschiedenen Filterarten und deren Einsatzmöglichkeiten dargestellt. Die Entscheider *Zugriff gewähren* oder *Zugriff verweigern* stellen jeweils den letzten Knoten eines Teilbaumes des Entscheidungsbaumes dar. Wird ein Entscheider erreicht, so wird mit dieser Entscheidung die Traversierung des Rechtebaumes beendet.

Um Regeln im Rechtesystem zu definieren stehen die folgenden Funktionen zur Verfügung:

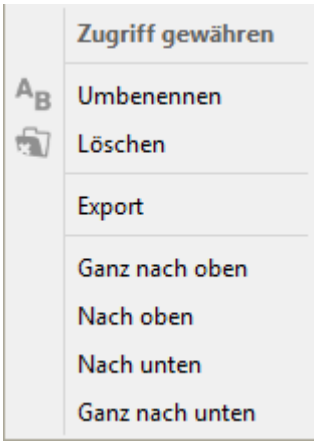
Symbol	Funktion	Beschreibung
	Neuer Operationsfilter	Ein neuer Operationsfilter wird erstellt.

	Neuer Suchfilter	Ein neuer Suchfilter wird erstellt.
	Neuer Eigenschaftsfilter	Ein neuer Eigenschaftsfilter wird erstellt.
	Neuer Strukturordner	Ein neuer Strukturordner wird erstellt.
	Zugriff gewähren	Ein positiver Entscheider, der den Zugriff erlaubt, wird erstellt.
	Zugriff verweigern	Ein negativer Entscheider, der den Zugriff verbietet, wird erstellt.

Um Rechte sinnvoll zu strukturieren, können Strukturordner verwendet werden. Sie haben keinen Einfluss auf die Traversierung des Rechtebaumes. Sie dienen lediglich dazu bei einer Vielzahl von Rechten, inhaltlich zusammengehörige Teilbäume des Rechtebaumes zu gruppieren.

Anordnung von Ordner im Rechtebaum ändern

Um die Filter und Entscheider im Rechtebaum in die richtige Reihenfolge zu bringen, kann über ein Klick mit der rechten Maustaste ein Kontextmenü aufgerufen werden:



In diesem Kontextmenü kann der Filter oder Entscheider umbenannt, gelöscht und exportiert sowie die Position im Rechtebaum verändert werden. Liegen zwei Ordner (Filter oder Entscheider) auf der gleichen Ebene, kann mithilfe der Funktion *Nach oben*, *Nach unten* der Ordner im Rechtebaum weiter nach vorne oder hinten verschoben werden. *Ganz nach oben* und *Ganz nach unten* verschiebt den Ordner entsprechend an die erste bzw. letzte Stelle der Ebene im Rechtebaum.

Sollen Ordner ineinander geschachtelt werden, also die Ebene im Entscheidungsbaum verändert werden, kann dies mit Drag & Drop durchgeführt werden.

Zusammensetzen von Rechten

Durch das Zusammensetzen von Filtern und Entscheidern im Rechtebaum gibt es eine Vielzahl von Kombinationsmöglichkeiten um Rechte zu definieren. Es gibt grundsätzlich 3 verschiedene Vorgehensweisen um Rechte zu definieren:

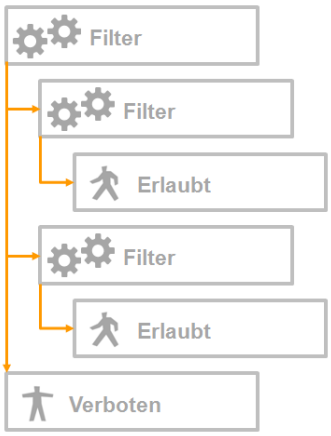
- Definition von Rechten für jede mögliche Zugriffsituation

- Positiv-Konfiguration
- Negativ-Konfiguration

Da die Definition von Zugriffsrechten für jede mögliche Zugriffsituation eine sehr aufwendige Vorgehensweise ist, wird i.d.R. eine der beiden anderen Konfigurationswiesen angewendet. Diese werden in den beiden folgenden Abschnitten erläutert.

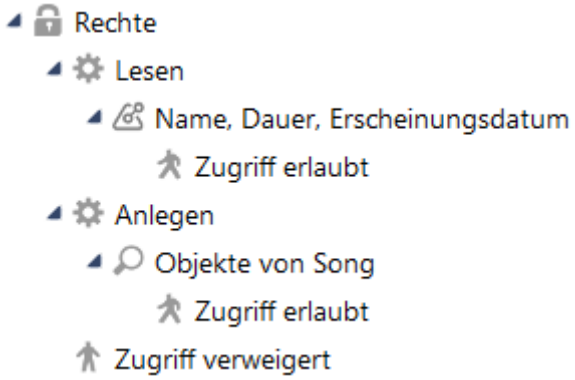
1.6.1.4.1 Positiv-Konfiguration von Rechten

Wenn im Rechtebaum nur Rechte definiert werden, die bestimmte Zugriffe erlauben und alle anderen Zugriffe, über die nichts ausgesagt wird, verboten sind, spricht man von einer Positiv-Konfiguration des Rechtebaumes. In jedem Teilbaum des Rechtebaumes werden Regeln definiert, die bestimmte Operationen erlauben. Alle zu prüfenden Operationen durchlaufen den Rechtebaum: Passt die zu prüfende Operation nicht auf die Bedingungen der Teilbäume, wird sie am Ende des Rechtebaumes abgelehnt.



Beispiel: Positiv-Konfiguration

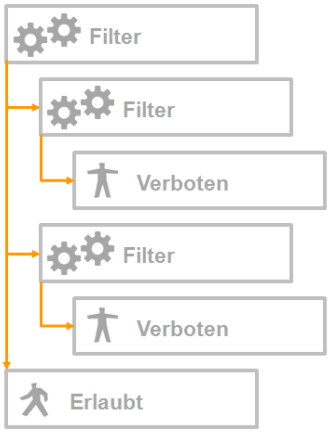
Dieses Beispiel zeigt, wie ein positiv formulierter Rechtebaum im Knowledge-Builder aussehen kann:



Der erste Teilrechtebaum definiert den lesenden Zugriff auf die Attribute Name, Dauer und Erscheinungsdatum. Die Operation Lesen wird für diese Attribute erlaubt. Der zweite Teilrechtebaum erlaubt das Anlegen von neuen Objekten des Typs Song. Alle anderen Operationen werden am Ende des Rechtebaumes generell verboten.

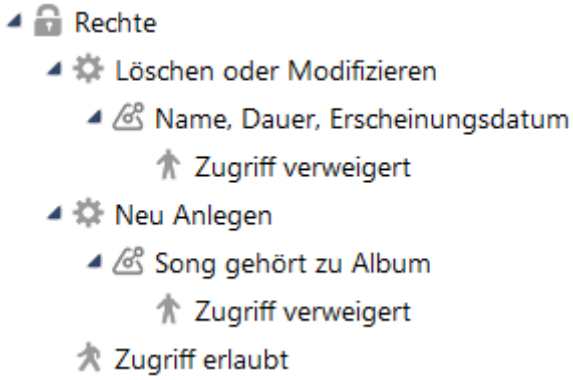
1.6.1.4.2 Negativ-Konfiguration von Rechten

Werden im Rechtebaum Regeln definiert, die bestimmte Operationen ablehnen und alle nicht darauf passenden zu prüfenden Operationen erlaubt werden, spricht man von einer Negativ-Konfiguration. In den Teilbäumen des Rechtebaumes werden bestimmte Operationen verboten. Passt eine zu prüfende Operation nicht auf die Bedingungen der Teilbäume, dann wird die Operation am Ende des Rechtebaumes erlaubt.



Beispiel: Negativ-Konfiguration

Dieses Beispiel zeigt, wie ein negativ formulierter Rechtebaum im Knowledge-Builder aussehen kann:

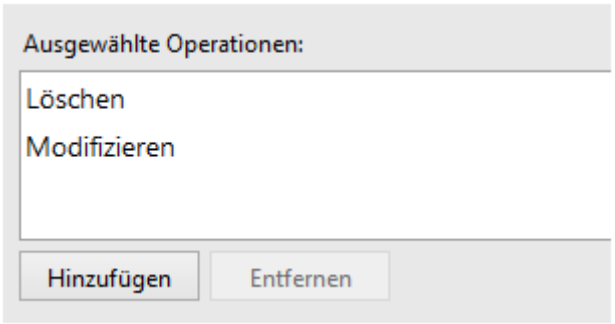


Der erste Teilrechtebaum verweigert im Gegensatz zum Beispiel Positiv-Konfiguration die Zugriffsrechte für das Löschen und Modifizieren der Attribute Name, Dauer und Erscheinungsdatum. Der Zweite Teilrechtebaum verbietet das Löschen der Relation die Songs mit dem Album verbindet, in dem sie enthalten sind. Alle anderen Operationen dürfen durchgeführt werden.

1.6.1.4.3 Beispiel: Jeder Benutzer darf selbst erstellte Elemente ändern und löschen

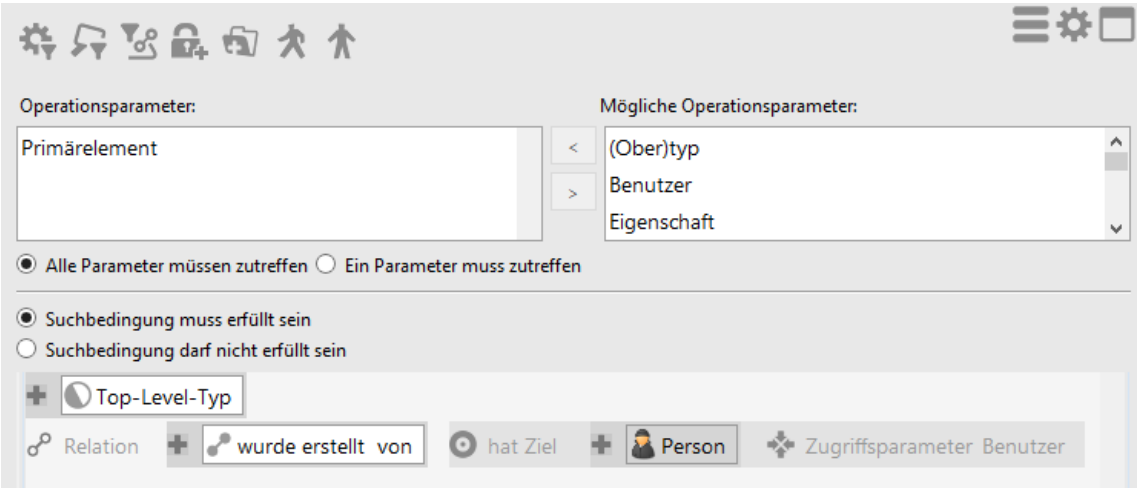
Was wird gebraucht um dieses Recht in i-views zu definieren? Zum einen wird ein Operationsfilter benötigt, da es um das Ändern und Löschen von Elementen geht. Zum anderen muss der Zusammenhang zwischen dem Benutzer und dem Element, an dem er eine Operation ausführen möchte, formuliert werden - das geht nur mithilfe von Suchfiltern.

Operationsfilter



Im Operationsfilter wurden die Operationen Löschen und Modifizieren ausgewählt.

Suchfilter



Im Suchfilter wird die Relation wurde erstellt von mit dem Relationsziel Person ausgewählt. An dem Relationsziel Person wurde der Zugriffsparameter Benutzer angegeben. Die Einstellung Alle Parameter müssen zutreffen und Suchbedingung muss erfüllt sein sind ausgewählt. In diesem Fall wurde der Operationsparameter Primärelement ausgewählt.

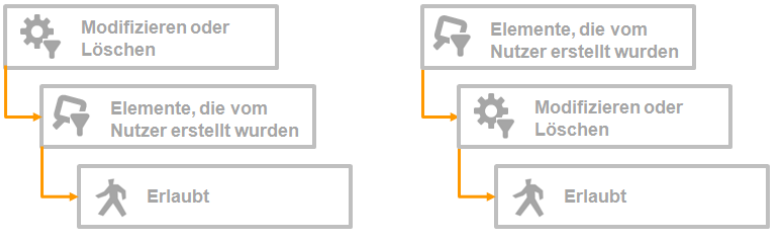
Ein Frage, die das Schema betrifft, ist: An welchen Elementen ist die Relation *wurde erstellt von* definiert? Es gibt verschiedene Möglichkeiten diese Relation in einem semantischen Netz umzusetzen:

1. Fall Definition an Objekten und Typen: Nur an Objekten und Typen wird die Relation verwendet.
2. Fall Definition an allen Elementen: An allen Objekten, Typen, Erweiterungen, Attributen und Relationen wird die Relation verwendet.

Im ersten Fall macht es Sinn den Operationsparameter Primärelement oder übergeordnetes Element zu verwenden. Definiert man das Recht mit dem übergeordneten Element, so gilt es für nicht nur für das Objekt an sich sondern auch für alle Eigenschaften, die an Objekten gespeichert sind, welche vom Nutzer erstellt wurden. Verwendet man stattdessen den Operationsparameter Primärelement so gilt das Recht ebenfalls für alle Metaeigenschaften des Objektes.
Im zweiten Fall wird der Operationsparameter Zugriffselement verwendet, da nur die Elemente geändert werden dürfen, an denen die Relation *wurde erstellt von* mit dem entsprechenden Relationsziel, dem Benutzer, vorkommt.

Das Recht im Rechtebaum zusammensetzen

Es gibt zwei verschiedene Varianten die Filter zu kombinieren. Gibt es in dem Rechteteilbaum keine Verzweigungen so ist die Reihenfolge der Teilbäume nicht relevant.

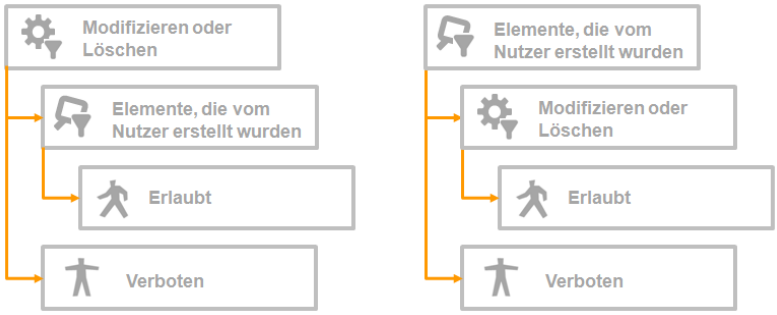


Die Graphik zeigt, die zwei möglichen Kombinationsweisen: Version 1 (links) erst Operationsfilter dann Suchfilter, Version 2 (rechts) erst Suchfilter dann Operationsfilter, als letztes folgt jeweils der Entscheider Erlaubt.

Empfehlung: Es ist sinnvoll den Operationsfilter an erster Stelle zu haben, so ist es möglich unter ihm alle anderen Rechte, welche auf die selbe Operation filtern, anzulegen. Dies schafft eine einfacher nachvollziebare Struktur in den Rechtebaum.

Erweitertes Recht: Elemente die nicht vom Nutzer erstellt wurden, dürfen nicht geändert oder gelöscht werden

Das Recht impliziert das Verbot für alle Elemente, die nicht vom Nutzer erstellt wurden - jedoch haben wir das in der Rechtedefinition noch nicht ausgedrückt. Dafür mussen wir bei der Rechteerstellung den Entscheider Zugriff verboten berücksichtigen. Betrachtet man beide Rechteversionen und kombiniert diese mit dem negativen Entscheider, kommen folgende Varianten heraus. Jedoch haben die beiden Varianten unterschiedliche Auswirkungen im Rechtesystem.



Fügt man an die beiden eben dargestellten Kombinationsweisen jeweils den Entscheider Verboten hinzu, so entstehen die beiden Versionen: Version 1 (links) erst Operationsfilter, dann Suchfilter und Entscheider Erlaubt. Auf den Operationsfilter folgt außerdem in einem zweiten Teilbaum der Entscheider Verboten. Version 2 (rechts) erst Suchfilter, dann Operationsfilter und Entscheider Erlaubt. In dieser Version folgt auf den Suchfilter ein zweiter Teilbaum mit dem Entscheider Verboten.

Auswirkungen der verschiedenen Versionen auf das Rechtesystem

Version 1 (links)

- Erlaubt wird das Modifizieren und Löschen selbst erstellter Elemente.
- Verboten wird das Modifizieren und Löschen aller anderen Elemente.
- Es wird keine Aussage über alle anderen Operationen gemacht.

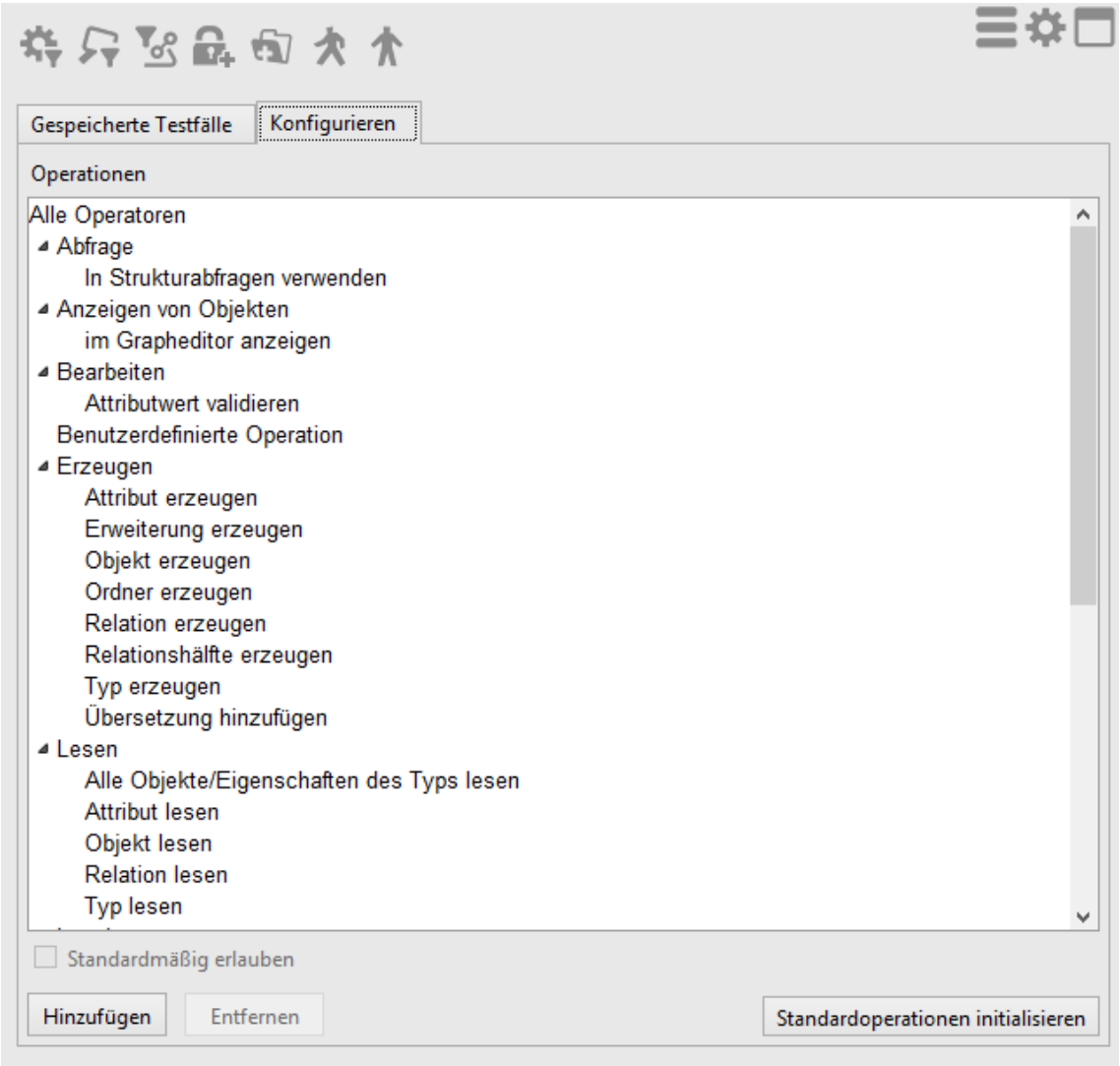
Version 2 (rechts)

- Erlaubt wird das Modifizieren und Löschen selbst erstellter Elemente.
- Verboten werden alle anderen Operationen auf selbst erstellte Elemente (wie z.B. das Lesen)
- Es wird keine Aussage über alle anderen Elemente gemacht.

Die Punkte zeigen, dass Version 2 **nicht** das geforderte Zugriffsrecht ausdrückt. Nur Version 1 formuliert das gewünschte Zugriffsrecht - Jeder Benutzer darf selbst erstellte Elemente ändern oder löschen sowie Elemente, die nicht vom Nutzer erstellt wurden, dürfen nicht geändert oder gelöscht werden.

1.6.1.5 Konfiguration von eigenen Operationen

Wird im Bereich *System* der Ordner *Rechte* ausgewählt, werden im Hauptfenster die Reiter *Gespeicherte Testfälle* und *Konfigurieren* angeboten. Auf dem Reiter *Konfigurieren* können eigene Operationen konfiguriert werden.



Die Konfiguration von eigenen Operationen findet i.d.R. nur dann Anwendung, wenn der Knowledge-Builder zusammen mit anderen Anwendungen verwendet wird. Eigene Operationen sind anwendungsspezifische Operationen, die gemeinsam geprüft werden sollen. Dabei

geht es darum, dass eine Kette von Operationen geprüft werden soll und nicht nur eine Operation.

Anleitung zur Konfiguration von eigenen Operationen

1. Wählen Sie im Knowledge-Builder den Bereich *System* den Ordner *Rechte* aus.
2. Wählen Sie im Hauptfenster den Reiter *Konfigurieren* aus.
3. Klicken Sie auf *Hinzufügen*, damit eine neue Operation erstellt wird.
4. Geben Sie in nachfolgenden Fenstern für die neue Operation einen internen Namen und eine Beschreibung an.
5. Die neue Operation wird als *Benutzerdefinierte Operation* hinzugefügt.
6. Über *Entfernen* können benutzerdefinierte Operationen wieder gelöscht werden.

1.6.2 Trigger

Trigger sind automatische Operationen, die in i-views ausgeführt werden, wenn ein bestimmtes Ereignis eintritt. Sie helfen dabei Arbeitsabläufe zu unterstützen, in dem immer gleich bleibende Arbeitsschritte automatisiert werden.

Beispiele für den Einsatz von Trigger sind:

- Versenden von E-Mails aufgrund einer bestimmten Änderung
- die Bearbeitung von Dokumenten in einer bestimmten Reihenfolge durch bestimmte Personen
- die Kennzeichnung von Aufgaben als offen oder erledigt aufgrund einer bestimmten Bedingung
- die Erstellung von Objekten und Relationen, wenn eine bestimmte Änderung durchgeführt wird
- die Berechnung von Werten in einer vorher definierten Art und Weise
- automatische Generierung des Namensattribut von Objekten (z.B. Zusammensetzung aus Eigenschaften des Objektes)

Wie funktionieren Trigger?

Trigger sind eng verwandt mit dem Rechtesystem. Sie nutzen den selben Filtermechanismus, um festzulegen, wann ein Trigger ausgelöst wird. Die Filter werden in einem Baum angeordnet, dem Triggerbaum, der wie der Rechtebaum aufgebaut ist. Er besteht aus Filtern, mit denen Bedingungen definiert werden, wann eine Trigger-Aktion ausgeführt werden soll. Tritt durch die Durchführung einer Operation eine Zugriffsituation ein, welche auf die definierten Bedingungen passt, wird die zugehörige Trigger-Aktion ausgeführt.

Trigger-Aktionen sind in den meisten Fällen Skripte, die abhängig von den Elementen der Zugriffsituation, mit diesen Operationen durchführen. Somit ist es möglich gleichbleibende Arbeitsschritte zu automatisieren oder intelligente Auswertungen auf Grundlage von bestimmten Konstellationen im sem. Netz durchzuführen. In Skripten können jegliche Operationen auf Elemente, die in Abhängigkeit von komplexen Auswertungen stehen, ausgeführt werden und damit situations- und anwendungsspezifische Anforderungen an das sem. Netz gewährleisten. Die meisten Trigger sind aus diesem Grund i.d.R. projekt- und netzspezifisch; Für den Einzelfall sollte eine Beratung durchgeführt werden.

1.6.2.1 Trigger aktivieren

Um mit Triggern arbeiten zu können, muss die Trigger-Funktionalität zunächst im Knowledge-Builder aktiviert werden.

Anleitung zur Aktivierung von Triggern

1. Rufen Sie die *Einstellungen* des Knowledge-Builders auf.
2. Wählen Sie dort den Reiter *System* und das Feld *Trigger* aus.
3. Setzten Sie im Feld *Trigger aktiviert* einen Haken.

Hier kann ein *Limit für rekursive Trigger* angeben werden. Die Standardeinstellung ist "Keine". Als rekursive Trigger werden Trigger bezeichnet, die sich selber aufrufen. Dies passiert, wenn im Trigger-Skript selbst Operationen im sem. Netz durchgeführt werden, die wiederum selbst auf die Filterdefinition des Triggers passen.

Vor der Aktivierung des Trigger-Funktionalität heißt der Trigger Ordner im Technikbereich von i-views *Trigger (deaktiviert)*. Durch die Aktivierung wird der Ordner in *Trigger* umbenannt.

Anmerkung: Wenn in Triggern der aktuelle Nutzer verwendet wird (z.B. in Suchfiltern oder über die entsprechende Skriptfunktion) und der Nutzer nicht in einer Anwendung Operationen ausführt sondern im Knowledge-Builder selbst, ist die Verknüpfung des Knowldge-Builder-Benutzer-Accounts mit einem Personenobjekt notwendig. Wie eine solche Verknüpfung erstellt wird, wird im Kapitel Aktivierung des Rechtesystems erklärt.

1.6.2.2 Der Triggerbaum

Der Triggerbaum ist wie der Rechtebaum aufgebaut. Er besteht aus Ästen (Teilbäumen), die aus Filtern und Triggern bestehen. Die Filter sind die Bedingungen, die geprüft werden müssen, damit der Trigger am Ende des Teilbaumes ausgeführt werden kann, wenn alle vorher zu prüfenden Bedingungen erfüllt sind.

Der Triggerbaum wird bei jeder Operation auf die Daten abgefragt - der Baum wird "traversiert". Passt ein Teilbaum auf die Zugriffsituation, so wird der Trigger ausgeführt. Passt die Bedingung eines Filters nicht auf die Zugriffssituation, so wird zum nächsten Teilbaum gewechselt. Nach der Ausführung einer Trigger-Aktion wird der Triggerbaum weiter durchlaufen, im Gegensatz zum Rechtesystem, dessen Abarbeitung mit dem Erreichen eines Entscheiders beendet ist. Um im Triggerbaum zu definieren, dass nach der Ausführung einer Aktion keine weiteren Filter geprüft werden sollen, dient die Schaltfläche *Keine weiteren Trigger auslösen*:

Symbol	Funktion	Beschreibung
	Keine weiteren Trigger auslösen	Die Traversierung des Triggerbaumes wird beendet.

Am Ende eines Teilbaumes steht im Gegensatz zum Rechtesystem kein Entscheider sondern Aktionen zur Verfügung.

Symbol	Funktion	Beschreibung

	Trigger definieren	Es wird eine neue Trigger-Aktion erstellt.

Die verfügbaren Trigger-Aktionen sind:

- *Log eintragen*: Ein Logeintrag wird geschrieben.
- *Script ausführen* > *JavaScript*: Eine Script-Datei in JavaScript wird ausgeführt.
- *Script ausführen* > *KScript*: Eine Script-Datei in KScript wird ausgeführt.

Gestaltung des Triggerbaumes

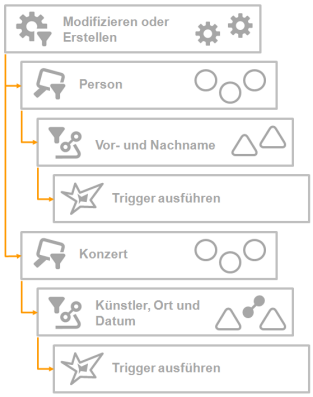
Bei der Gestaltung des Triggerbaumes hat die Reihenfolge, in der man die Trigger definiert i.d.R. keinen Einfluss auf die Performance von i-views. Beim Rechtebaum gibt es Empfehlung zur Gestaltung, die aber nicht auf den Triggerbaum übertragbar sind, da nach Ausführung einer Trigger-Aktion der Triggerbaum weiter traversiert wird.

Für die übersichtlichere Gestaltung der Trigger können diese in Strukturordnern gesammelt werden. Die Strukturordner selbst haben keinen Einfluss auf die Traversierung des Triggerbaumes.

Symbol	Funktion	Beschreibung
	Strukturordner	Strukturordner für die Gruppierung von Teilbäumen

Beispiel: Triggerbaum

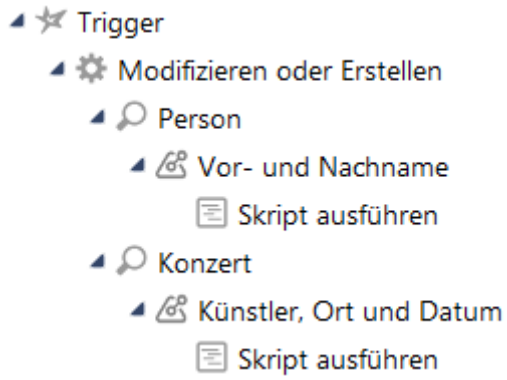
Dieses Beispiel zeigt einen Triggerbaum, der die Namen von Personen und Konzerten automatisch aus Eigenschaften der Objekte zusammensetzt:



Dieser einfach Triggerbaum beginnt mit einem Operationsfilter und teilt sich nach den Operationsfilter in zwei getrennte Teilbäume. Wird einer der beiden Operationen Modifizieren oder Erzeugen ausgeführt, wird diese vom Operationsfilter durchgelassen. Der Teilbaum Person filtert Operationen, die an Attributen, Relationen von Objekten des Typs Person durchgeführt werden. Ist von der Operation entweder das Attribut Vorname oder das Attribut Nachname betroffen, wird diese vom Eigenschaftsfilter durchgelassen. Das dazugehörige Skript, welches das Namensattribut einer Person aus Vor- und Nachname zusammensetzt wird ausgeführt. Der zweite Teilbaum bezieht sich ebenfalls auf den Operationsfilter Modifizieren oder Erstellen aber filtert dann Attribute und

Relationen, die an Objekte des Typs Konzert gespeichert sind. Der Eigenschaftsfilter lässt nur Operationen durch, welche an den Attributen Datum, die Relationen zum Veranstaltungsort oder dem Künstler durchgeführt werden. Treffen diese Bedingungen zu, wird das dazugehörige Skript ausgeführt, welches den Name des Konzertes zusammensetzt.

So würde dieser Trigger Baum in i-views aussehen:



1.6.2.3 Trigger erstellen

Wie im Abschnitt Triggerbaum beschrieben, bestehen Trigger aus Filtern und Trigger-Aktionen. Diese werden miteinander kombiniert, so dass eine bestimmte Trigger-Aktion nur dann ausgeführt wird, wenn sie benötigt wird.

Die folgenden Funktionen stehen im Bereich Trigger zur Verfügung:

Symbol	Funktion	Beschreibung
	Neuer Operationsfilter	Ein neuer Operationsfilter wird erstellt.
	Neuer Suchfilter	Ein neuer Suchfilter wird erstellt.
	Neuer Eigenschaftsfilter	Ein neuer Eigenschaftsfilter wird erstellt.
	Neuer Löschfilter	Ein neuer Löschfilter wird erstellt.
	Neuer Strukturordner	Ein neuer Strukturordner wird erstellt.
	Neuer Trigger	Eine neue Trigger-Aktion wird erstellt.
	Keine weiteren Trigger auslösen	Ein neuer "Stopp"-Ordner wird erstellt. Dieser beendet die Traversierung des Triggerbaumes.

Bei der Erstellung von Triggern sollten zwei grundsätzliche Eigenschaften des Trigger Mechanismus beachtet werden:

- Die Ausführung eines Trigger Skriptes, kann dazu führen, dass weitere Trigger ausgelöst werden. Dies passiert dann wenn im Trigger-Skript selbst Operationen in der semantischen Graph-Datenbank ausgeführt werden.

- Nach der Ausführung einer Trigger-Aktion wird der Triggerbaum weiter durchlaufen. Alle Trigger-Aktionen der Teilbaume, die auf die Zugriffssituation zutreffen, werden ausgeführt.

1.6.2.4 Trigger-Aktionen

Trigger-Aktionen dienen dazu intelligente Operationen in der semantischen Graph-Datenbank durchzuführen, die beispielsweise Arbeitsabläufe automatisieren oder unterstützen. Sie werden nur in bestimmten Situationen ausgeführt, nämlich dann wenn die Zugriffsituation und die Verknüpfungen im semantischen Netz einen bestimmten Zustand annehmen, der durch Filter definiert wird.

Anleitung zum Anlegen von Trigger-Aktionen

1. Wählen Sie im Triggerbaum die Stelle, an der die Trigger-Aktion angelegt werden soll.
2. Fügen Sie über den Button ✨ einen neuen Trigger ein.
3. Wählen Sie aus der Liste den Aktionstyp aus: Log eintragen oder Skript ausführen (Wenn Sie ein Skript ausführen wollen, wählen Sie die Skriptsprache aus.)
4. Der Trigger wird als Unterordner des aktuell ausgewählten Ordners erstellt.

1.6.2.4.1 Skript Trigger

Für die Ausführung des Skriptes muss ein Operationsparameter angegeben werden. Im Gegensatz zu Suchfiltern, kann nur ein Operationsparameter angegeben werden. Auf dem im Operationsparameter enthaltenem Element startet die Ausführung des Skriptes.

Zeitpunkt/Art der Ausführung

- Vor der Änderung: Der Trigger wird ausgeführt bevor die Operation durchgeführt wird.
- Nach der Änderung: Der Trigger wird direkt nach der Durchführung der Operation ausgeführt.
- Ende der Transaktion: Der Trigger wird erst am Ende der gesamten Transaktion ausgeführt.
- Job-Client: Der Jobclient bestimmt den Zeitpunkt der Ausführung.

Beachte: Trigger, die bei Löschoperationen ausgelöst werden, sollten vorzugsweise als Zeitpunkt *Vor der Änderung* verwenden, da ansonsten das zu löschende Element nicht mehr zur Verfügung steht. Für andere Operationen bietet sich als Zeitpunkt eher *Nach der Änderung* oder *Ende der Transaktion* an, da dann beispielsweise eine Eigenschaft zu dem neu erstellten Element hinzugefügt werden kann oder automatisch der Name aus verschiedenen Eigenschaften eines Objektes generiert werden kann, wenn eine oder mehrere Eigenschaften geändert wurden.
Werden z.B. mehrere Datensätze in einem Import in i-views importiert, die eine Trigger-Aktion auslösen, die auf Basis von importierten Relationen, Aktionen im sem. Netz durchführen, kann es sinnvoll sein den Import in einer Transaktion durchzuführen und entsprechend *Ende der Transaktion* als Zeitpunkt der Ausführung auszuwählen, da sonst noch nicht alle Relationen die das Script benötigt importiert wurden.

Je Operationsparameter nur ein mal ausführen

Ist diese Einstellung ausgewählt, dann wird das in Operationsparameter ausgewählte Ele-

ment maximal ein mal pro Transaktion ausgeführt. Wenn diese Einstellung gesetzt ist, sollte der Ausführungszeitpunkt auf *Ende der Transaktion* gesetzt werden, damit im Skript der endgültige Zustand des Elements verwendet wird.

Beispiel: Bei Personen soll der Name des Objekts aus Vorname und Nachname zusammengesetzt werden. Mit dieser Einstellung wird bei gleichzeitiger Änderung von Vor- und Nachname der Trigger nur ein mal ausgeführt.

Ausführung löst keine Trigger aus

Mit dieser Einstellung wird festgelegt, dass durch die Operationen, die innerhalb eines Triggers ausgeführt werden, keine weiteren Trigger ausgelöst werden können. Mit dieser Einstellung lassen sich Endlosschleifen vermeiden.

Bei Skriptfehlern Skript weiter ausführen

Ist diese Einstellung aktiv, so wird versucht nach Ausführungsfehlern wieder aufzusetzen und die Ausführung des Skriptes fortzuführen. Diese Einstellung eignet sich vorwiegend für Skripte, die voneinander unabhängige Anweisungen ausführen sollen, nicht für solche, die auf vorherige Schritte des Skriptes aufbauen.

Transaktion abbrechen, wenn Trigger fehlschlägt

Diese Einstellung legt das Abbruchverhalten bei Skriptfehlern fest. Tritt bei der Ausführung des Skriptes ein Fehler auf und diese Einstellung ist aktiv, werden alle Aktionen der Transaktion rückgängig gemacht. Ist diese Einstellung nicht aktiv, werden alle Aktionen durchgeführt außer diese, die von der Fehlerstelle betroffen sind. Die ursprüngliche Aktion, die zum Aufruf des Triggers geführt hat, wird trotzdem durchgeschrieben.

Ausführen während eines Daten-Refactorings

Unter Daten-Refactoring werden Operationen zur Umstrukturierung des semantischen Netzes verstanden wie z.B. *Typ wechseln* oder *Relationsziel neu wählen*. Daten-Refactoring-Operationen können unter Umständen ungewollte Trigger-Aktionen auslösen und in bestimmten Fällen auch Fehler bei der Durchführung des Skriptes erzeugen. Aus diesem Grund kann pro Trigger eingestellt werden, ob er bei Daten-Refactorings ausgeführt werden soll.

Der Funktionsrumpf für JavaScript-Triggerfunktion wird automatisch angelegt. Ein JavaScript Trigger braucht zwangsläufig den Funktionsrumpf der Triggerfunktion. (Er darf z.B. nicht auskommentiert werden.)

```
/**
 * Perform the trigger
 * @param parameter The chosen parameter, usually a semantic element
 * @param {object} acccess Object that contains all parameters of the access
 * @param {$k.User} user User that triggered the access
**/

function trigger(parameter, access, user)
{
}
```

Das Parameter "access" kann (je Operation variierend) folgende Eigenschaften enthalten:

Eigenschaft	Beschreibung

accessedObject	Zugriffselement
core	Kernobjekt
detail	Detail
inversePrimaryCoreTopic	Primäres Relationsziel
inverseRelation	Inverse Relation
inverseTopic	Relationsziel
operationSymbol	"read", "deleteRelation", etc.
primaryCoreTopic	Primäres Kernobjekt
primaryProperty	Primäreigenschaft
primaryTopic	Primärelement
property	Eigenschaft
topic	Übergeordnetes Element
user	Benutzer (identisch zu "user"-Parameter der Funktion)

1.6.2.4.2 Log Trigger

Möchte man die Trigger-Funktionalität überwachen bzw. dokumentieren, wann welcher Trigger ausgelöst wurde und welche Operationen im sem. Netz ausgeführt wurden, eignen sich Log Trigger. Der Log wird in das jeweilige Log File (bridge.log, importtool.log etc.) geschrieben in dessen Anwendungsumgebung die Operation, welche den Trigger ausgelöst hat, durchgeführt wird.

Zeilen des Logeintrages	Zustand des sem. Netzes zum Zeitpunkt
#pre	vor Auslösung
#post	nach Auslösung
#end	am Ende der Transaktion
#commit	bei erfolgreicher Beendigung der Transaktion

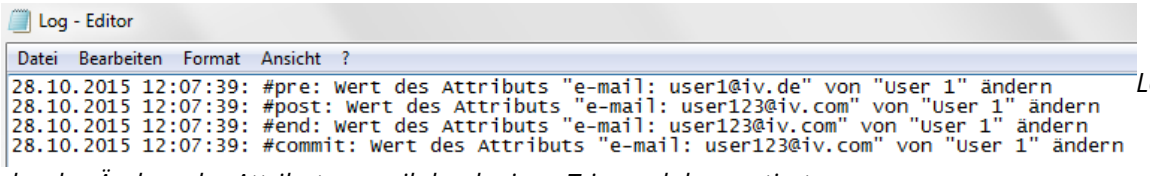
Logeinträge dienen dazu nachzuvollziehen, ob in einer bestimmten Zugriffsituation, die tatsächlich geschehen ist, ein Trigger ausgeführt wurde und was er gemacht hat. Im Gegensatz dazu kann in der Testumgebung getestet werden, ob in einer bestimmten Zugriffsituation ein Trigger ausgelöst werden würde oder nicht, ohne dass die konkrete Zugriffsituation durchgeführt wird.

Anleitung zum Anlegen von Log Triggern

1. Wählen Sie im Triggerbaum das Trigger-Skript aus, welches geloggt werden soll.

2. Erstellen Sie über den Button ein Trigger vom Typ *Log eintragen* im Triggerbaum direkt vor dem Skript-Trigger.

Beispiel:



Logeintrag der das Ändern des Attributs e-mail durch einen Trigger dokumentiert.

1.6.3 Filterarten

Mithilfe von Filtern werden die Bedingungen im Rechtebaum bzw. im Triggerbaum definiert, um Zugriffssituationen einschränken zu können, wann ein Entscheider bzw. Trigger ausgeführt werden soll. Neue Filter werden im Baum unterhalb des aktuell ausgewählten Knotens angelegt. Auf diese Weise werden sie untereinander geschachtelt.

Im Rechtesystem stehen die drei Filterarten Operationsfilter, Suchfilter und Eigenschaftsfilter zur Verfügung. Zusätzlich zu den drei grundsätzlichen Filterarten bietet der Bereich Trigger einen spezifischen Filter - den Löschfilter.

Es gibt verschiedene Arten von Filtern - Wann benutzen wir welchen Filter?

Symbol	Filter	Beschreibung
	Operationsfilter	Filtert die Operationen; Auswahl aus Liste
	Suchfilter	Filtert Elemente durch Strukturabfrage
	Eigenschaftsfilter	Filtert Relationen und Attribute; Auswahl aus Liste
	Löschfilter	Filtert das Löschen von Elementen

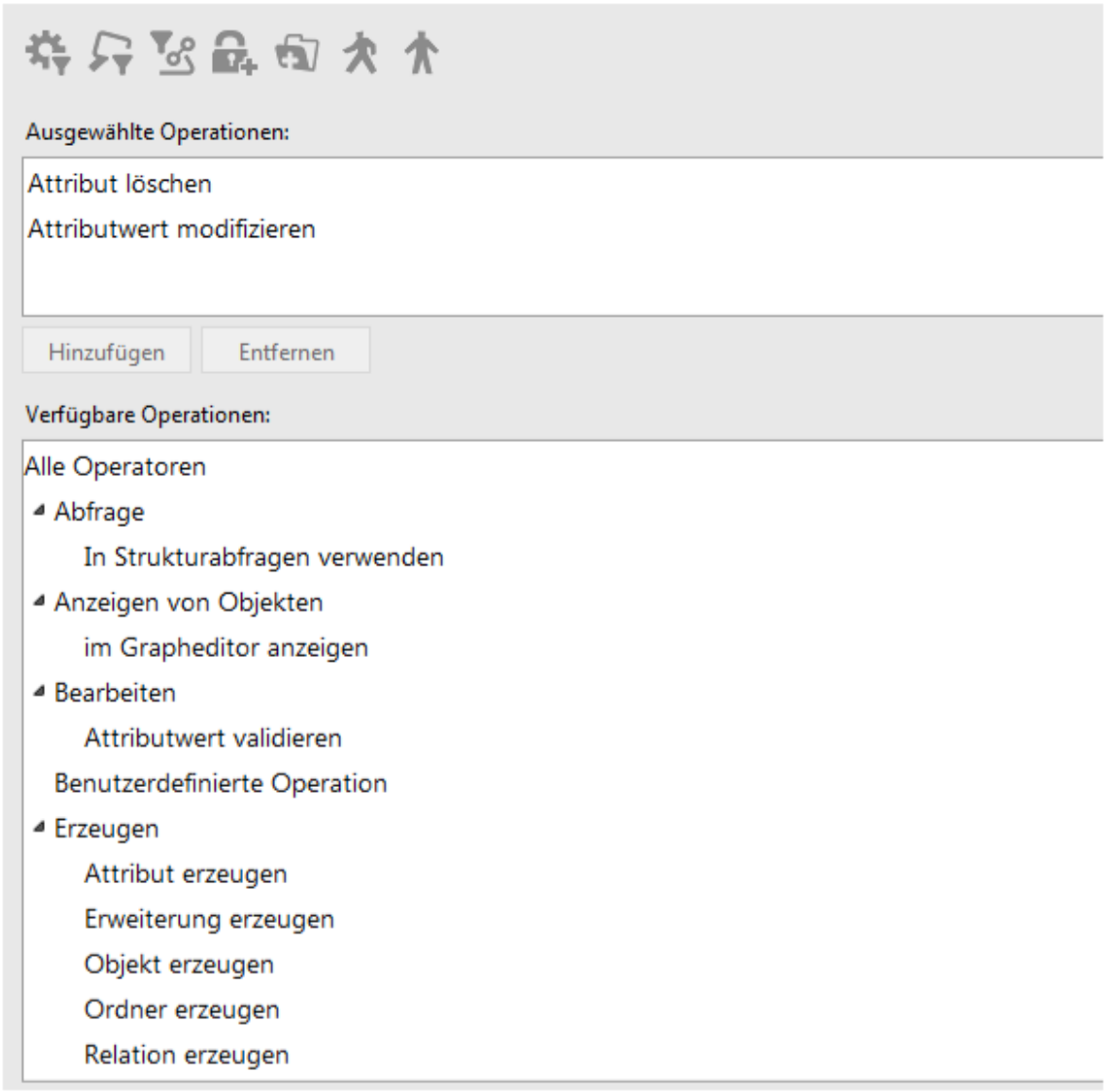
Operationen können nur mit einem Operationsfilter bestimmt werden. Benutzer können nur durch Suchfilter bestimmt werden. Eigenschaften können entweder mit Such- oder Eigenschaftsfiltern bestimmt werden. Die Verwendung von Eigenschaftsfiltern ist dann sinnvoll, wenn unabhängig von weiteren Eigenschaften im semantischen Modell wie Relationen zum Nutzer, Eigenschaften gefiltert werden sollen. Vor allem wenn große Mengen von Eigenschaften gefiltert werden sollen, ist es einfacher und übersichtlicher, das in einer Liste zu tun, anstatt in einer Strukturabfrage. Sollen Relationen zum Zugriffsobjekt oder zum Nutzer einbezogen werden, muss allerdings ein Suchfilter verwendet werden.

Anleitung zum Anlegen eines Filters

1. Wählen Sie im Rechte- bzw. Triggerbaum die Stelle aus, an der Sie einen neuen Filter anlegen wollen.
2. Erstellen Sie über die Buttons , , oder einen neuen Filter.
3. Der Filter wird als Unterordner des aktuell ausgewählten Ordners im Baum angelegt.
4. Geben Sie dem Ordner einen Namen.

1.6.3.1 Operationsfilter

Für welche Operationen ein Zugriffsrecht gelten soll oder ein Trigger ausgeführt werden soll, kann nur mithilfe von Operationsfiltern angegeben werden. Durch die Auswahl der gewünschten Operation kann diese dem Filter hinzugefügt oder wieder entfernt werden.



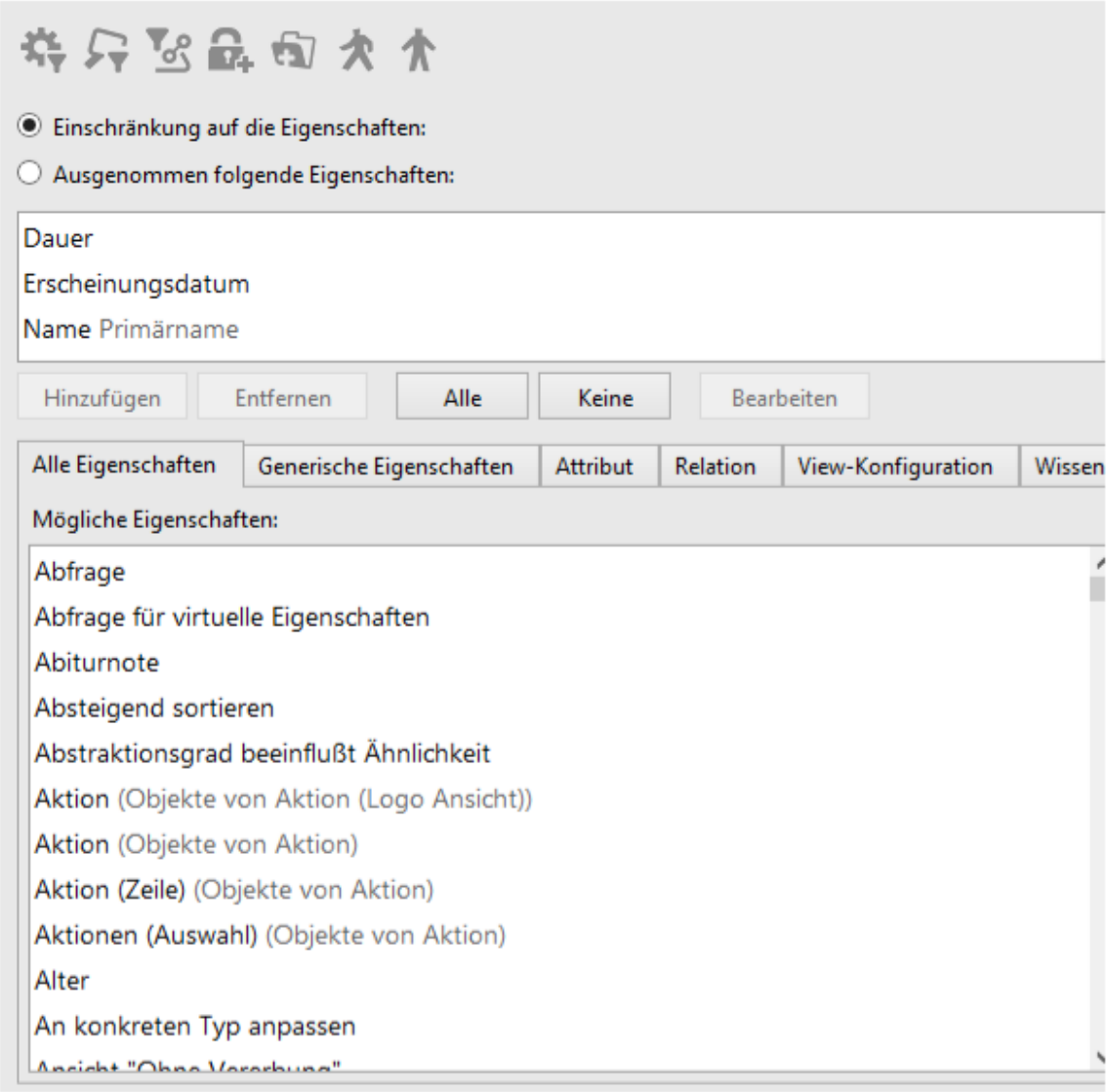
Die Operationen sind in Gruppen gegliedert. Wählt man den übergeordneten Knoten einer Gruppe aus, werden auch alle darunterliegenden Operationen mit gefiltert. Wenn beispielsweise die *Erzeugen* Operation ausgewählt wird, dann werden die Operationen *Attribut erzeugen*, *Erweiterung erzeugen*, *Ordner erzeugen*, *Relation erzeugen*, *Relationshälfte erzeugen*, *Typ erzeugen* und *Übersetzung erzeugen* vom Filter berücksichtigt.

Im Kapitel Operationen werden alle verfügbaren Operationen aufgelistet und zusätzlich wird angegeben, welche Operationsparameter in Kombination verwendet werden können. Die verschiedenen Operationsparameter werden entsprechend im Kapitel Operationsparameter erklärt.

1.6.3.2 Eigenschaftsfilter

Mit Eigenschaftsfiltern können Attribute und Relationen gefiltert werden. Es gibt zwei verschiedene Vorgehensweisen einen Eigenschaftsfilter zu verwenden:

- *Einschränkung auf Eigenschaften*: Angabe der Eigenschaften für die die Bedingung gelten soll. Nachfolgende Filter oder Entscheider des Teilbaumes werden nur ausgeführt, wenn die Zugriffseigenschaft mit den ausgewählten Eigenschaft übereinstimmt.
- *Ausgenommen folgende Eigenschaften*: Angabe der Eigenschaften für die die Bedingung nicht gelten soll. Stimmt die Zugriffseigenschaft mit einer der ausgewählten Eigenschaften überein, werden nachfolgende Filter, Entscheider oder Trigger nicht ausgeführt.



Über *Hinzufügen* und *Entfernen* können die unten aufgeführten Eigenschaften selektiert werden. Alle unten stehenden Eigenschaften können mithilfe von *Alle* ausgewählt werden. *Keine* entfernt alle ausgewählten Eigenschaften. Über das *Bearbeiten* Feld wird der Detaileditor des Attributs oder der Relation aufgerufen, das oder die im oberen Auswahlfeld markiert ist. Die Reiter *Alle Eigenschaften*, *Generische Eigenschaften*, *Attribut, Relation*, *View-Konfiguration* und

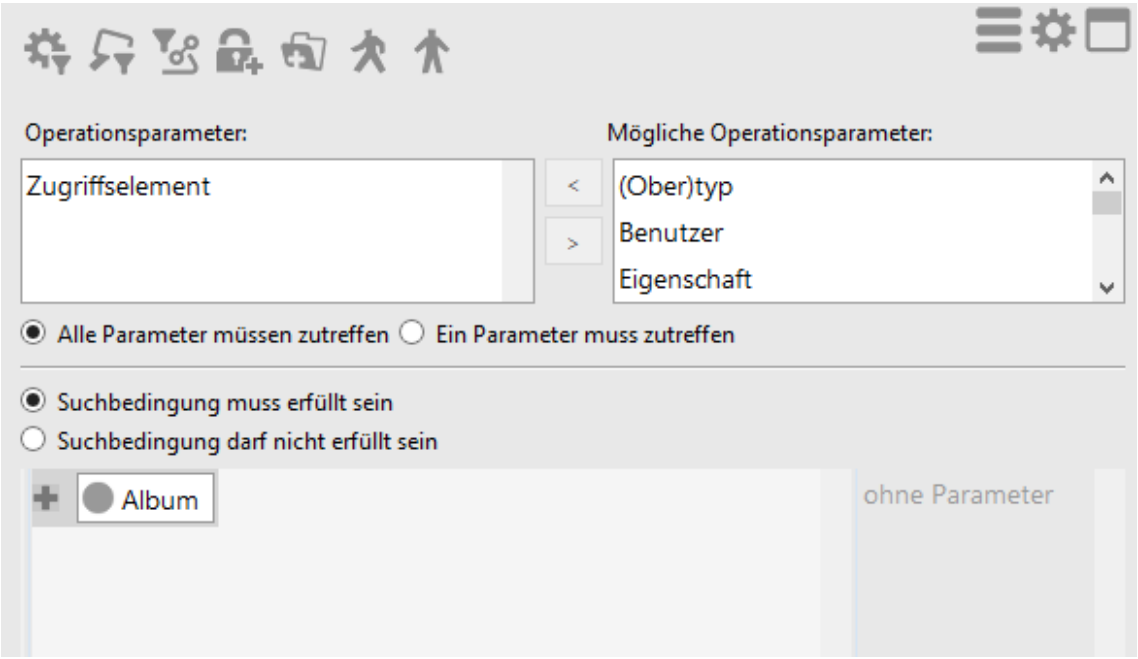
Wissensnetz sollen dem Anwender helfen, die zu filternden Eigenschaften schneller zu finden. Im Reiter *Wissensnetz* werden alle selbst angelegten Relationen und Attribute angezeigt.

1.6.3.3 Suchfilter

Suchfilter ermöglichen es Elemente im Umfeld des Elementes, auf das zugegriffen werden soll, einzubeziehen. So können nicht nur einzelne Eigenschaften sondern auch Zusammenhänge zwischen Objekten, Eigenschaften und Attributen in die Rechte- bzw. Triggerdefinition einbezogen werden. Bei der Verwendung von Suchfiltern muss ein Operationsparameter angegeben werden, mit dem das Ergebnis der Strukturabfrage verglichen wird. Alle verfügbaren Operationsparameter werden im Kapitel Operationsparameter erklärt.

Es gibt zwei verschiedene Vorgehensweise Suchfilter zu definieren:

- *Suchbedingung muss erfüllt sein*: Diese Einstellung ist initial ausgewählt. Stimmt das Suchergebnis der Strukturabfrage mit dem Operationsparameter überein, ist die Bedingung des Filters erfüllt und nachfolgende Filter, Entscheider oder Trigger werden ausgeführt.
- *Suchbedingung darf nicht erfüllt sein*: Liefert die Strukturabfrage als Ergebnis das selbe Element wie der Zugriffsparameter, ist die Bedingung nicht erfüllt und die Prüfung des Rechte- bzw. Triggerbaumes wechselt zum nächsten Teilbaum. Ist das Ergebnis der Strukturabfrage ein anderes als der Zugriffsparameter liefert, ist die Bedingung erfüllt und der nachfolgende Filter, Entscheider oder Trigger wird ausgeführt.



Die Objekte des Typs links oben, die auf die Suchbedingung passen, sind das Ergebnis der Strukturabfrage. Diese werden mit dem Element, das vom Operationsparameter übergeben wird, verglichen. In der Strukturabfrage können Zugriffsparameter verwendet werden, mit diesen können beispielsweise der Benutzer, das Zugriffsobjekt usw. in die Suche einbezogen werden.

Bei der Auswahl der Operationsparameter kann konfiguriert werden, ob

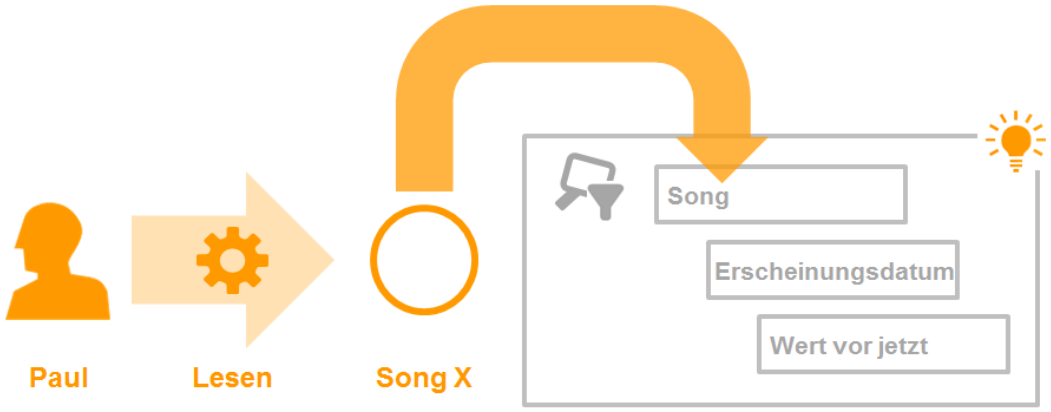
- alle ausgewählten Parameter zutreffen müssen (*Alle Parameter müssen zutreffen*)

- oder nur ein Parameter zutreffen muss (*Ein Parameter muss zutreffen*).

Beachte: Initial ist die Einstellung *Alle Parameter müssen zutreffen* ausgewählt. Werden beispielsweise die Operationsparameter *Zugriffselement* und *Primärelement* ausgewählt, ist die Bedingung nur dann erfüllt, wenn das Ergebnis der Strukturabfrage sowohl Zugriffselement als auch Primärelement der zu prüfenden Operation ist.

Beispiel 1: Suchfilter im Rechtesystem

Es soll ein Recht definiert werden, das besagt, dass bereits veröffentlichte Songs von allen gesehen werden dürfen unveröffentlichte Songs hingegen nicht.

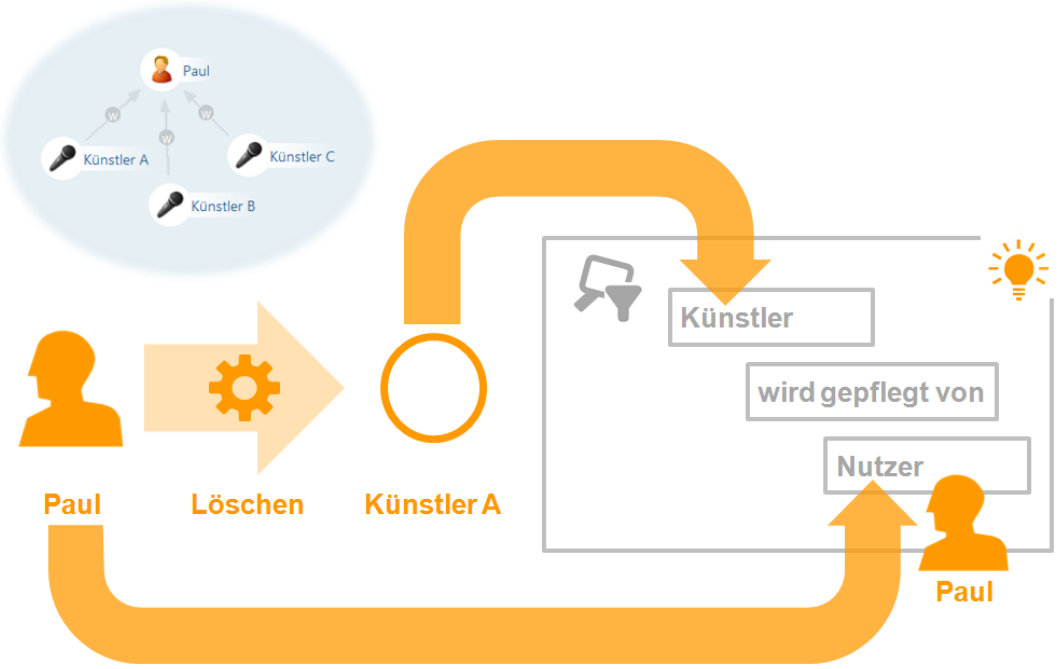


In diesem Beispiel möchte die Benutzer Paul den Song X lesen. Diese Operation wird nun vom Rechtesystem geprüft. Dort ist ein Suchfilter definiert, der prüft, ob der Song bereits veröffentlicht ist. In der Strukturabfrage des Suchfilters werden Objekte vom Typ Song gesucht, mit der Einschränkung, dass das Attribut Erscheinungsdatum in der Vergangenheit liegt. Die Strukturabfrage liefert alle Songs, die diese Bedingung erfüllen. Ist der Song X einer davon, fällt die Prüfung des Filters positiv aus und der auf den Suchfilter nachfolgende Ordner (mit einem Filter oder Entscheider) wird ausgeführt.

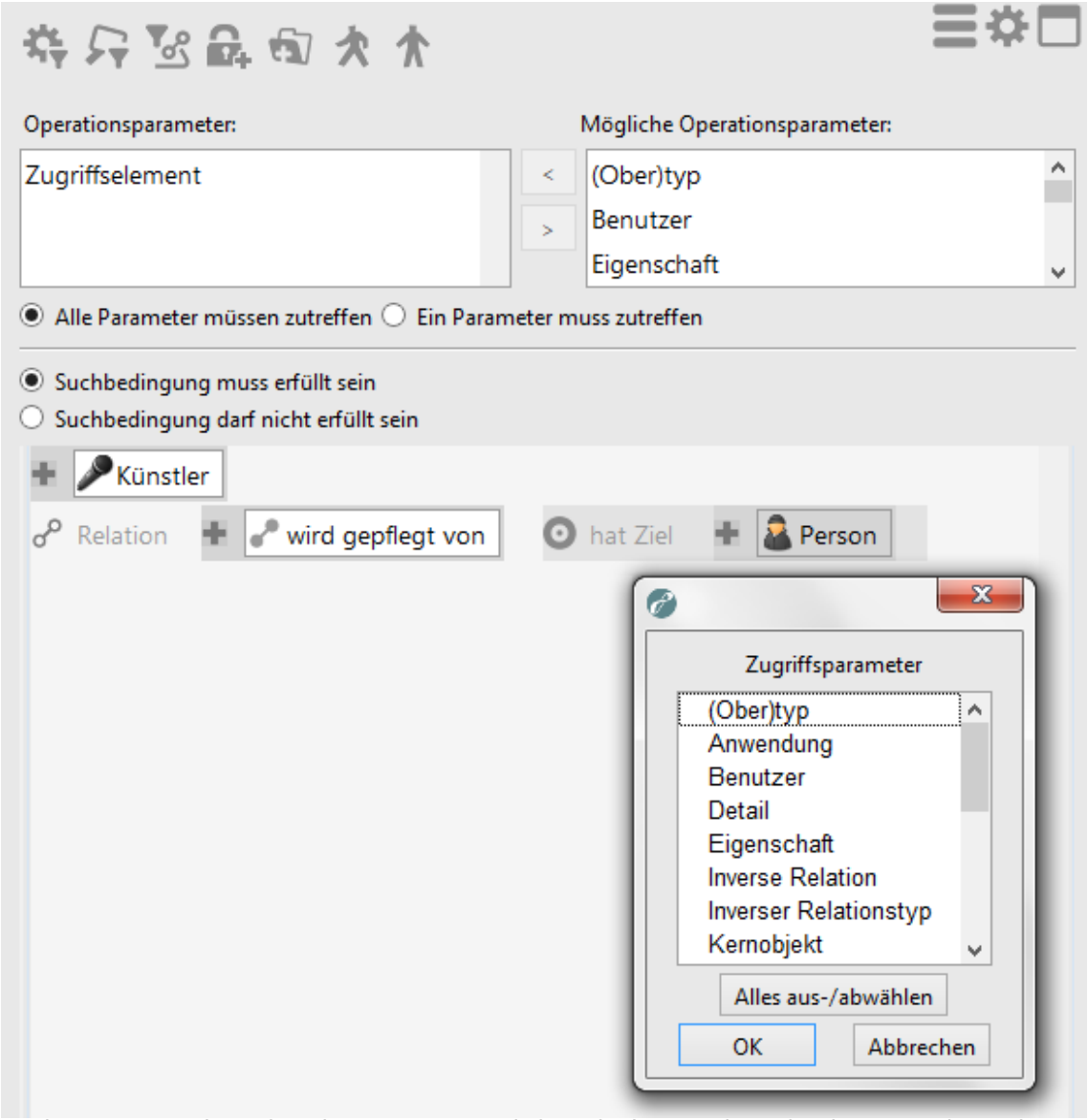
Bei dem Suchfilter wurden die Einstellungen Suchbedingung muss erfüllt sein und Alle Parameter müssen zutreffen ausgewählt.

Beispiel 2: Suchfilter im Rechtesystem

In den meisten Fällen gibt es eine Verbindung zwischen dem Benutzer, der zugreifen will und den Objekten oder Eigenschaften, auf die er zugreifen will. Ein Beispiel dafür wäre: "Mitarbeiter einer Abteilung die eine Branche betreuen, dürfen alle Kunden aus dieser Branche bearbeiten." Eine andere Version dieses Beispiels, das unten dargestellt wird, wäre: "Nutzer, die einen Künstler pflegen, dürfen diesen bearbeiten und löschen."



Auf der linken Seite ist ein Ausschnitt des Wissensnetzes abgebildet: Das Objekt Paul ist mit den Objekten Künstler A, Künstler B und Künstler C über die Relation pflegt verknüpft. Die inverse Relation von pflegt ist wird gepflegt von, die zwischen den Objekten Künstler A, Künstler B, Künstler C und dem Objekt Paul besteht und im Suchfilter abgefragt wird. Diese Relation im semantischen Netz steht dafür, dass eine Person für die Datenpflege rund um einen Künstler verantwortlich ist.

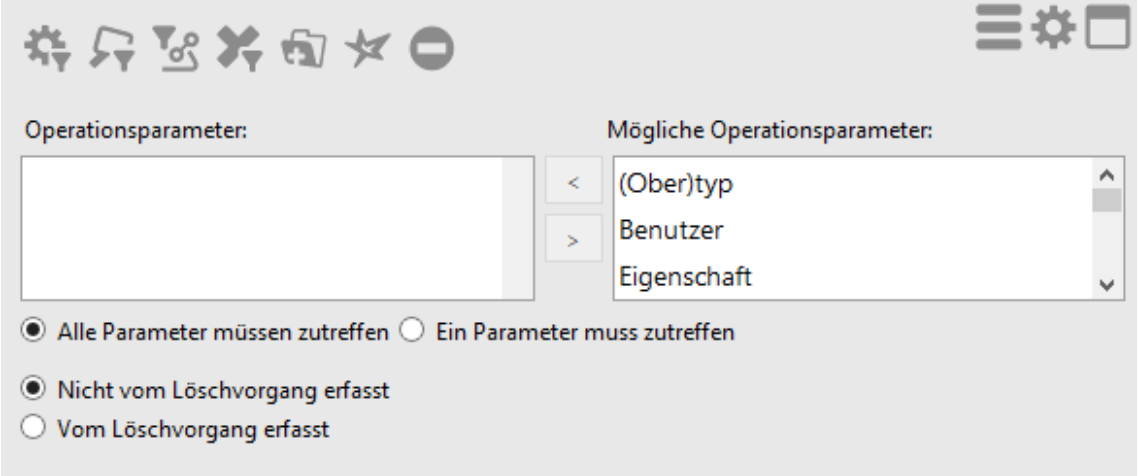


In diesem Beispiel möchte der Benutzer Paul das Objekt Künstler A löschen. Der dazugehörige Suchfilter liefert als Suchergebnis alle Künstler die von einem bestimmten Benutzer gepflegt werden. Der aktuelle Benutzer wird als Zugriffsparameter in die Strukturabfrage übergeben. Zugriffsparameter in Strukturabfragen werden im Kapitel Strukturabfragen erklärt. Somit liefert die Suche in dieser Zugriffsituation alle Künstler, die von Paul gepflegt werden. Da Künstler A einer davon ist, fällt die Prüfung des Suchfilters positiv aus.

Von der Zugriffssituation werden in diesem Beispiel zwei Aspekte in den Suchfilter eingebracht. Das ist der Künstler der gelöscht werden soll und der Benutzer. Der Suchfilter kann entsprechend auf zwei verschiedene Arten definiert werden. Entweder wird der Künstler als Zugriffselement an den Suchfilter übergeben und der Benutzer als Zugriffsparameter in der Strukturabfrage verwendet. Oder der Benutzer wird als Operationsparameter Benutzer an den Suchfilter übergeben und die Firma als Zugriffsparameter Zugriffselement in der Strukturabfrage verwendet.

1.6.3.4 Löschfilter

Löschfilter stehen nur bei der Definition von Triggern zur Verfügung. Sie werden dazu eingesetzt, in einer Löschsituation zu testen, ob das übergeordnete Element auch von dem Löschvorgang betroffen ist. Will man beispielsweise, dass ein Trigger nicht ausgeführt wird, wenn ein Objekt samt all dessen Eigenschaften gelöscht wird, aber dann wenn eine bestimmte Eigenschaft des Objektes gelöscht wird, muss ein Löschfilter verwendet werden.



Bei der Definition eines Löschfilters, muss mindestens ein Operationsparameter angegeben werden, der bestimmt, die Löschung welches Objektes getestet werden soll.

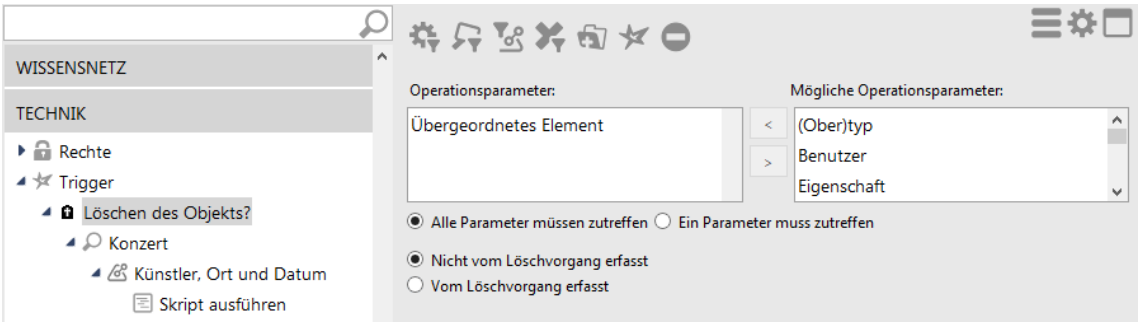
- *Alle Parameter müssen zutreffen*: Alle angegebenen Operationsparameter müssen zutreffen. Werden beispielsweise zwei Operationsparameter angegeben (Zugriffsobjekt und Primärobjekt), dann wird geprüft, ob der Löschvorgang sowohl für Zugriffsobjekt als auch für Primärobjekt gilt, das kann nur der Fall sein, wenn das Primärobjekt auch das Zugriffsobjekt ist.
- *Ein Parameter muss zutreffen*: Nur einer der angegebenen Operationsparameter muss zutreffen.

Anmerkung: In den meisten Fällen bietet sich der Operationsparameter übergeordnetes Element oder Primärobjekt an, da überprüft werden soll, ob entweder nur die Eigenschaft gelöscht wird, oder ob die Eigenschaft gelöscht wird, weil das gesamte Objekt gelöscht wurde.

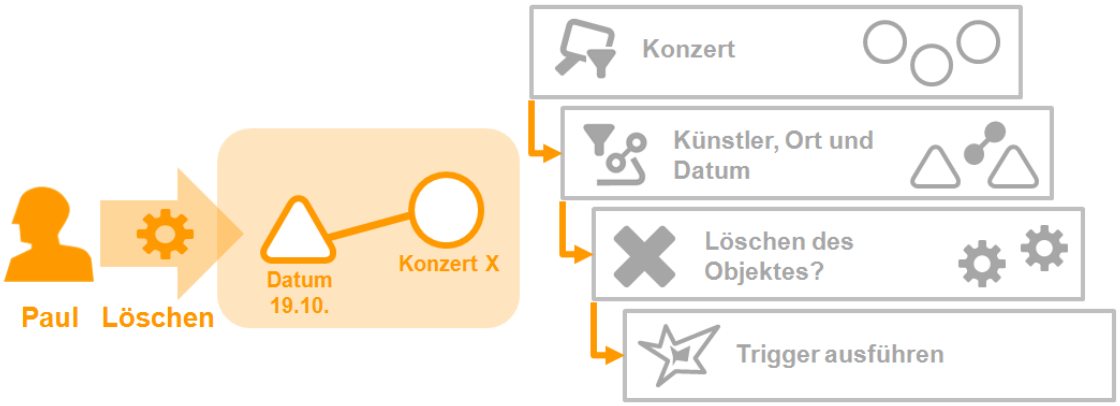
- *Nicht vom Löschvorgang erfasst*: Die Bedingung des Filters ist positiv, wenn das in Operationsparameter übergebene Element in dieser Transaktion nicht gelöscht wird.
- *Vom Löschvorgang erfasst*: Die Bedingung des Filters ist entsprechend positiv, wenn das in Operationsparameter übergebene Element in dieser Transaktion gelöscht wird.

Beispiel: Löschfilter bei Triggern

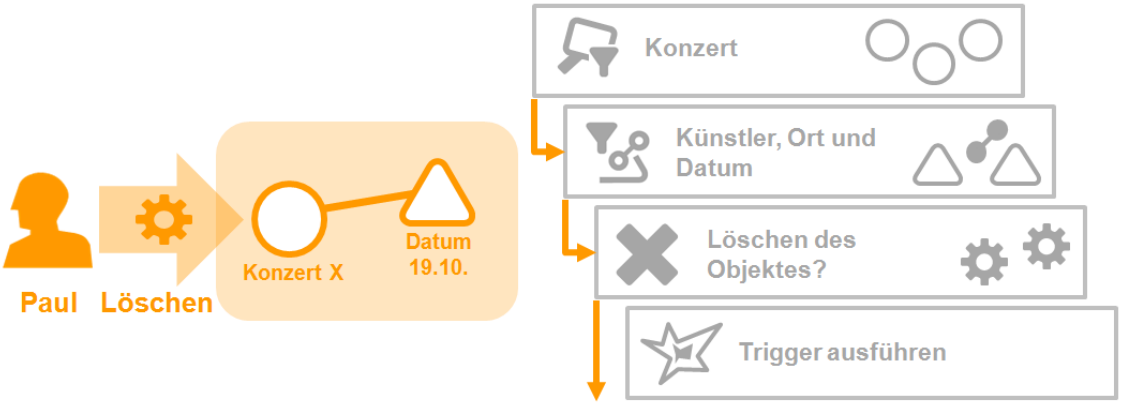
In diesem Beispiel soll ein Trigger nur dann ausgeführt werden, wenn der Künstler, der Ort oder das Datum einer Veranstaltung geändert oder gelöscht wird, aber nicht wenn das Objekt gelöscht wird, an denen die Eigenschaften gespeichert sind. Dafür wird die Einstellung *Nicht vom Löschvorgang erfasst* verwendet. Ist das übergeordnete Zugriffselement vom Löschvorgang erfasst, das in diesem Fall das Konzert-Objekt selbst ist, dann wird die Prüfung des Teilbaumes, aufgrund des negativen Ergebnisses des Filters, abgebrochen.



Verwendet wird der Operationsparameter Übergeordnetes Element und die Einstellung Nicht vom Löschvorgang erfasst.



In dieser beispielhaften Zugriffsituation wird das Attribut Datum mit dem Wert "19.10." am Objekt "Konzert X" gelöscht. Das Objekt selbst wird nicht gelöscht. Der Suchfilter "Konzert", der mit dem Operationsparameter übergeordnetes Zugriffselement definiert ist, und der Eigenschaftsfilter "Künstler, Ort und Datum" werden positiv beantwortet. Der darauffolgende Löschfilter liefert ebenfalls eine positive Antwort, da das Objekt an dem die Eigenschaft gespeichert ist (übergeordnetes Zugriffselement) nicht vom Löschvorgang betroffen ist - entsprechend der Einstellung Nicht vom Löschvorgang erfasst des Löschfilters.



In dieser Zugriffsituation wird das Objekt "Konzert X" vom Nutzer Paul gelöscht. Durch das Löschen des Objektes werden automatisch alle Eigenschaften des Objektes mit gelöscht - also auch alle Attribute des Objektes. Die Prüfung des Triggerbaums wird sowohl für die Löschung des Objektes

als auch des Attributes durchgeführt. Der Suchfilter "Konzert" und der Eigenschaftsfilter "Künstler, Ort und Datum" sind in der Prüfung des Triggerbaumes für den Löschvorgang des Attributs erfüllt. Der Löschfilter selbst ist in dieser Situation nicht erfüllt, da das Objekt "Konzert X" an dem die Eigenschaft "Datum 19.10." gespeichert ist, gelöscht wird.

Die Verwendung von Löschfiltern ist z.B. dann sinnvoll, wenn das Trigger-Skript den Namen des Objektes aus dessen Eigenschaften zusammensetzt. So wird der Name Objektes nicht erst mehrmals geändert, wenn die Eigenschaften des Objekts gelöscht werden, sondern das Objekt und alle damit verbunden Eigenschaften werden gelöscht ohne, dass das Skript ausgeführt wird, welches den Namen zusammensetzt. Dies erspart i.d.R. unnötige Berechnungszeit und kann in bestimmten Anwendungsszenarien, z.B. wenn der Trigger eine E-Mail Benachrichtigung schickt, dass ein Objekt umbenannt wird, durchaus sinnvoll sein (, da so das Verschicken von zahlreichen überflüssigen E-Mails zur Namensänderung vermieden wird).

1.6.4 Operationsparameter

Operationsparameter steuern bei Suchfiltern, mit welchem Element das Ergebnis der Strukturabfrage für die Prüfung der Bedingung verglichen werden soll. Im einfachsten Fall wird das Ergebnis mit dem Element verglichen, mit dem die zu prüfende Operation durchgeführt werden soll. Mithilfe von Operationsparametern kann das übergebene Element verändert werden. Es kann der aktuelle Benutzer oder Elemente aus dem Umfeld des Elements ausgewählt werden, die als Vergleichselement für den Suchfilter verwendet werden sollen.

Sie werden unter anderem auch bei Löschfiltern und Skript-Triggern verwendet. Dort geben sie an, ausgehend vom Element auf dem der Zugriff durchgeführt wird, auf welchem Element das Skript ausgeführt werden soll bzw. das Löschen welchen Elements gefiltert werden soll.

Wann ist dies sinnvoll? Statt des betroffenen Objekts ein Element aus dessen Umgebung zum Vergleich herziehen zu können, ist in manchen Fällen unverzichtbar: z.B. wenn es darum geht Zugriffsrechte für das Anlegen neue Objekte oder Typen zu prüfen. Es ist nicht möglich eine Strukturabfrage zu definieren, die das noch nicht angelegte Objekt zurückliefert. In diesem Fall muss der Suchfilter gegen etwas anderes verglichen werden, nämlich gegen den Typ des anzulegenden Objekts und bei Objekttypen gegen den Obertyp des anzulegenden Typs.

Operationsparameter	**Beschreibung**
(Ober)typ	Der (Ober)typ ist bei Typen der Obertyp des Typs. Bei Objekten ist der (Ober)typ der Typ des Objektes. Bei Attributen oder Relationen ist der (Ober)typ der Typ der Eigenschaft.
Benutzer	Der *Benutzer* ist das Objekt des Benutzers, der die Operation ausführt.
Eigenschaft	Die *Eigenschaft* ist die von der Operation betroffene Eigenschaft (Attribut oder Relation). Wird die Operation an einem Objekt, Typ oder Erweiterung durchgeführt, ist der Operationsparameter *Eigenschaft* leer.

Inverse Relation	Falls die von der Operation betroffene Eigenschaft eine Relation ist, enthält der Parameter die inverse Relationshälfte.
Inverser Relationstyp	Der *Inverse Relationstyp* ist der Typ der inversen Relation. Dieser kann bei Erzeugung von Relationen verwendet werden.
Kernobjekt	Wenn das übergeordnete Element eine Erweiterung ist, dann ist das *Kernobjekt* das Objekt an dem die Erweiterung gespeichert ist. Ansonsten ist das *Kernobjekt* identisch mit Zugriffselement.
Ordner	Der Operationsparameter *Ordner* ist der von der Operation betroffene Ordner.
Primäreigenschaft	Bei Metaeigenschaften ist die *Primäreigenschaft* die dem Objekt, Typ oder Erweiterung nächste Eigenschaft. Ansonsten ist *Primäreigenschaft* identisch mit Eigenschaft.
Primäres Kernobjekt	Wenn das Primärelement eine Erweiterung ist, dann ist das *Primäre Kernobjekt* das Kernobjekt der Erweiterung. Ansonsten ist das *Primäre Kernobjekt* identisch mit Kernobjekt.
Primäres Relationsziel	Das *Primäre Relationsziel* ist das Primärelement des Relationsziels.
Primärelement	Falls das übergeordnete Zugriffselement eine Eigenschaft ist, ist das *Primärelement* das Objekt, der Typ oder die Erweiterung an dem die Eigenschaft gespeichert ist (transitiv). Ansonsten ist das *Primärelement* identisch mit dem übergeordneten Element.
Relationsziel	Falls die von der Operation betroffene Eigenschaft eine Relation ist, enthält der Parameter *Relationsziel* das Relationsziel der Relationshälfte. (Die Relationsquelle wäre in diesem Fall das übergeordnete Element.)
Übergeordnetes Element	Das *Übergeordnete Element* ist das von der Operation betroffene Objekt, der Typ oder die Erweiterung. Bei Eigenschaften ist das *Übergeordnete Element* das Objekt, der Typ oder die Erweiterung an der die Eigenschaft gespeichert ist.

Zugriffselement	Das *Zugriffselement* ist das von der Operation betroffene Element.

1.6.4.1 Operationsparameter Zugriffselement

Das Zugriffselement ist das Element aus dem semantischen Netz auf das gerade zugegriffen wird. Bei Suchfiltern im Rechtesystem ist das Zugriffselement beispielsweise das Element auf das durch eine Operation zugegriffen werden soll. Beim Prüfen einer Zugriffsituation wird dann das Element an den Suchfilter übergeben, an dem die Operation durchgeführt werden soll. Der Suchfilter vergleicht dann das Zugriffselement mit dem Ergebnis der Strukturabfrage.

1.6.4.2 Operationsparameter Benutzer

Der Parameter Benutzer ist unabhängig vom Zugriffselement immer das Benutzerobjekt des aktuell angemeldeten Nutzers. Hierfür muss der Knowledge-Builder-Account mit einem Wissensnetzobjekt verknüpft werden. Wie die Verknüpfung vorgenommen wird, wird im Kapitel Aktivierung des Rechtesystems vorgestellt.

Zugriffselement	Benutzer
Objekt, Typ, Erweiterung oder Eigenschaft	Objekt des aktuell angemeldeten Nutzers

1.6.4.3 Operationsparameter (Ober)typ

Der Parameter (Ober)typ wird beispielsweise dann verwendet, wenn im Rechtesystem Operationen geprüft werden sollen, die neue Elemente anlegen. Beim Anlegen von Elementen kann der Suchfilter nicht so definiert werden, dass er das noch nicht angelegte Element findet. Der Suchfilter muss auf dem Obertyp oder Typ des Elements arbeiten, welches angelegt werden soll. Bei der Erstellung von Objekten, Attributen und Relationen wird der Typ des Objektes, Attributes oder der Relation verwendet. Bei Typen wird der Obertyp des anzulegenden Typs verwendet.

Zugriffselement	(Ober)typ
Objekt oder Erweiterung	Der Typ des Objektes oder der Erweiterung
Typ	Der Obertyp
Eigenschaft	Der Typ der Eigenschaft

1.6.4.4 Operationsparameter Übergeordnetes Element

Das übergeordnete Element wird dann verwendet, wenn direkte Eigenschaften eines Elementes abgefragt werden sollen.

Zugriffselement	Übergeordnetes Element
Objekt, Typ oder Erweiterung	Das Zugriffselement selbst
Eigenschaft	Objekt, Typ oder Erweiterung an dem oder der die Eigenschaft gespeichert ist
Metaeigenschaft	Eigenschaft, an der die Metaeigenschaft gespeichert ist

1.6.4.5 Operationsparameter Eigenschaft

Als Eigenschaften werden Attribute und Relationen verstanden. Der Operationsparameter enthält das Attribute oder die Relation auf der die Operation durchgeführt wird. Wird die Operation auf einem Objekt oder Typ durchgeführt, ist der Operationsparameter Eigenschaft leer.

Zugriffselement	Eigenschaft
Attribute oder Relation	Das Zugriffselement selbst
Objekt, Typ oder Erweiterung	Leer

1.6.4.6 Operationsparameter Inverse Relation

Die inverse Relation ist die "Gegenrichtung" einer Relationshälfte. Betrachtet man eine Relationshälfte als gerichteten Graphen, so besteht eine Relation aus zwei entgegengesetzten Graphen (der "Hinrichtung" und der "Rückrichtung" der Relation), die zwischen zwei Elementen aufgehängt ist. Die inverse Relation ist also die entgegengesetzte Relationshälfte. Die inverse Relationshälfte hat als Relationsziel die Relationsquelle der Relationshälfte und umgekehrt.

Zugriffselement	Inverse Relation
Relationshälfte	Die inverse Relationshälfte
Objekt, Typ, Erweiterung oder Attribut	Leer

1.6.4.7 Operationsparameter Inverser Relationstyp

Der inverse Relationstyp ist der Typ der inversen Relation.

Zugriffselement	Inverser Relationstyp
Relationshälfte	Typ der inversen Relationshälfte
Objekt, Typ, Erweiterung oder Attribut	Leer

1.6.4.8 Operationsparameter Relationsziel

Das Relationsziel ist nicht die Quelle sondern das "Ziel" einer Relationshälfte. Es kann auch als Relationsquelle der inversen Relationshälfte betrachtet werden.

Zugriffselement	Relationsziel
Relationshälfte	Das Relationsziel ist die Relationsquelle der inversen Relation
Objekt, Typ, Erweiterung oder Attribut	Leer

1.6.4.9 Operationsparameter Primärelement

Das Primärelement liefert immer ein Objekt, Typ oder Erweiterung. Wird das Primärelement auf Metaeigenschaften ausgeführt, werden die Eigenschaften transitiv abgearbeitet, bis das Objekt, der Typ oder die Erweiterung gefunden wurde, an dem die Eigenschaften aufgehängt sind.

Zugriffselement	Primärelement
Objekt, Typ oder Erweiterung	Das Zugriffselement selbst
Eigenschaft	Objekt, Typ oder Erweiterung an dem oder der die Eigenschaft gespeichert ist
Metaeigenschaft	Objekt, Typ oder Erweiterung an dem oder der die Eigenschaft gespeichert ist, an der wiederum die Metaeigenschaft gespeichert ist (transitiv)

1.6.4.10 Operationsparameter Primäres Relationsziel

Das primäre Relationsziel ist im Gegensatz zum Primärelement einer Relationshälfte nicht das Objekt, der Typ oder die Erweiterung an der die Relationshälfte angebracht ist sondern

das Objekt, der Typ oder die Erweiterung an der die inverse Relationshälfte aufgehängt ist.

Zugriffselement	Primäres Relationsziel
Relationshälfte	Das Primärelement des Relationsziels (Objekt, Typ oder Erweiterung an dem oder der die inverse Relationshälfte gespeichert ist)
Relationshälfte deren Relationsziel eine Eigenschaft oder Metaeigenschaft ist	Das Primärelement des Relationsziels (Objekt, Typ oder Erweiterung der Metaeigenschaft oder Eigenschaft an der die inverse Relationshälfte gespeichert ist)
Objekt, Typ, Erweiterung oder Attribut	Leer

1.6.4.11 Operationsparameter Kernobjekt

Das Kernobjekt wird verwendet, wenn mit Erweiterungen gearbeitet wird. Das Kernobjekt liefert anstatt der Erweiterung das Objekt, an dem die Erweiterung gespeichert ist.

Zugriffselement	Kernobjekt
Objekt, Typ oder Eigenschaft	Das Zugriffselement selbst
Erweiterung	Das Objekt an dem die Erweiterung gespeichert ist

1.6.4.12 Operationsparameter Primäres Kernobjekt

Wenn bei einem Element das zugehörige Objekt oder der zugehörige Typ verarbeitet werden soll, muss das primäre Kernobjekt verwendet werden. Im Gegensatz zum Primärelement werden keine Erweiterungen zugelassen. Bei diesen wird das Kernobjekt ausgegeben.

Zugriffselement	Primäres Kernobjekt
Erweiterung	Das Objekt an dem die Erweiterung gespeichert ist
Objekt oder Typ	Das Zugriffselement selbst
Eigenschaft oder Metaeigenschaft einer Erweiterung	Das Objekt an dem die Erweiterung gespeichert ist
Eigenschaft oder Metaeigenschaft eines Objektes oder Typs	Primärelement - Objekt oder der Typ an dem die Eigenschaft gespeichert ist (transitiv)

1.6.4.13 Operationsparameter Primäreigenschaft

Die Primäreigenschaft ist immer eine Eigenschaft. Sie ähnelt dem Primärelement in der Hinsicht, dass sie transitiv Metaeigenschaften abarbeitet. Sie liefert aber im Gegensatz die letzte Eigenschaft die vor dem Primärelement kommt - also die Eigenschaft, die direkt am Primärelement gespeichert ist.

Zugriffselement	Primäreigenschaft
Eigenschaft	Das Zugriffselement selbst
Metaeigenschaft (oder Metaeigenschaft einer Metaeigenschaft)	Die Eigenschaft, die dem Objekt, Typ oder der Erweiterung am nächsten ist
Objekt, Typ oder Erweiterung	Leer

1.6.4.14 Operationsparameter Ordner

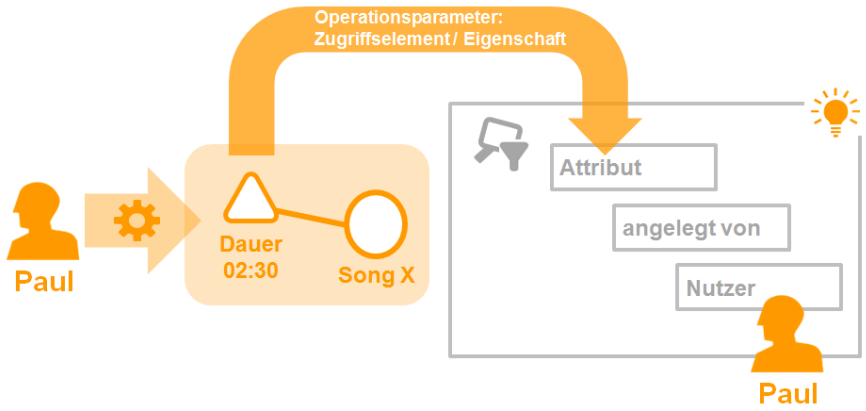
Soll ein Ordner aus dem Bereich *Ordner* des Wissensnetzes als Parameter an die Suche übergeben werden, dann muss der Operationsparameter Ordner verwendet werden.

Zugriffselement	Ordner
Ordner	Das Zugriffselement selbst
Objekt, Typ, Erweiterung oder Eigenschaft	Leer

1.6.4.15 Beispiele: Die Verwendung von Operationsparametern

Beispiel 1: Zugriffselement und Eigenschaft im Rechtesystem

Das unten aufgeführte Beispiel zeigt auf der linken Seite die Zugriffsituation und auf der rechten Seite den dazugehörigen Suchfilter.



Zugriffsituation: Der Nutzer Paul möchte das Attribut Dauer von Song X ändern.

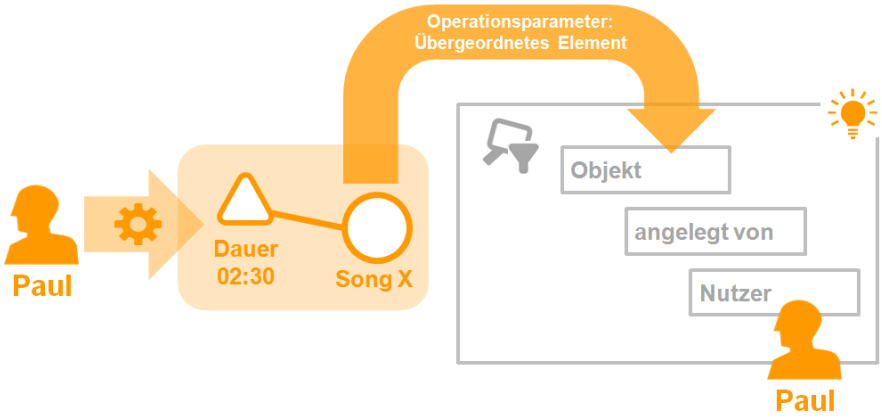
Suchfilter: Es werden alle Attribute gefiltert die, die von einem bestimmten Benutzer angelegt wurden. In der Strukturabfrage wird der Zugriffsparameter Benutzer verwendet, der die Objekte von Nutzer auf die Person einschränkt, welche die Operation ausführen möchte. Entsprechend sind das alle Attribute, die von Paul angelegt wurden.

Prüfung der Zugriffsrechte: Für die Prüfung der Zugriffsrechte wird das Attribut (das Zugriffselement/die Eigenschaft), an dem die Operation durchgeführt werden soll, an den Suchfilter übergeben. Ist dieses Attribut in der Menge der Suchergebnisse enthalten, dann ist die Prüfung des Suchfilters positiv.

Operationsparameter: Das Attribut Dauer selbst wird an den Suchfilter übergeben. In diesem Fall könnte sowohl der Operationsparameter Zugriffselement als auch Eigenschaft verwendet werden, da das Attribut Dauer selbst eine Eigenschaft ist und das Zugriffselement der Operation darstellt.

Beispiel 2: Übergeordnetes Element und Primärelement im Rechtesystem

Dieses Beispiel zeigt auf der linken Seite die Zugriffsituation und auf der rechten Seite den dazugehörigen Suchfilter.



Zugriffssituation: Der Nutzer Paul nimmt eine Änderung des Attributes Dauer, das aktuell den Wert 02:30 annimmt und zum Objekt Song X gehört, vor.

Suchfilter: Der Suchfilter ist so definiert, dass er alle Objekte sucht, die von einem bestimmten Benutzer angelegt wurden, das ist als Zugriffselement der aktuell angemeldete

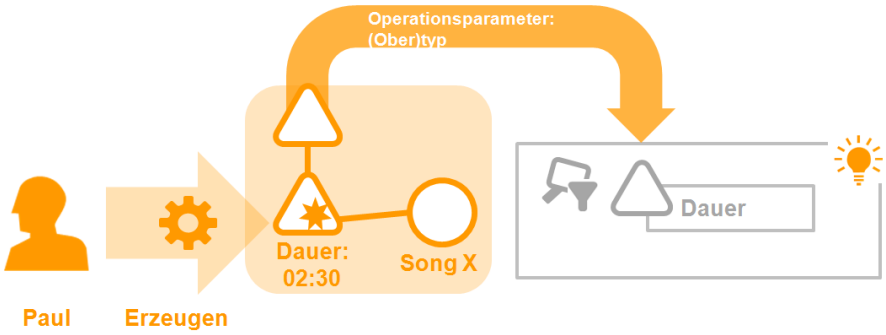
Nutzer. Entsprechend findet der Suchfilter alle Objekte, die von Paul angelegt wurden.

Prüfung der Zugriffsrechte: Ist in der Ergebnismenge des Suchfilters der Song X enthalten, wird der nachfolgende Ordner (Filter oder Entscheider) ausgeführt.

Operationsparameter: Die Verwendung des Operationsparameter übergeordnetes Element führt dazu, dass nicht das Attribut Dauer an dem die Änderung stattfinden soll, an den Suchfilter übergeben wird, sondern das Objekt an dem es definiert wurde. Das ist in diesem Fall der Song X. Neben dem übergeordneten Element könnte in diesem Fall auch der Operationsparameter Primärelement verwendet werden. Der Operationsparameter übergeordnetes Element führt dazu, dass alle Eigenschaften und das Objekt selbst positiv von Filter bewertet würde. Zusätzlich würde der Operationsparameter Primärelement auch Metaeigenschaften des Objektes zulassen, egal wie viele andere Eigenschaften zwischen Objekt und Metaeigenschaft hängen.

Beispiel 3: (Ober)typ im Rechtesystem

Das Beispiel stellt auf der linken Seite die Zugriffsituation dar und auf der rechten Seite wird der Suchfilter abgebildet, der in dieser Situation zum Einsatz kommt.



Zugriffsituation: Der Nutzer Paul möchte das Attribut Dauer am Objekt Song X erstellen. Es soll den Wert 02:30 haben.

Suchfilter: Der Suchfilter liefert den Attributtyp Dauer.

Prüfung der Zugriffsrechte: Ist das zu erstellende Attribut vom Typ Dauer, dann fällt die Prüfung des Suchfilters positiv aus.

Operationsparameter: Bei der Erstellung von Elementen, kann kein Suchfilter definiert werden, der das zu erstellende Element zurückliefert und damit die Zugriffsrechte prüfen kann. Bei der Erstellung von Elementen muss also ein anderer Operationsparameter als Zugriffselement ausgewählt werden. Der Operationsparameter (Ober)typ ist in diesen Situationen geeignet. In diesem Beispiel wird der Typ des Attributes verwendet, das ist der Attributtyp Dauer.

1.6.5 Operationen

In Operationsfiltern können Operationen angegeben werden, die dann im Filterprozess von Operationsfilter zugelassen werden. Wird in der Zugriffssituation eine andere Operation ausgeführt, als im Operationsfilter angegeben, wird bei der Traversierung des Rechte bzw. Triggerbaumes zum nächsten Teilbaum gewechselt.

Die allgemeinen Operationen *Erzeugen*, *Lesen*, *Modifizieren* und *Löschen* bestehen aus mehreren einzelnen Operationen. Wird eine der Operationsgruppen verboten, werden somit auch alle darin enthaltenen Operationen nicht erlaubt und umgekehrt wird eine Operationsgruppe

erlaubt, so werden alle enthaltenen Operationen automatisch mit erlaubt.

Die Tabelle zeigt eine Übersicht zu allen verfügbaren Operationen, die in Operationsfiltern ausgewählt werden können. Je nach Operation können nur bestimmte Operationsparameter in Suchfiltern verwendet werden. Diese werden in der Spalte Operationsparameter angegeben.

Anmerkung: Abgeleitete Operationsparameter wie z.B. Primärelement oder primäres Kernobjekt können immer dann eingesetzt werden, wenn der Parameter von dem sie abgeleitet sind, verwendet werden kann.

Besonderheiten bei Triggern
Bei Triggern können keine lesenden Operationen verwendet werden. Außerdem stehen bei Triggern die Operationsgruppen Abfrage (Operation: In Strukturabfrage verwenden), Anzeige von Objekten (Operation: Im Grapheditor anzeigen) und Bearbeiten (Operation: Attributwert validieren) nicht zur Verfügung.

Außerdem steht bei den Erzeugen Operationen bei Triggern der Operationsparameter Zugriffselement zur Verfügung, wenn Zeitpunkt/Art der Ausführung auf *Nach der Änderung* oder *Ende der Transaktion* gesetzt ist.

Operationsgruppe	Operation	Operationsparameter
Abfrage	In Strukturabfrage verwenden	Zugriffselement
Anzeigen von Objekten	im Grapheditor anzeigen	Zugriffselement
Bearbeiten	Attributwert validieren	Zugriffselement, Eigenschaft, übergeordnetes Element, (zu prüfender Parameter: Attributwert)
Benutzerdefinierte Operation		
Erzeugen	Attribut erzeugen	(Ober)typ, übergeordnetes Element
	Erweiterung erzeugen	(Ober)typ, übergeordnetes Element, Kernobjekt
	Objekt erzeugen	(Ober)typ
	Ordner erzeugen	Ordner
	Relation erzeugen	(Ober)typ, übergeordnetes Element, Relationsziel, inverser Relationstyp
	Relationshälfte erzeugen	(Ober)typ, übergeordnetes Element, Relationsziel
	Typ erzeugen	(Ober)typ

	Übersetzung hinzufügen	Zugriffselement, Eigenschaft, übergeordnetes Element
Lesen	Alle Objekte/Eigenschaften des Typs lesen	(Ober)typ
	Attribut lesen	Zugriffselement, Eigenschaft, übergeordnetes Element
	Objekt lesen	Zugriffselement, übergeordnetes Element
	Relation lesen	Zugriffselement, übergeordnetes Element, Eigenschaft, inverse Relation, Relationsziel, inverses Relationsziel
	Typ lesen	Zugriffselement, übergeordnetes Element
Löschen	Attribut löschen	Zugriffselement, übergeordnetes Element
	Erweiterung löschen	Zugriffselement, Eigenschaft, übergeordnetes Element
	Objekt löschen	Zugriffselement, übergeordnetes Element
	Ordner löschen	Ordner
	Relationshälfte löschen	Zugriffselement, inverse Relation, Eigenschaft, übergeordnetes Element, Relationsziel, inverses Relationsziel
	Typ löschen	Zugriffselement, übergeordnetes Element
	Übersetzung entfernen	Zugriffselement, Eigenschaft, übergeordnetes Element
Modifizieren	Arttributwert modifizieren	Zugriffselement, Eigenschaft, übergeordnetes Element
	Ordner modifizieren	Ordner
	Schema modifizieren	Zugriffselement, übergeordnetes Element
	Typ wechseln	Zugriffselement, übergeordnetes Element
Werkzeuge verwenden	Export	
	Import	
	Script bearbeiten/ausführen	

Objekt lesen
Die Operation *Objekt lesen* deckt das Anzeigen von Objekten auf dem Reiter Objekte bei dem enstprechenden Objekttyp ab. Die Operation verbietet aber nicht das Anzeigen des Objektes, wenn es über ein verknüpftes Objekt aufgerufen wird. In diesem Fall gelten dann die Operationen für Eigenschaften *Attribut lesen* und *Relation lesen*.

In Strukturabfrage verwenden
Ist ein negatives Zugriffsrecht für ein Element definiert, das auf die Operation *In Strukturabfrage verwenden* gefiltert wird, dann darf das Element nicht in einer Strukturabfrage verwendet werden. Es wird auch dann nicht in Strukturabfragen berücksichtigt, wenn der (abstrakte) Obertyp angegeben wird.

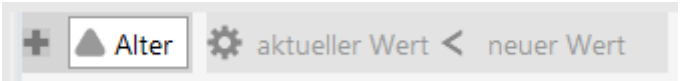
Attributwert validieren
Die Operation *Attributwert validieren* wird dann verwendet, wenn der zu setzende Attributwert bestimmte Bedingungen erfüllen muss. Die Definition der Bedingung an den Attributwert wird in einer Strukturabfrage gemacht. Dort stehen für die Validierung des Attributwertes zwei Definitionsmöglichkeiten zur Verfügung:

- *Bedingung für den zu setzenden Attributwert*:
  Der neue Wert des Attributes kann durch Vergleich mit einem angegebenen Wert in der Strukturabfrage validiert werden.

  Beispiel: Der Attributwert darf nur kleiner gleich 4,0 sein.

- *Vergleiche mit dem zu setzenden Attributwert*:
  Hierbei wird der aktuelle Wert mit dem neuen Wert verglichen.

  Beispiel: Der neue Wert des Attributs Alter darf in diesem Fall nur größer werden. Kleinere Werte werden nicht zugelassen.

Für die Validierung stehen verschiedene Vergleichsoperatoren zur Verfügung, mit denen der zu setzende Attributwert gegen einen anderen Wert geprüft werden kann.
Entspricht der neue Wert nicht der definierten Bedingung, so ergibt die Prüfung des Filters ein negatives Ergebnis, sofern die initiale Einstellung *Suchbedingung muss erfüllt sein* ausgewählt ist.

Schema modifizieren
Die Operation Schema modifizieren, betrifft Änderungen am Definitionsbereich von Relationen und Änderungen an der Typenhierarchie (*ist Untertyp von* und *ist Obertyp von* Relationen).

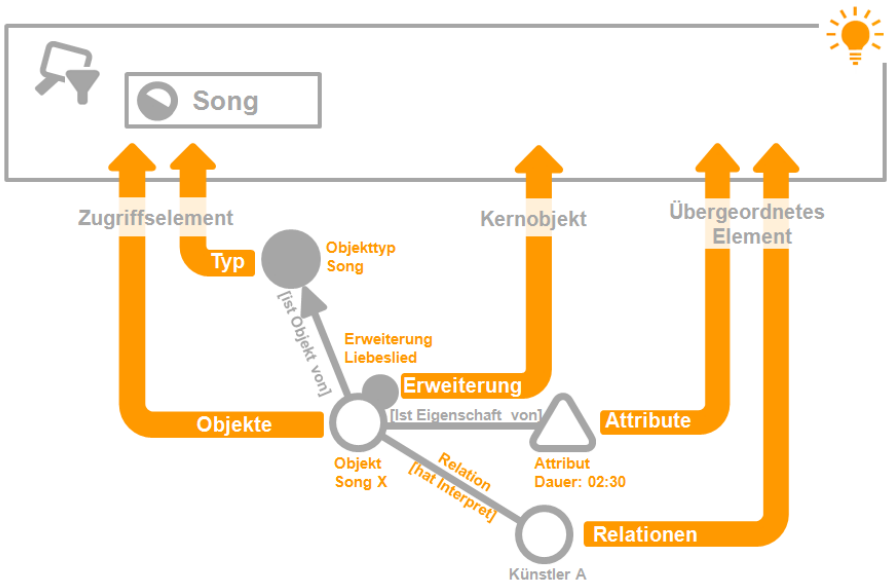
1.6.5.1 Beispiel: Die Verwendung von Operationsgruppen im Rechtesystem

In diesem Beispiel wird gezeigt wie Operationsgruppen (Lesen, Erzeugen, Modifizieren, Löschen) bei der Rechtedefinition sinnvoll eingesetzt werden können. Es sollen alle Operationen für den Typ Song und dessen Objekte verboten werden. Dies umfasst die folgenden Aktionen:

- Das Löschen des Objekttyps Song
- Das Löschen von bestimmten Songs (Objekte von Songs)
- Das Löschen von Attributen, welches an einem Song vorkommt
- Das Löschen von Relationen, die an einem Song vorkommt (Relationsziel und -quelle)

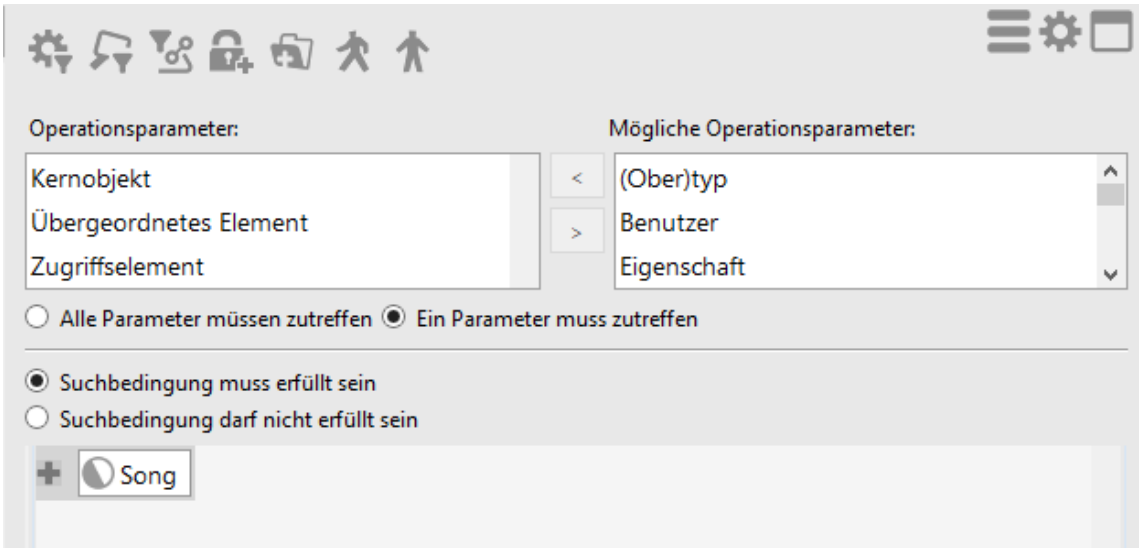
- Das Löschen von Erweiterungen, die Objekte von Song erweitern
- Das Löschen von Attribut- und Relationstypen die Objekte oder Untertypen von Song als Definitionsbereich haben

Sollen beispielsweise alle Löschen Operationen bei einem Objekt und dem dazugehörigen Typen verboten werden, muss man bei der Auswahl der Operationsparameter im Suchfilter des Rechtes darauf achten alle Löschen Operationen durch die entsprechenden Parameter abzudecken:



Der verwendete Suchfilter hat als einzige Bedingung den Objekttyp Song, bei dem die Einstellung Objekte und Untertypen ausgewählt ist. Der Operationsparameter Zugriffselement deckt den Objekttyp Song und alle Objekte, die zu diesem Typ gehören, ab. Der Parameter Kernobjekt deckt die Erweiterungsobjekte ab, die zu Songs gehören. Attribute und Relationen werden durch den Operationsparameter übergeordnetes Element abgedeckt.

Im Rechtebaum kommt der Operationsfilter der Operation Löschen an erster Stelle. Darauf folgt der unten abgebildete Suchfilter und als letztes der Entscheider Zugriff verweigert.



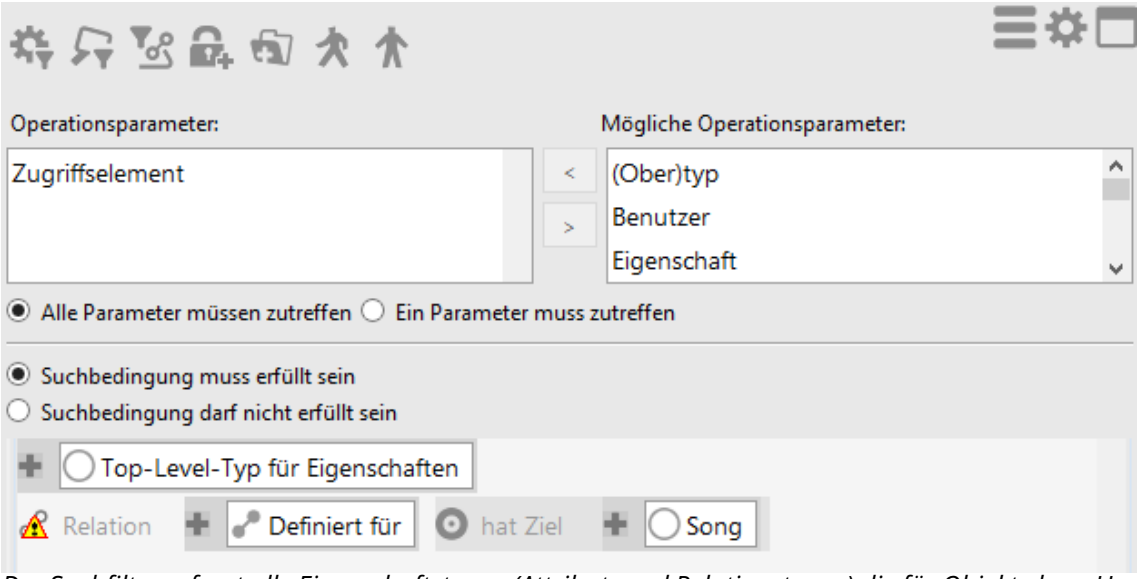
Im Beispiel verwendeter Suchfilter: Kernobjekt, übergeordnetes Element und Zugriffselement wur-

den als Operationsparameter ausgewählt. Die Einstellungen Ein Parameter muss zutreffen und Suchbedingung muss erfüllt sein werden verwendet.

Erweiterung des Rechtes um Attribut- und Relationstypen

Ein so definiertes Recht deckt die alle bis auf einen der oben formulierten Anforderungspunkte des Rechtes ab. Lediglich das Löschen von Attribut- und Relationstypen, die für Objekte und Untertypen von Songs definiert sind, wird in dieser Rechtedefinition nicht berücksichtigt.

Eine Erweiterung der Rechtedefinition wird durch den folgenden Filter realisiert:

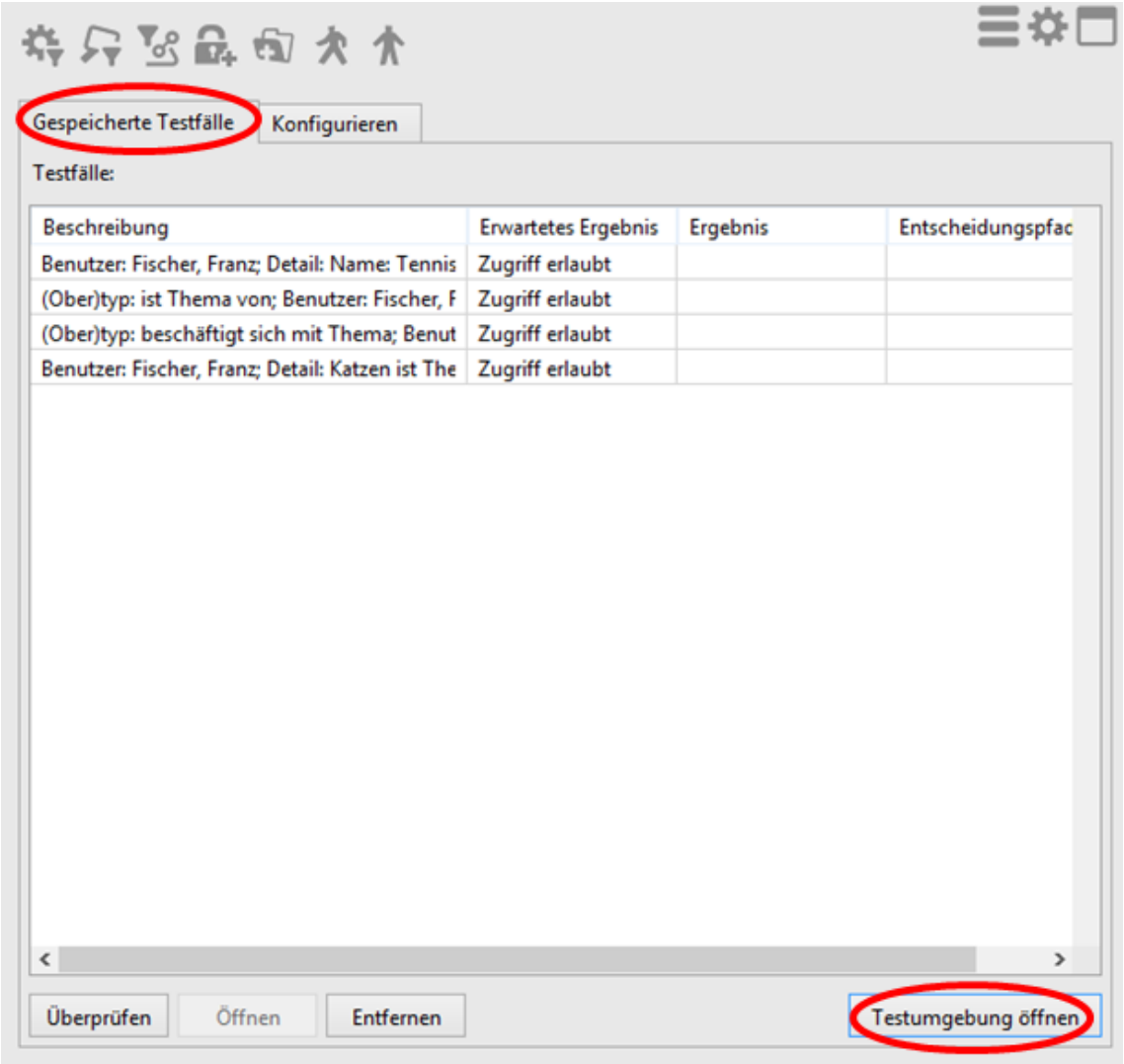


Der Suchfilter erfasst alle Eigenschaftstypen (Attribut- und Relationstypen) die für Objekte bzw. Untertypen von Songs definiert sind. In der Suchfilterdefinition wird der Parameter Zugriffselement und die Einstellung Suchbedingung muss erfüllt sein verwendet.

1.6.6 Testumgebung

Wird im Bereich *System* der Ordner *Rechte* ausgewählt, werden im Hauptfenster die Reiter *Gespeicherte Testfälle* und *Konfigurieren* angeboten. Der Bereich des Testsystems befindet sich im Reiter *Gespeicherte Testfälle*. Das Testsystem für Trigger wird über den Bereich *System* im Ordner *Trigger* aufgerufen.

Hier können die gespeicherten Testfälle erneut getestet werden. Die Testoberfläche in der die Testfälle definiert werden können, kann über die Schaltfläche *Testumgebung öffnen* aufgerufen werden.



Zusätzlich zu den Funktionalitäten, die in den folgenden Kapiteln Eine Zugriffssituation testen und Testfälle definieren beschrieben werden, gibt es die Möglichkeit direkt an einem Objekt oder Typ Zugriffsrechte zu testen. Über das Kontextmenü (rechte Maustaste) die Funktion Zugriffsrechte auswählen. Dort stehen die folgenden Menüpunkte zur Auswahl:

- **Objekt:** Es werden alle Operationen (Modifizieren, Löschen, Lesen und im Graph-Editor anzeigen) am Objekt geprüft und deren Ergebnis ausgegeben.
- **Alles:** Es werden alle Operationen (Modifizieren, Löschen, Lesen und im Graph-editor anzeigen) am Objekt und all dessen Eigenschaften (Attribute und Relationen) geprüft.
- **Testumgebung Berechtigungssystem:** Die Testumgebung für die Rechteprüfung wird geöffnet.

1.6.6.1 Eine Zugriffssituation testen

Zum Testen des Rechtesystems und der Trigger-Funktionalität sind zwei Bereiche relevant:

- Die Testumgebung selbst: Die Testumgebung bietet die Möglichkeit für einen bestimmten Testfall die Zugriffsrechte bzw. wann ein Trigger ausgeführt wird zu testen.

- Der Reiter *Gespeicherte Testfälle*: Hier werden die Testfälle aufgelistet und für spätere Überprüfungen zur Verfügung gestellt.

Anleitung zum Öffnen der Testumgebung

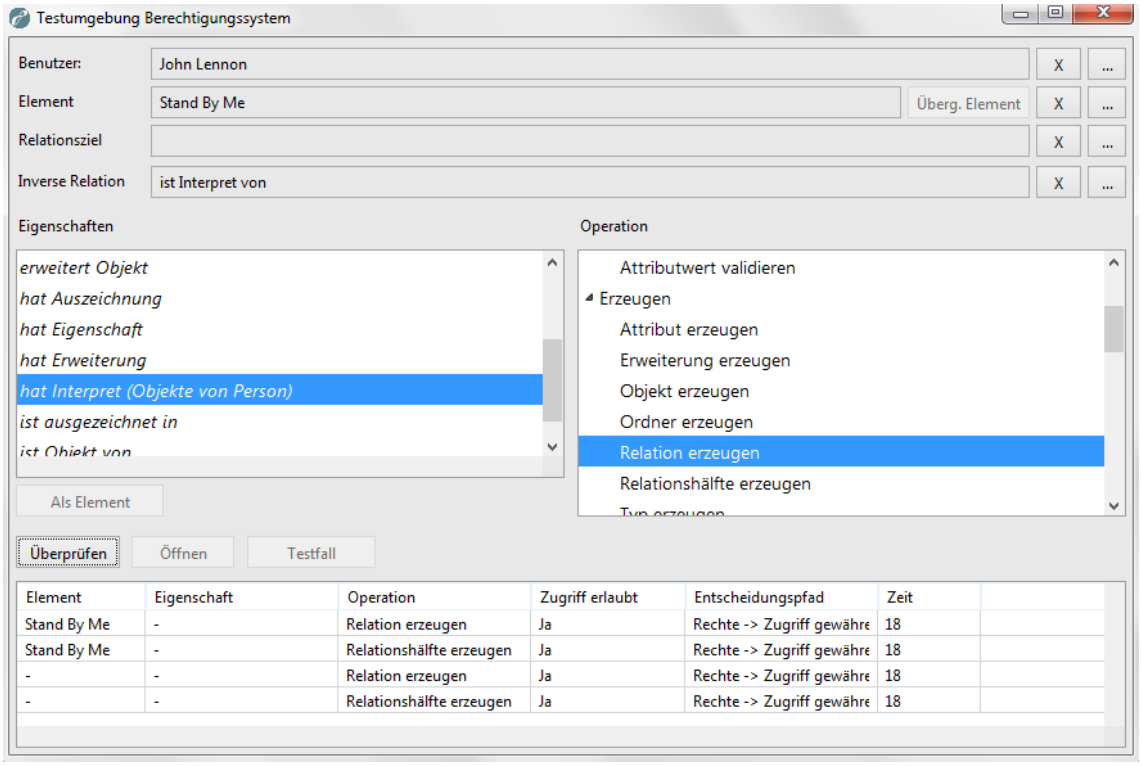
1. Wählen Sie im Knowledge-Builder im Bereich *Technik* den Ordner *Rechte* bzw. *Trigger* aus.
2. Wenn Sie im Rechtesystem arbeiten, wählen Sie im Hauptfenster den Reiter *Gespeicherte Testfälle* aus.
3. Klicken Sie *Testumgebung öffnen* (rechts unten) an, damit sich die Testumgebung in einem neuen Fenster öffnet.

Die Testumgebung besteht aus mehreren Bereichen: Im oberen Bereich wird der Benutzer und das Element definiert, an dem die Eigenschaft angebracht ist, die geprüft werden soll. Das Element kann ein Objektes, ein Typs oder eine Eigenschaft (wenn diese als Element übergeben wird) sein.

Der Bereich *Eigenschaften* listet alle Eigenschaften des ausgewählten Elements aus. Nicht kursive Eigenschaften, sind konkrete Eigenschaften, die bereits am Objekt oder der Eigenschaft vorliegen. Kursive Eigenschaften hingegen sind Eigenschaften, die vom Schema her angelegt werden können, aber noch nicht wurden. Soll die Erstellung einer neuen Eigenschaft getestet werden, muss die Eigenschaft in kursiv-Form ausgewählt werden.

Im Fenster *Operation* kann die Operation ausgewählt werden, die getestet werden soll. Je nach ausgewählten Parametern, ist eine Rechteprüfung möglich oder nicht.

Beachte: Soll eine Eigenschaft einer Eigenschaft also eine Metaeigenschaft getestet werden, dann muss die Eigenschaft im Eigenschaftsfenster markiert werden und die Schaltfläche *Als Element* ausgewählt werden. Dann wird beispielsweise bei Relationen die konkrete Relation zwischen zwei Objekten oder Eigenschaften als Objekt ausgewählt. Jetzt stehen im Eigenschaftsfenster alle Eigenschaften der konkreten Relation zur Verfügung. (Dies geht auch mit Attributen.) Über die Schaltfläche *Überg. Element* kann dieser Schritt wieder rückgängig gemacht werden.



Das Ergebnis der Prüfung wird im unteren Fenster angezeigt. Hierfür muss die Schaltfläche *Überprüfen* ausgewählt werden. Das Ergebnisfenster zeigt alle getesteten Fälle an.

- *Element*: das Objekt, der Typ oder die Eigenschaft an dem oder der die Eigenschaft definiert ist
- *Eigenschaft*: die konkrete Eigenschaft die getestet werden soll (ist leer wenn kursive Eigenschaften getestet werden)
- *Operation*: die Operation, die überprüft werden soll
- *Zugriff erlaubt*: das Ergebnis der Prüfung des Testfalls
- *Entscheidungspfad*: die entsprechenden Ordner, die zu dem Testergebnis führen
- *Zeit*: die Zeit, die für die Rechteprüfung benötigt wurde

Beachte: Bei der Prüfung von Relationen werden i.d.R. die Relation, die inverse Relation und beide Relationshälften einzeln getestet.

1.6.6.2 Testfälle definieren

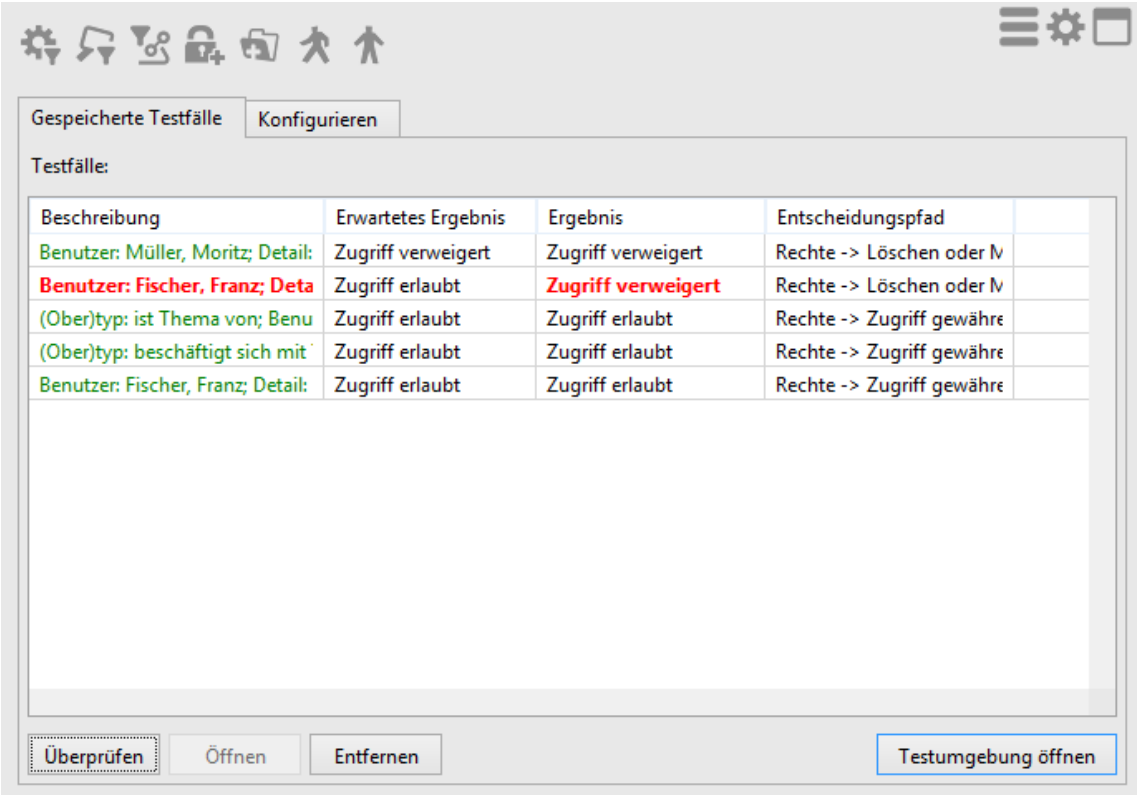
Um die Funktionalität des Rechtesystems zu überwachen, können Testfälle gespeichert werden. Dies ist gerade dann wichtig, wenn Änderungen am Rechtesystem vorgenommen werden und hinterher geprüft werden soll, ob das neue Ergebnis noch dem erwarteten Ergebnis entspricht. Alle gespeicherten Testfälle werden auf dem Reiter *Gespeicherte Testfälle* angezeigt. Dort können alle Testfälle gleichzeitig geprüft werden.

Anleitung zur Definition eines Testfalls

1. Wählen Sie in der Testumgebung das Element und die zu prüfende Eigenschaft aus.
2. Wählen Sie die Operation aus, die getestet werden soll.

3. Betätigen Sie die Schaltfläche *Überprüfen*. Jetzt werden die Zugriffsrechte für die abgegebenen Parameter getestet.
4. Wählen Sie in der Ergebnisausgabe den Testfall aus, der gespeichert werden soll. (Es kann immer nur eine Operation als Testfall gespeichert werden.)
5. Betätigen Sie die Schaltfläche Testfall. Der ausgewählte Testfall wird gespeichert und steht für spätere Prüfungen zur Verfügung.

Mehrere Testfälle gleichzeitig testen



Screenshot mit gespeicherten Testfällen, der zweite Testfall wird in Rot angezeigt.

In Grün werden alle Testfälle angezeigt, deren Testergebnis mit dem erwarteten Testergebnis übereinstimmen. Wird ein Testfall Rot angezeigt, dann ist das Ergebnis der Prüfung ein anderes als das erwartete Testergebnis. Das erwartete Testergebnis wird dadurch bestimmt, dass bei der Definition des Testfalls die Prüfung des Testfalls erstmalig durchgeführt wurde. Das Ergebnis dieser ersten Prüfung wird bei späteren Prüfungen des Testfalls als erwartetes Ergebnis angezeigt. Im Testsystem ist das erwartete Ergebnis entweder *Zugriff erlaubt* oder *Zugriff verweigert*; Bei Triggern ist das erwartete Ergebnis entweder *Skript ausführen* oder "nichts passiert" in Form eines Bindestriches.

Gespeicherte Testfälle können über *Entfernen* gelöscht werden. Soll ein Testfall bearbeitet werden, kann dies über die Schaltfläche *Testumgebung öffnen* gemacht werden. Der Testumgebung werden dann alle Parameter des Testfalls übergeben.

1.7 View-Konfiguration

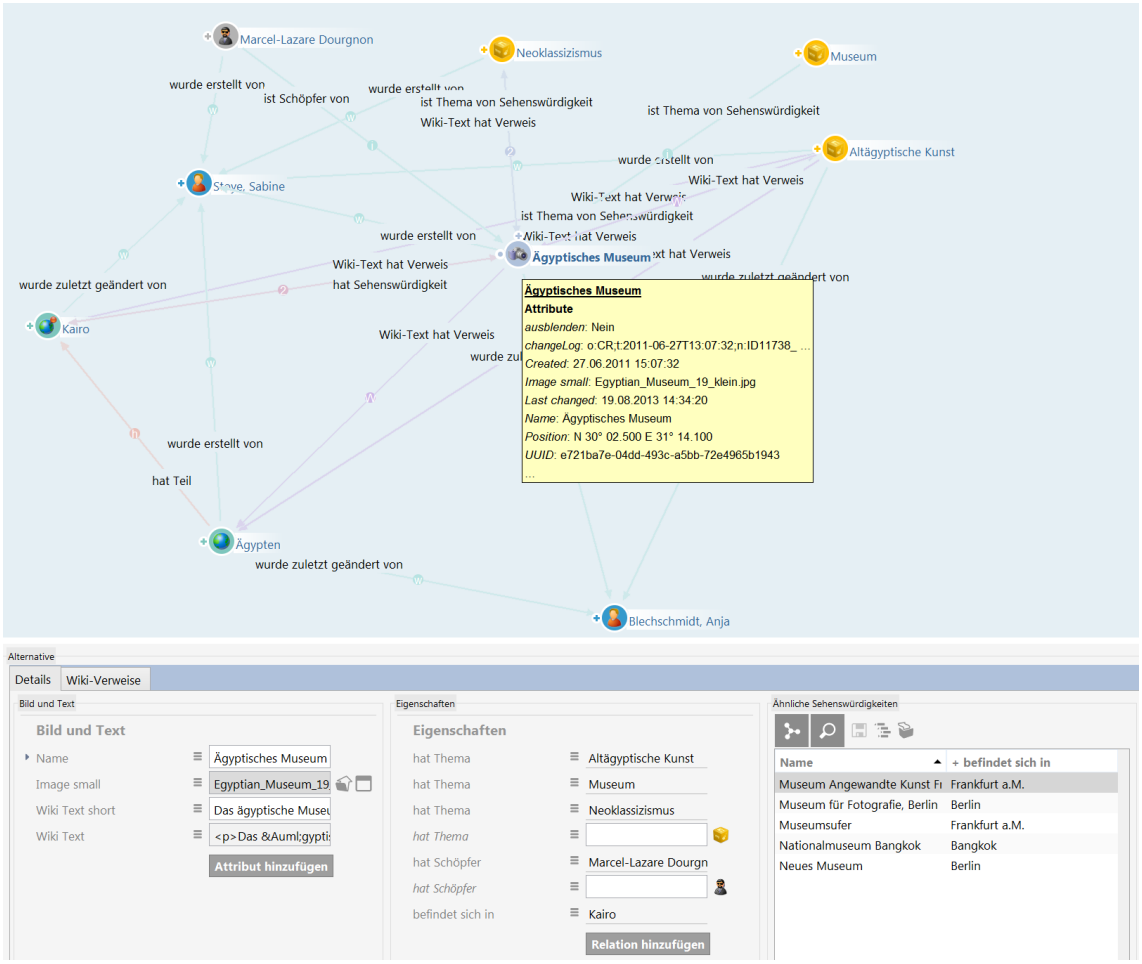
Die View-Konfiguration ermöglicht es verschiedene Sichten auf die Daten von i-views zu konfigurieren. Die konfigurierten Sichten kommen in Anwendungen zum Einsatz. Es können beispielsweise Teilausschnitte des semantischen Modells gezeigt oder bestimmte Zusam-

menstellungen der Daten (z.B. in Formularen, Tabellen, Ergebnislisten u.v.m.) erstellt werden.

So können wir u.a. folgende Fragen entscheiden und die entsprechend gewünschten Ansichten mit View-Konfigurationen erstellen:

Wie sollen die Eigenschaften von bestimmten Objekten dargestellt werden? In welcher Reihenfolge sollen die Eigenschaften dargestellt werden? Wenn wir ein neues Objekt anlegen, welche Attribute und Relationen sollen dann so dargestellt werden, damit sie auf keinen Fall übersehen und nicht ausgefüllt werden? Wie soll die Liste von Objekten zu einem Typ aussehen? Soll es überhaupt eine einfache Liste sein oder sollen die Objekte in Tabellen dargestellt werden? Welche Elemente sollen dann in den einzelnen Spalten zu sehen sein? Sollen Relationsziele direkt dargestellt werden? Oder nur bestimmte Attribute? Sollen wir verschiedene Reiter definieren, die zusammengehörige Eigenschaften und Attribute zusammenfassen? ...

Ein Beispiel: Konkrete Personen haben die Eigenschaften Name, Alter, Geschlecht, Adresse, Festnetznummer, E-Mail, Mobilnummer, Fax, *kennt*, *ist befreundet mit* und *ist Kollege von*. Nun könnten wir mithilfe der View-Konfiguration mehr Struktur in die Ansicht der Daten bringen, indem wir einen Reiter mit der Überschrift „Allgemeines“ definieren, der Name, Alter und Geschlecht zusammenfasst, einen mit der Überschrift „Kontaktdaten“, der Adresse, Festnetznummer, E-Mail, Mobilnummer und Fax beinhaltet und einen Reiter mit der Überschrift „Kontakte“, der die Eigenschaften *kennt*, *ist befreundet mit* und *ist Kollege von* enthält.



Beispiel einer View-Konfiguration. Oberer Screenshot: Unkonfigurierter Ausschnitt eines Objektes in der Graph-Ansicht mit allen seinen Eigenschaften. Unterer Screenshot: Konfigurierte Ansicht desselben Objektes, in der zusammengehörige Eigenschaften gruppiert, unwichtige Relationen wegge-

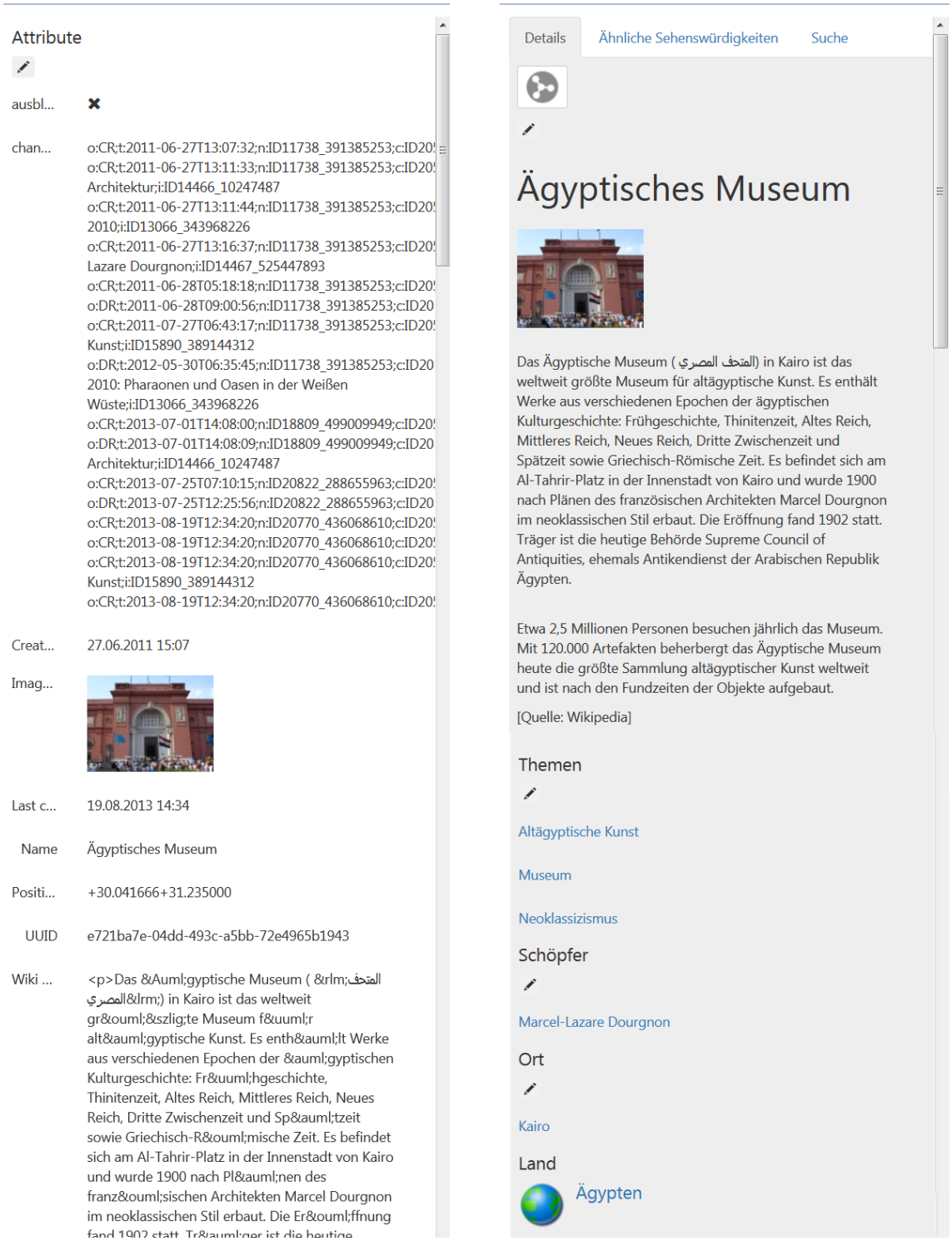
lassen und Ähnlichkeitsbeziehungen direkt dargestellt sind.

Ein Spezialfall der View-Konfiguration ist die Konfiguration der Ansicht der Daten im Knowledge-Builder, denn auch der Knowledge-Builder ist eine Anwendung, in der verschiedene Sichten auf die Daten möglich sind. Hilfreich ist dies dann, wenn wir den Knowledge-Builder als Preview benutzen wollen, um bestimmte Konfigurationen auszuprobieren. Die View-Konfiguration im Knowledge-Builder eignet sich außerdem, wenn Daten systematisch erfasst werden sollen, denn dann kann man beispielsweise die Detailseiten von Objekten so konfigurieren, dass wichtige zu ergänzende Eigenschaften gut sichtbar abgefragt werden.

1.7.1 Grundlagen der View-Konfiguration

1.7.1.1 Erstellung einer View-Konfiguration - Schritt für Schritt

Anhand eines Beispiels aus einer Datenbank zu Reisezielen und Sehenswürdigkeiten wird hier gezeigt, wie eine View-Konfiguration Schritt für Schritt erstellt wird. Die Detailseite zu Sehenswürdigkeiten in einer Web-Anwendung zur Reise-Datenbank soll konfiguriert werden. Die unkonfigurierte Standard-Ansicht einer Sehenswürdigkeit (links), listet alle ihre Attribute und Relationen untereinander auf, was in diesem Fall weder sinnvoll noch attraktiv für Anwender ist:

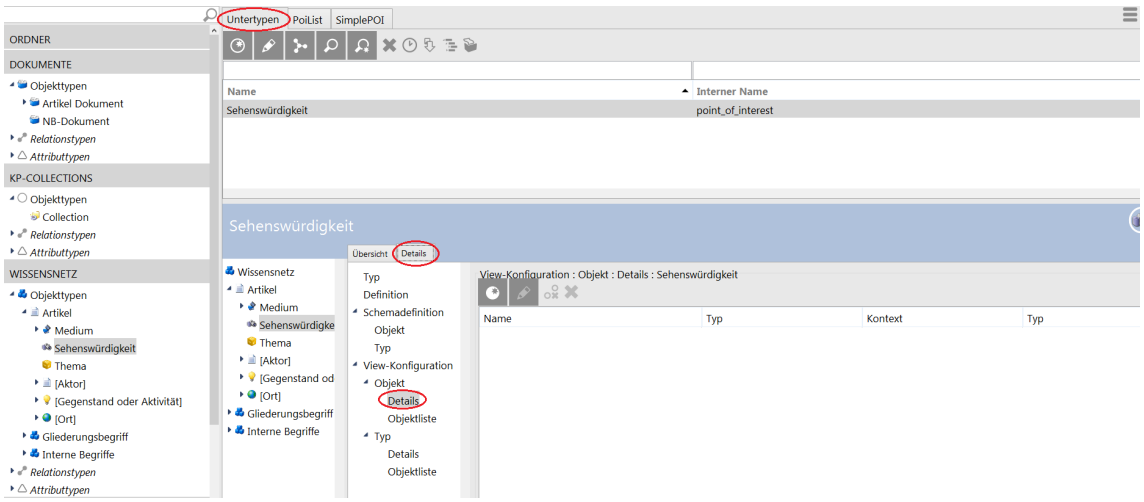


Die konfigurierte Ansicht (rechts) zeigt hingegen nur relevante und nützliche Informationen zur Sehenswürdigkeit, stellt diese übersichtlich, geblockt und mit passenden Überschriften dar und beinhaltet zudem eine Relation, die eine indirekte Beziehung darstellt: dass das Ägyptische Museum in Kairo steht, bedeutet es befindet sich im Land Ägypten. Darüber hinaus wurden der Graph-Button zur Ansicht des Objektes in einer Graph-Ansicht und mehrere Reiter hinzugefügt ("Details", "Ähnliche Sehenswürdigkeiten" und "Suche"), mit denen die Anwender definierte Aktionen durchführen können. Im Folgenden wird erläutert, wie diese Kon-

figurationen im Einzelnen erstellt werden.

1.7.1.1.1 Den View-Konfigurations-Editor öffnen

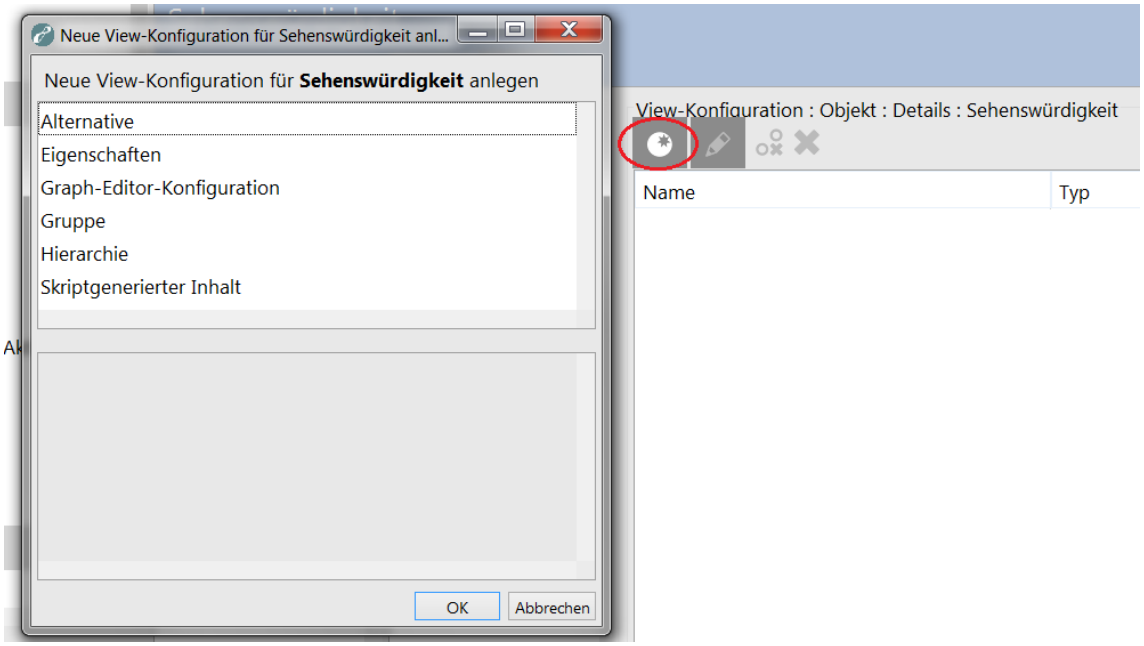
Wir wollen die Detail-Ansicht von Objekten des Typs "Sehenswürdigkeit" konfigurieren. Darum bewegen wir uns im Knowledge-Builder zum Typ "Sehenswürdigkeit", wählen dort den Reiter "Details" aus und klicken in der angezeigten Hierarchie dann unter "View-Konfiguration" -> "Objekt" auf "Details":



Unter "View-Konfiguration" stehen neben "Details" (von Objekten) noch andere Auswahlmöglichkeiten bereit:

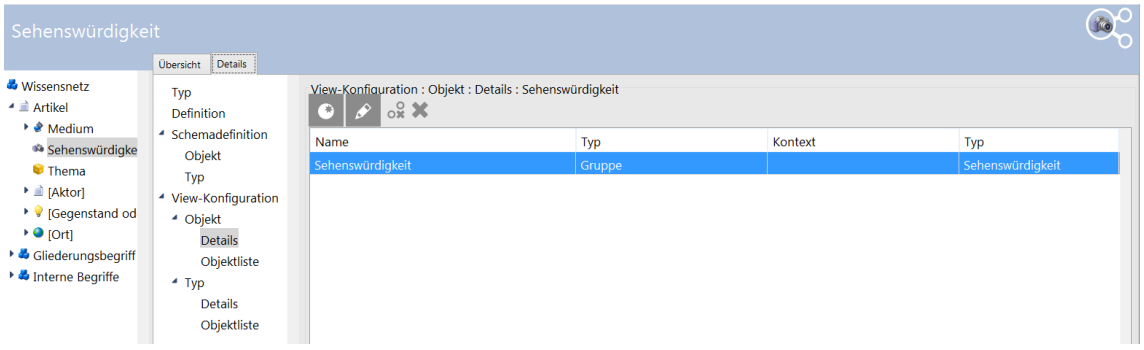
- Objekt -> Objektliste: Hier kann die Objektliste konfiguriert werden, die im Knowledge-Builder die Objekte des ausgewählten Typs anzeigt.
- Typ -> Details: Hier kann die Detailansicht zu Typen konfiguriert werden.
- Typ -> Objektliste: Hier kann die Objektliste der Untertypen des ausgewählten Typs konfiguriert werden, die im Knowledge-Builder zu sehen ist.

Mit einem Klick auf "Neu"  können wir hier eine neue View-Konfiguration anlegen. Für eine View-Konfiguration der Art Details bei Objekten stehen in einem sich anschließend öffnenden Dialog sechs Typen der View-Konfiguration bereit, von denen man einen auswählen muss:



Wir benötigen den Typ "Gruppe", da wir verschiedenste Unterkonfigurationen in einer Ansicht zusammenfassen wollen. (Die anderen Typen sind im Kapitel Detailkonfiguration für Objekte oder Typen erklärt.)

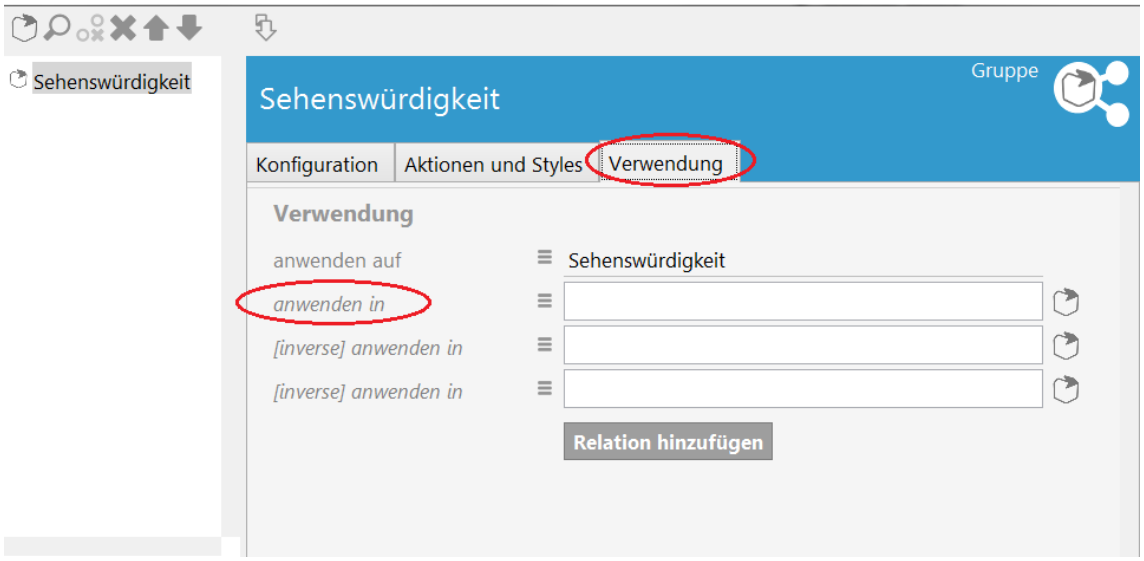
Nachdem wir den Typ der Konfiguration ausgewählt haben sehen wir unsere neu angelegte View-Konfiguration im rechten Fenster. Für den Namen der View-Konfiguration wurde automatisch der Name des Typs übernommen, der in Spalte vier noch einmal zu sehen ist. Die zweite Spalte zeigt den Typ der View-Konfiguration. In der dritten Spalte "Kontext" wird angezeigt, in welcher Anwendung die View-Konfiguration zum Einsatz kommt. Da wir noch keine Anwendung definiert haben, ist die Spalte noch leer.



Mit einem Klick auf den Bearbeiten-Button oder einem Doppelklick auf die ausgewählte View-Konfiguration öffnen wir den Editor, mit dem wir die Ansicht konfigurieren können.

1.7.1.1.2 Die Verwendung der View-Konfiguration festlegen

Bevor wir eine Ansicht konfigurieren, gehen wir auf den Reiter "Verwendung" um dort die Anwendung festzulegen, in der die View-Konfiguration gültig ist. Die Anwendung legt man unter *anwenden in* fest:

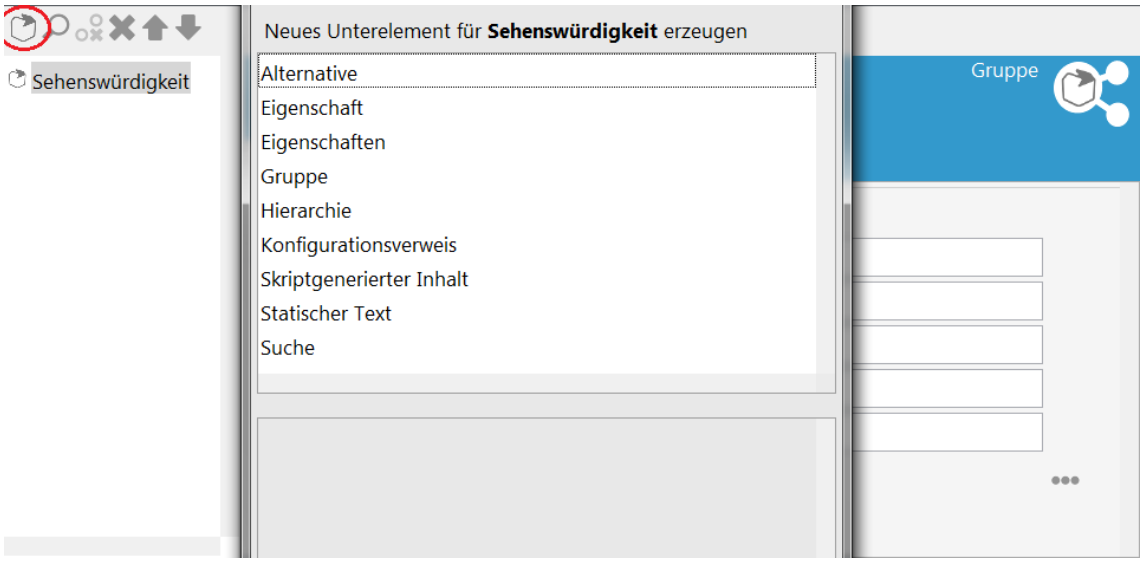


In diesem Beispiel wählen wir die Anwendung *Viewkonfiguration-Mapper* in dem Feld *anwenden auf* aus. Unter *anwenden auf* wird der Objekttyp angegeben, der in unserem Fall bereits übernommen ist.

Die View-Konfiguration hat eine Baumstruktur. Unser Wurzelelement ist die von uns zu Anfang angelegte Gruppe. Darum genügt es, wenn wir nur hier Angaben zur Verwendung machen, da das Prinzip der Vererbung auch in der View-Konfiguration greift und das Wurzelelement die Verwendung an seine Unterelemente vererbt.

1.7.1.1.3 Reiter konfigurieren

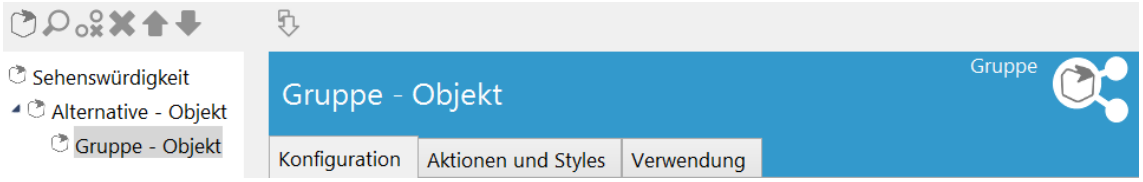
Um dem der erstellten Wurzelkonfiguration vom Typ Gruppe nun weitere Konfigurationselemente hinzuzufügen klicken wir "Objektkonfiguration neu anlegen" . Es öffnet sich ein Dialog, der alle möglichen Arten der View-Konfiguration anzeigt. (Die einzelnen Typen sind im Kapitel Detailkonfiguration für Objekte oder Typen erklärt.)



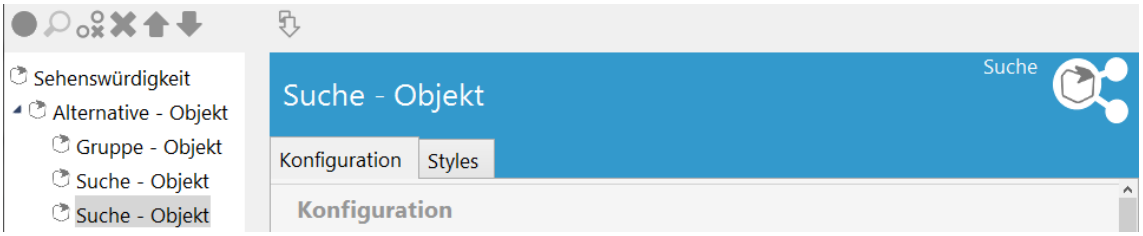
Da wir eine Darstellung mit Reitern erstellen wollen, müssen wir das Konfigurationselemente "Alternative" anlegen.

Die einzelnen Reiter müssen nun unter der "Alternative" angelegt werden.

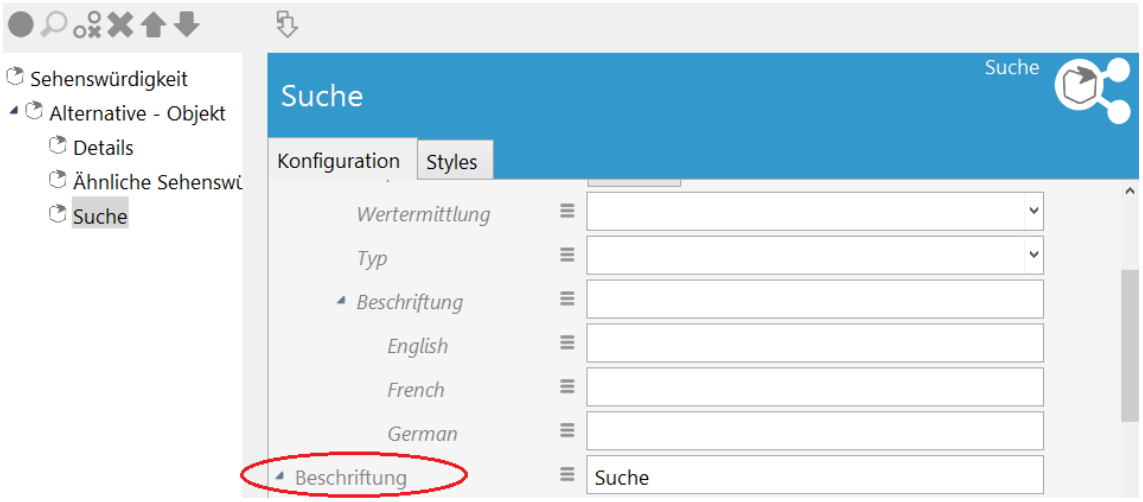
Der erste Reiter soll die Details der Sehenswürdigkeit zum Inhalt haben, also die wichtigen direkten Attribute und Relationen (Beschreibungstext, Bild, Themen, etc.). Hierfür könnten wir demnach das Konfigurationselement "Eigenschaften" wählen, mit dem man einzelne Eigenschaften eines Objekttyps anzeigen lassen kann. Da wir allerdings noch Überschriften definieren und eine einzelne indirekte Relation abbilden wollen, benötigen wir stattdessen ein weiteres Konfigurationselement des Typs "Gruppe".



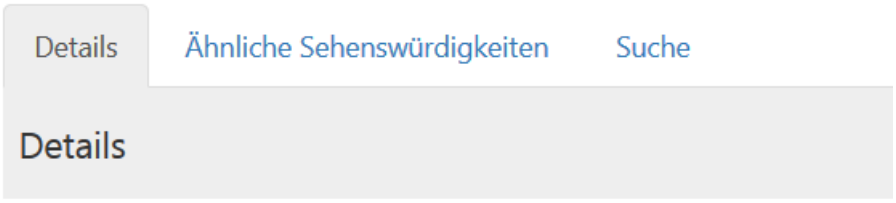
Der zweite Reiter soll ähnliche Sehenswürdigkeiten anzeigen, die mithilfe einer Ähnlichkeitssuche berechnet werden und der dritte Reiter soll die Suchfunktion der Web-Anwendung zugänglich machen. Für beide Reiter müssen wir demnach jeweils ein Konfigurationselement des Typs "Suche" wählen.



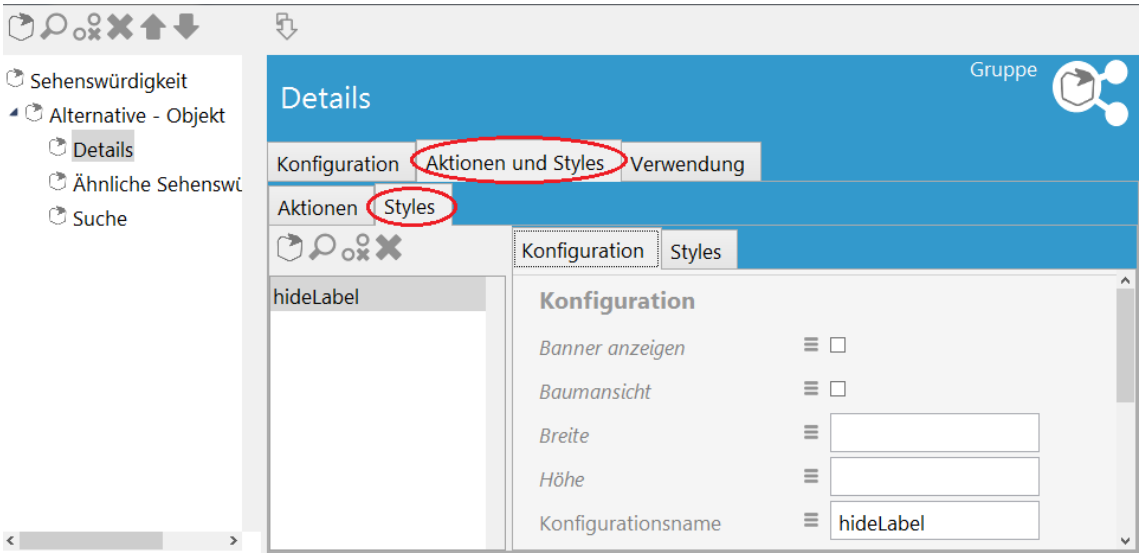
Wir müssen uns zu jedem Reiter eine passende Beschriftung überlegen, denn ohne Beschriftung kann der Reiter in der Anwendung nicht angezeigt werden. Diese können wir unter "Konfiguration" bei *Beschriftung* eintragen.

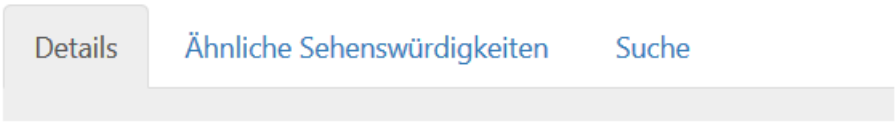


Wenn wir die View-Konfiguration nun über den Button "View-Konfigurationen aktualisieren" aktualisieren, können wir das Ergebnis in unserer Web-Anwendung betrachten:



Wir sehen, dass die Beschriftung nicht nur als Beschriftung der Reiter dient, sondern auch als Überschrift bei einem ausgeklappten Reiter. Wenn wir dieses im Prinzip doppelte Vorkommen der Überschrift nicht wollen, können wir jedem Reiter eine Style-Eigenschaft geben, die die Überschrift versteckt. Dazu müssen wir bei dem jeweiligen Konfigurationselement unter "Aktionen und Styles" mithilfe des Knopfes "Style hinzufügen" das Style-Element "hideLabel" auswählen.



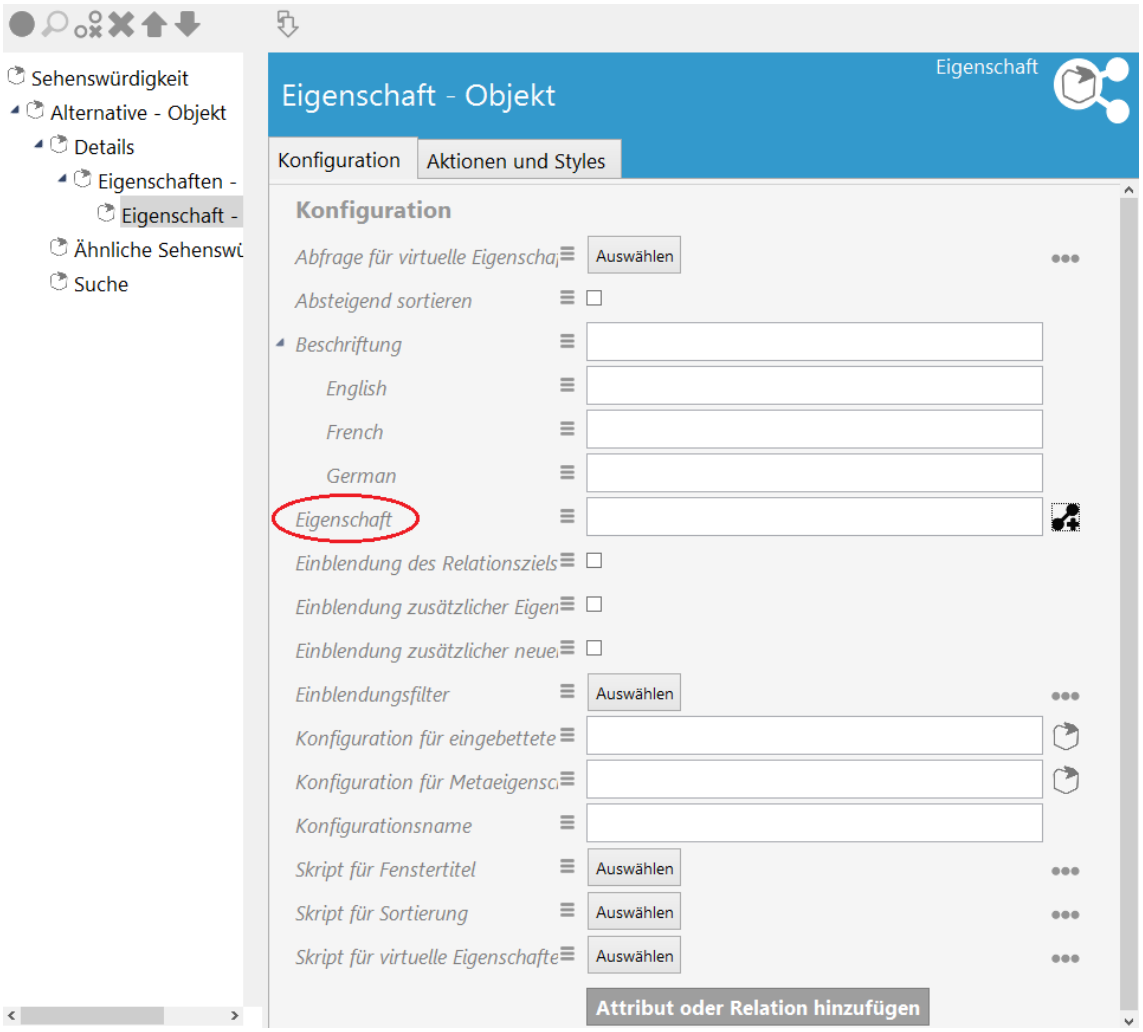


Das Ergebnis: unsere drei Reiter sind fertig konfiguriert. Mit dem Style-Element hideLabel können wir Beschriftungen verstecken.

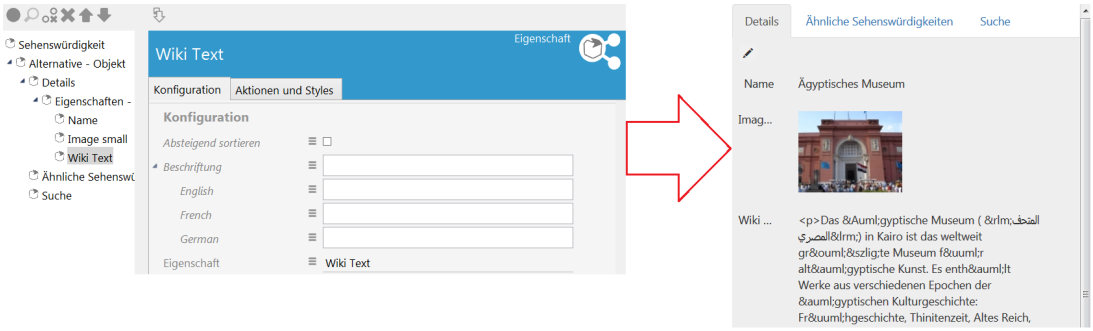
1.7.1.1.4 Direkte Eigenschaften konfigurieren

Nun wollen wir festlegen, welche Attribute und Relationen der Objekte des Typs Sehenswürdigkeit in dem Reiter "Details" zu sehen sein sollen. Den Reiter "Details", haben wir in unserem Konfigurationsbaum bereits als Gruppe definiert. Das Unterelement, das wir nun benötigen ist "Eigenschaften". Dieses Element dient dazu die verschiedenen Eigenschaften eines Objektes in einer Liste untereinander darzustellen. Mit einem weiteren Klick auf den Button "Eigenschaft neu anlegen", fügen wir der Eigenschaftsliste nun eine Eigenschaft hinzu. Damit haben wir in unserem Baum ein Blatt erreicht, d.h. wir sind an einem Punkt angelangt, bei dem wir schließlich eine Eigenschaft festlegen, die in einer Anwendung dargestellt werden soll. Wir können hier keine weiteren Unterelemente mehr anlegen.

Unter *Eigenschaft* müssen wir festlegen, welche Eigenschaft angezeigt werden soll. Eine Eigenschaft kann ein Attribut, eine Relation oder eine Abkürzungsrelation sein. Alle Einstellungsmöglichkeiten werden in Kapitel Eigenschaft erläutert.



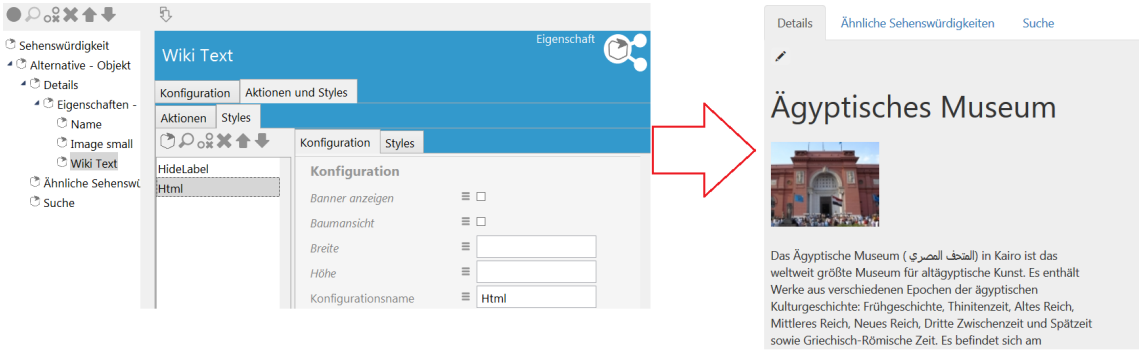
So definieren wir alle Eigenschaften der Sehenswürdigkeit, die wir für die Anwendung als sinnvoll erachten. Die Reihenfolge, wie wir sie im View-Konfigurationsbaum sehen ist dieselbe, wie wir sie nach der Aktualisierung über den Button "View-Konfigurationen aktualisieren" sehen:



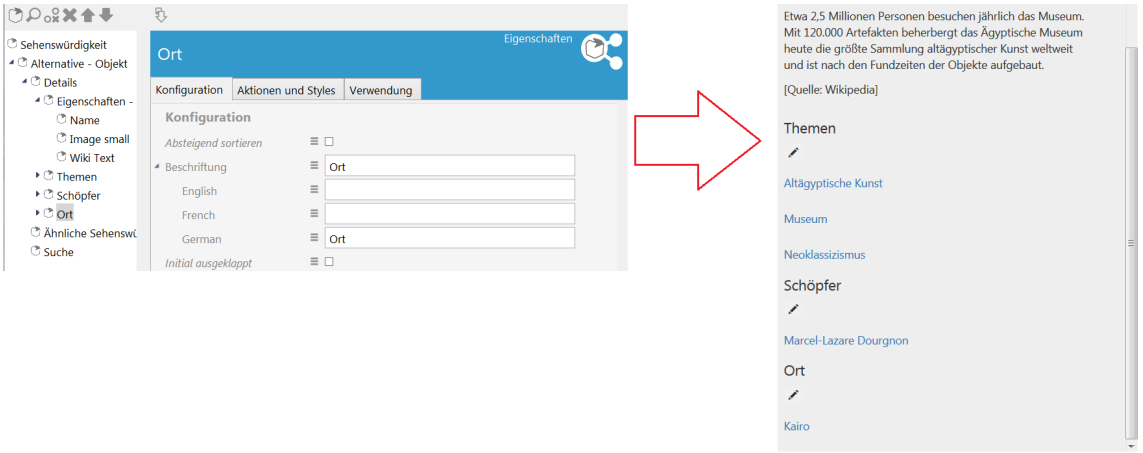
Um sie Ansicht noch etwas schöner zu gestalten, können wir verschiedene Style-Elemente verwenden:

- Mit dem Style-Element *HideLabel* können wir die Namen der Eigenschaften verstecken.
- Mit dem Style-Element *H1* können wir den Namen der Sehenswürdigkeit wie eine Überschrift darstellen.

- Mit dem Style-Element *Html* können wir den Text des Wiki-Textes ohne die HTML-Codes ausgeben.

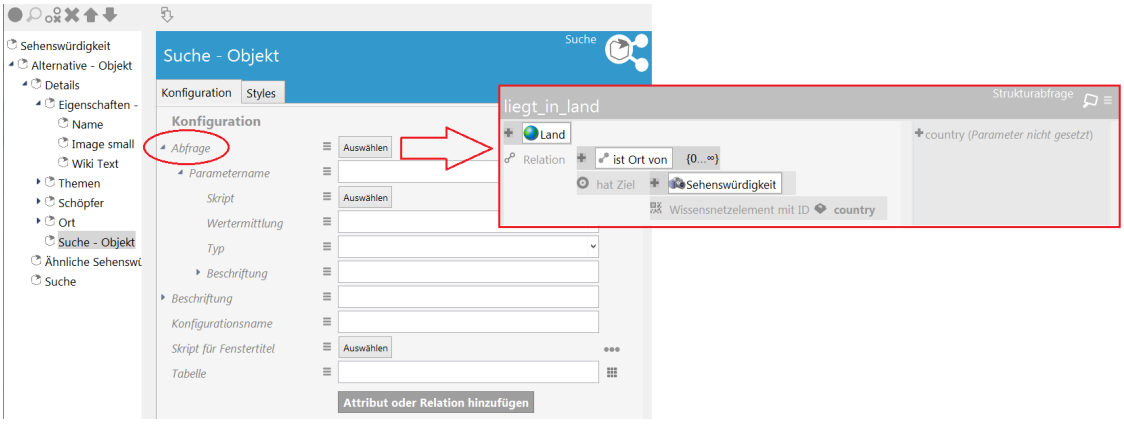


Mithilfe weiterer Eigenschaftslisten stellen wir nun noch die Themen der Sehenswürdigkeiten, sowie die Schöpfer und den Ort dar.



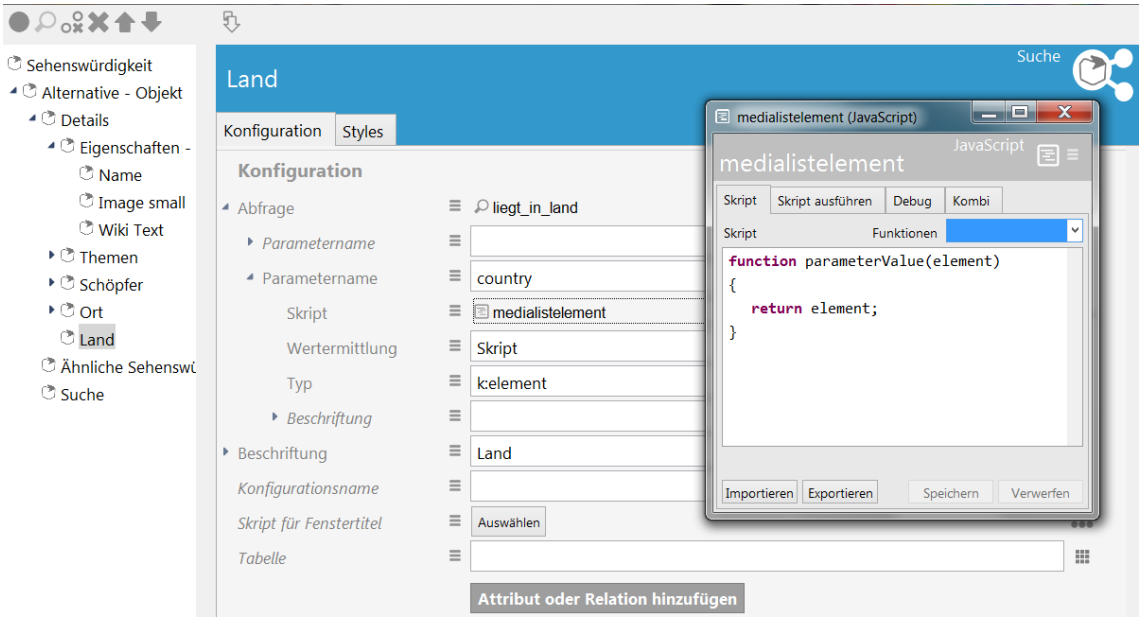
1.7.1.1.5 Indirekte Eigenschaften konfigurieren

Wenn wir wissen, in welcher Stadt sich eine Sehenswürdigkeit befindet, wissen wir auch in welchem Land sie sich befindet, auch wenn das Land nicht direkt mit der Sehenswürdigkeit verbunden ist. Diese indirekte Beziehung zwischen Sehenswürdigkeit und Land können wie auch mit der View-Konfiguration darstellen. Dazu brauchen wir ein Konfigurationselement des Typs "Suche", denn hier können wir eine beliebige Abfrage, also auch die hierfür benötigte Strukturabfrage, definieren:

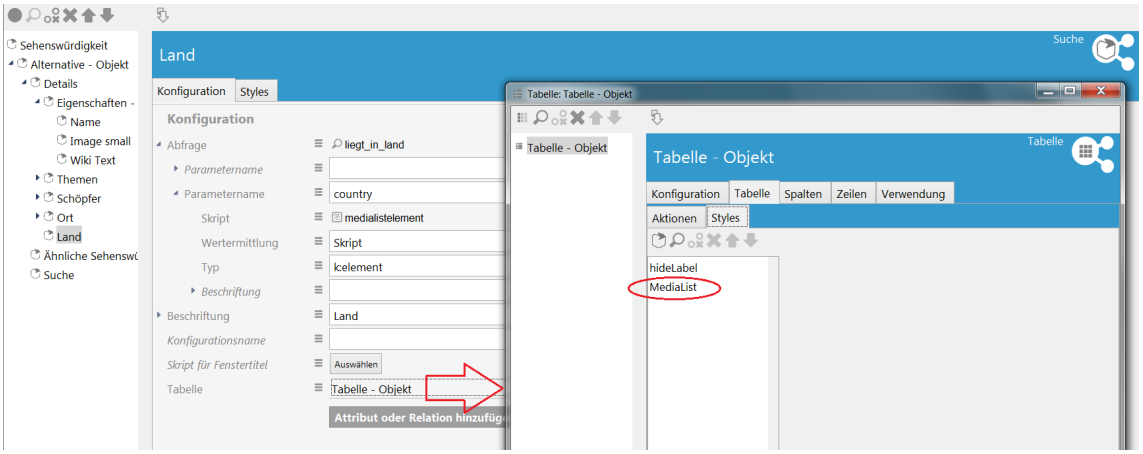


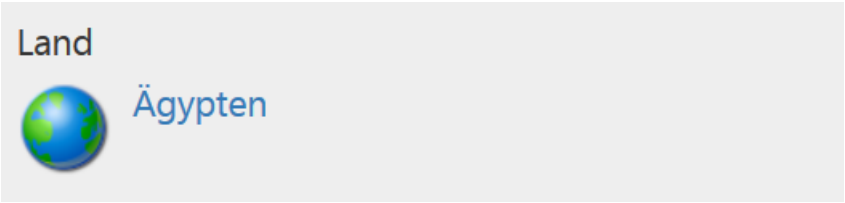
Die Strukturabfrage muss so aufgebaut sein, dass das gewollte Ergebnis, bzw. die gewollten Ergebnisse, Ausgangspunkt der Strukturabfrage sind. Das Objekt Sehenswürdigkeit wird mit einem Parameter versehen.

Der Parameter des Objekts muss ebenfalls in der View-Konfiguration angegeben werden. Ob der Wert über ein Skript oder über eine Benutzereingabe ermittelt werden soll, muss unter *Wertermittlung* angegeben werden. Ein Skript, das das Element des Parameters zurückgibt, wird ebenfalls benötigt. Als Typ muss angegeben werden, dass es sich um ein Element des semantischen Modells handelt (k:element). Ausführlichere Erklärung zu diesen und den weiteren Einstellungsmöglichkeiten sind im Kapitel Suche nachzuschlagen.



Unter Tabelle wird definiert, wie (d.h. mit welchen Eigenschaften) das Ergebnis, bzw. die Ergebnisse der Suche dargestellt werden sollen. Hier können ebenfalls Style-Elemente angegeben werden. Wir verwenden das Style-Element *MediaList*, welches die Ergebnisse mit Namen und dem Icon ihres Objekttyps darstellt. Alle Details zur Konfiguration einer Ergebnistabelle sind im Kapitel Tabelle nachzuschlagen.



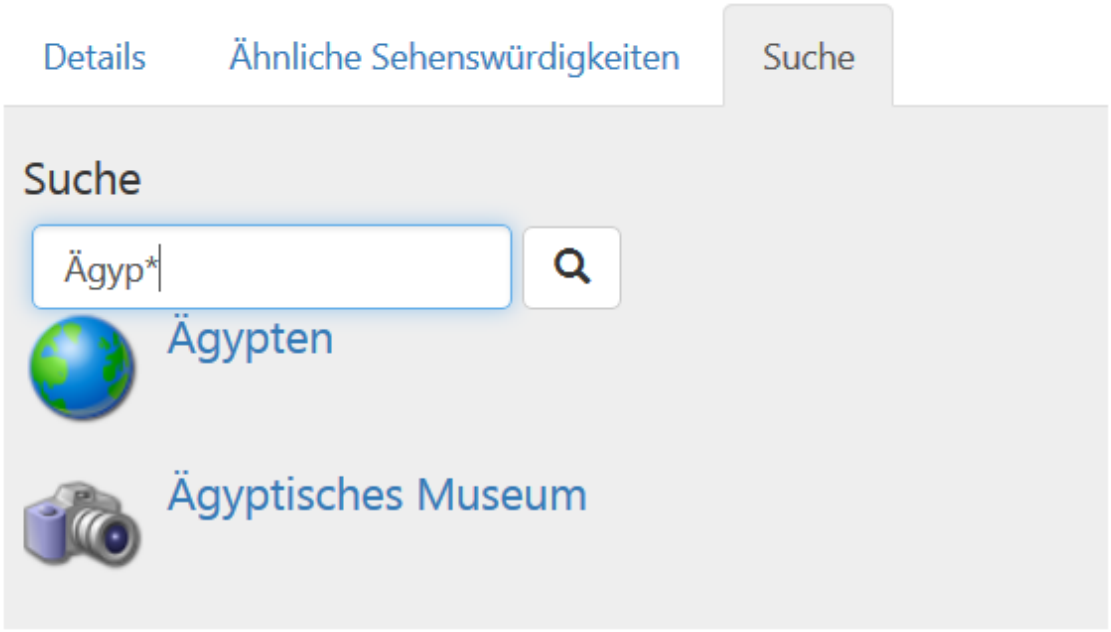


Das Style-Element MediaList ermöglicht eine alternative Darstellung von Ergebnissen einer Abfrage

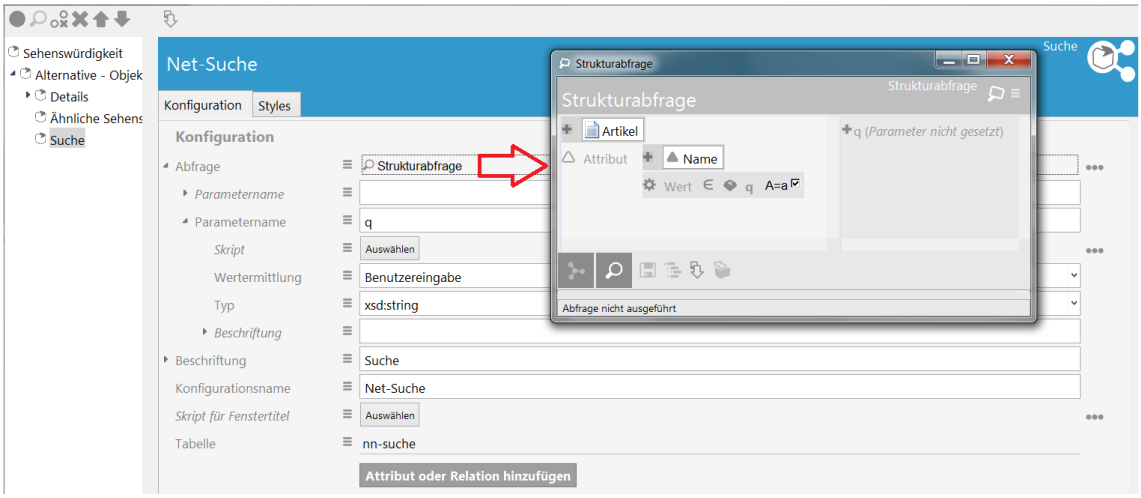
Prinzipiell funktioniert die Einstellung der Konfiguration von ähnlichen Sehenswürdigkeiten genau wie die Konfiguration der indirekten Eigenschaft "Land", mit dem einzigen Unterschied, dass hier eine Such-Pipeline anstelle einer Strukturabfrage verwendet wird, um beispielsweise die fünf ähnlichsten Sehenswürdigkeiten auf Grundlage von Themen, Schöpfer und Ort zu berechnen.

1.7.1.1.6 Suchen konfigurieren

Der letzte Reiter unserer Anwendung soll eine Suche beinhalten.



Den Reiter "Suche" haben wir in unserem Konfigurationsbaum bereits als Konfigurationselement vom Typ "Suche" definiert. Hier legen wir in einer Strukturabfrage fest, dass die Namen aller konkreten Objekte der Datenbank mögliche Ergebnisse sein können. Der Wert des Namens wird über den Parameter *q* definiert. Der Parameter *q* soll nicht zwischen Groß- und Kleinschreibung einer Benutzereingabe unterscheiden und wenn der Benutzer ein Wort eines zusammengesetzten Begriffes eingibt (z.B. "Museum"), so sollen alle Objekte, bei denen dieses Wort im Namen vorkommt, in der Ergebnisliste erscheinen (z.B. "Ägyptisches Museum"). Dies wird durch den Operator *Enthält Zeichenkette (strings to words filter (Beschreibungstexte bereinigen))* in der Strukturabfrage festgelegt.

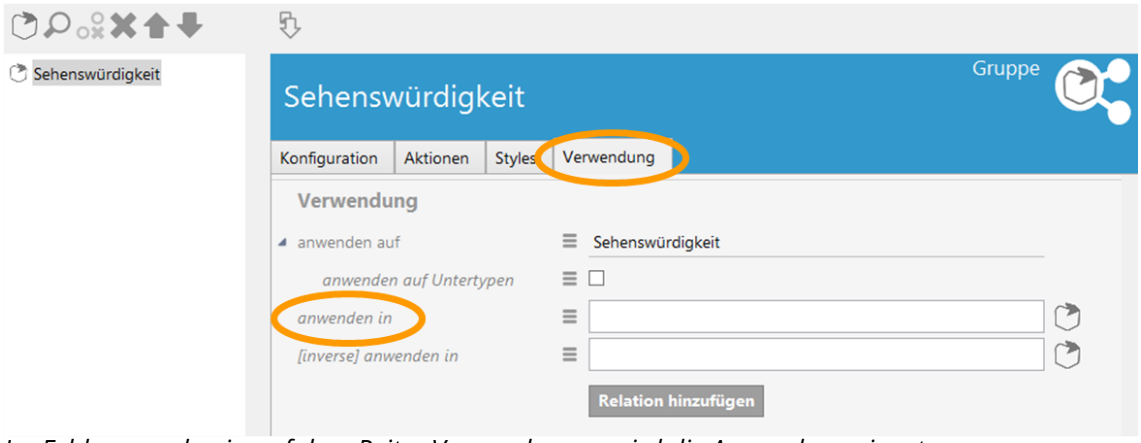


Im Editor der View-Konfiguration des Reiters "Suche" muss der Parameter *q* noch unter *Parametername* angegeben werden. Hier muss zudem definiert werden, dass sich der Wert des Parameters über die Benutzereingabe ermittelt und dass die Benutzereingabe eine beliebige Zeichenkette sein kann (Typ *xsd:string*).

Wie die Ergebnisliste dargestellt werden soll, wird wieder über das Element *Tabelle* konfiguriert. In unserem Fall wird hier eine Tabelle angelegt, die lediglich die Namen der Ergebnisse zeigt und das Style-Element *MediaList* verwendet.

1.7.1.2 Die Verwendung von View-Konfigurationen

Auf dem Reiter *Verwendung* einer View-Konfiguration muss angegeben werden, in welcher Anwendung und für welchen Objekttyp die View-Konfiguration angezeigt werden soll.



Im Feld anwenden in auf dem Reiter Verwendungen wird die Anwendung eingetragen.

Ist keine Anwendung als Verwendung der View-Konfiguration eingetragen, so wird die View-Konfiguration nicht angezeigt. View-Konfigurationen werden als Baumstruktur definiert, in der das Prinzip der Vererbung gilt. Aus diesem Grund muss die Anwendung bei Unterkonfigurationen nicht extra angegeben werden. Sie werden als Teil der Oberkonfiguration mit angezeigt. Zum Beispiel wird eine Eigenschaftskonfiguration angezeigt, wenn diese Teil einer Gruppe ist, deren Verwendung angegeben wurde.

Anwendungen der View-Konfiguration festlegen

Die folgenden Anwendungen stehen von Anfang an zur Verfügung:

- **Graph-Editor:** Die Konfigurationen haben Einfluss auf die Darstellung im Graph-Editor. Der Graph-Editor dient zur Visualisierung der semantischen Elemente und deren Zusammenhängen.
- **Knowledge-Builder:** Die View-Konfigurationen werden im Knowledge-Builder selbst angewendet. Hier stehen neben den Detailkonfigurationen auch die Objektlisten-Konfigurationen zur Verfügung.
- **Knowledge-Portal:** Das Knowledge-Portal ist eine Komponente von i-views, die als Frontend eingesetzt werden kann. Es stellt die Objekte des semantischen Netzes auf Detailseiten und in Kontextboxen basierend auf deren semantischen Kontext dar.
- **Net-Navigator:** Er dient zur Visualisierung von semantischen Elementen. Er kann im Gegensatz zum Graph-Editor der Teil des Knowledge-Builders ist, in den Anwendungen Knowledge-Portal und Viewkonfigurations-Mapper eingesetzt werden.
- **Topic-Chooser:** Er ermöglicht die Auswahl von Relationszielen in einem Fenster.
- **Viewkonfiguration-Mapper:** Der Viewkonfigurations-Mapper ist ein intelligentes Frontend, das im Gegensatz zum Knowledge-Portal die View-Konfigurationen verwendet. Mit ihm können einfach und schnell Sichten auf die Daten erstellt werden.

Darüber hinaus können auch eigene beliebige Anwendungen definiert werden, die an dieser Stelle mit der View-Konfiguration verknüpft werden können.

Objekttyp der View-Konfiguration festlegen

Im Feld *anwenden auf* wird der Objekttyp automatisch eingetragen, wenn die View-Konfiguration am Objekttyp definiert wird. Hier können nach Bedarf weitere Objekttypen angegeben werden. Je nach dem wo man die View-Konfiguration angelegt hat, entweder unter *Objekt* oder unter *Typ*, wird die View für Objekte des ausgewählten Typs oder den Typ einschließlich seiner Untertypen angezeigt. Durch das Setzen oder Wegnehmen des Häkchens des Attributs *anwenden auf Untertypen* kann diese Auswahl (gültig für Objekte oder Typen) auch nach der Definition der View-Konfiguration geändert werden.

Wiederverwendung von View-Konfigurationen

View-Konfigurationen können in anderen View-Konfigurationen wieder verwendet werden. So kann man beispielsweise eine Tabelle konfigurieren, die für die Anzeige von mehreren View-Konfigurationen vom Typ Suche eingesetzt wird. Falls die View-Konfiguration Baustein einer anderen View-Konfiguration ist, wird dies unter *[inverse] anwenden in* angezeigt.

1.7.1.3 Die Gültigkeit von View-Konfigurationen

Im Kapitel Die Verwendung von View-Konfigurationen wurde bereits beschrieben, dass die Angabe über die Verwendung von View-Konfigurationen ausschlaggebend ist ob, in welcher Anwendung und für welche Objekte bzw. Typen die View angezeigt wird. Trotzdem ist es möglich, dass die View-Konfiguration nicht in der ausgewählten Anwendung angezeigt wird. Hier stellt sich die Frage: Wann ist eine View-Konfiguration gültig? Und für welche Objekt bzw. Typen ist die View-Konfiguration gültig?

Vererbung von View-Konfigurationen

View-Konfigurationen verhalten sich in Bezug auf die Vererbung wie Eigenschaften. View-Konfigurationen werden auf die Untertypen bzw. die Objekte der Untertypen vererbt.

Anwendung der konkretesten View-Konfiguration

Die Untertypen verwenden nach dem Prinzip der Vererbung die View-Konfiguration der

Obertypen solange sie keine eigenen View-Konfigurationen besitzen. Es wird immer die konkreteste View-Konfiguration angewendet: Das ist die Konfiguration, die direkt am Typ definiert ist. Ist das nicht der Fall, so wird geprüft ob es am Obertyp eine View-Konfiguration gibt. Ist das ebenfalls nicht der Fall so wird in der Typenhierarchie jeweils eine Ebene nach oben gegangen und geprüft ob eine View-Konfiguration definiert ist. Es wird dann diejenige View-Konfiguration angewendet, die dem Objekttyp am nächsten steht. Wird keine View-Konfiguration an den Obertypen gefunden, wird für Administratoren die Default-Konfiguration verwendet.

Was passiert wenn zwei gleichwertige View-Konfigurationen existieren?

Gibt es zwei gleichwertige View-Konfigurationen, so wird keine View-Konfiguration angezeigt. Wurde bei einer View-Konfiguration die Anwendung oder der Objekttyp nicht definiert, zählt diese nicht zu den aktiven View-Konfigurationen. In diesem Fall wird die andere View-Konfiguration verwendet. Möchte man für unterschiedliche Benutzer jeweils andere Views anzeigen, kann im Detektorsystem eine Regel definiert werden. In diesem Fall wird dann die View-Konfiguration entsprechend der definierten Regel angewendet, solange die Regel abhängig von Nutzer nur eine gültige View-Konfiguration liefert.

1.7.2 Arten der View-Konfiguration

1.7.2.1 Detailkonfiguration für Objekte oder Typen

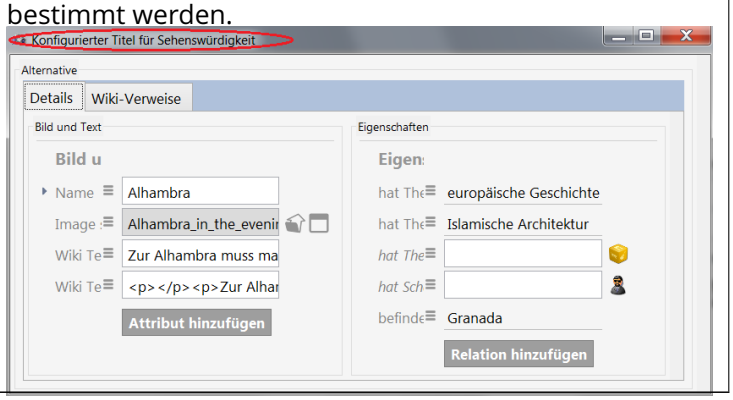
Eine Detailkonfiguration beschreibt, wie Objekte oder Typen dargestellt werden sollen. Im folgenden werden die verschiedenen Detailkonfigurationsarten, die der View-Konfiguration zur Verfügung stehen, beschrieben.

Die einzelnen Detailkonfigurationen lassen sich teilweise beliebig zusammenstecken. Ebenfalls können die Konfigurationen mehrfach als Unterkonfiguration verwendet werden.

Liste der verschiedenen Detailkonfigurationstypen

Konfigurationstyp	Top-Level-Konfiguration	Kann folgende Unterkonfiguration enthalten
Alternative	x	beliebig
Eigenschaft		
Eigenschaften	x	Eigenschaft
Gruppe	x	beliebig
Hierarchie	x	beliebig
Skriptgenerierter Inhalt	x	
Statischer Text		
Suche		Tabelle

Einstellungsmöglichkeiten, die alle Detailkonfigurationstypen gemeinsam haben

Name	Wert
Konfigurationsname	Findet keine Verwendung im Userinterface. Der Ersteller einer Konfiguration hat hier die Möglichkeit einen für ihn verständlichen Namen zu vergeben, um diese Konfiguration später besser wiederfinden und in anderen Konfigurationen wieder verwenden zu können.
Skript für Fenstertitel	Nur zur Verwendung im Knowledge-Builder. Öffnet man ein Objekt beispielsweise per Doppelklick in der Objektliste, öffnet sich ein Fenster mit den Eigenschaften dieses Objektes. Der Titel dieses Fensters kann durch ein Skript bestimmt werden. 

Anmerkung: In den folgenden Abschnitten werden die Einstellungsmöglichkeiten für die einzelnen Konfigurationstypen beschrieben. Die obligatorischen Parameter sind fett gedruckt.

1.7.2.1.1 Alternative

Eine Alternative verwenden wir, um beliebig viele verschiedene alternative Ansichten auf ein Objekt zu konfigurieren. Die alternativen Ansichten können in der Anwendung mittels Reitern ausgewählt werden. Zur Beschreibung, wie man eine Alternative anlegt, siehe Kapitel Reiter konfigurieren.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. einer weiteren *Alternative* eingebettet ist.
hat Default-Alternative	Die Unteransicht, die initial selektiert sein soll, kann hier festgelegt werden.

Darstellung in einer Anwendung

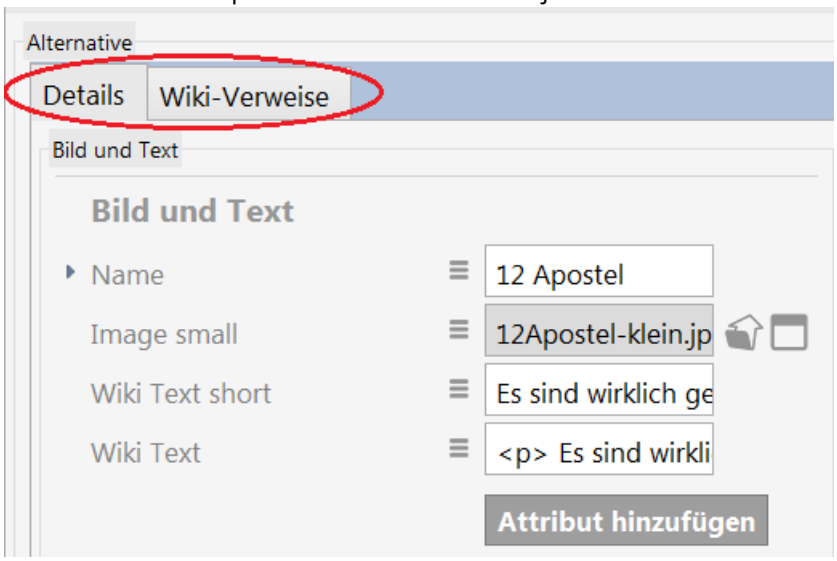
Werden die Ansichten in JSON herausgeschrieben, werden die einzelnen Unteransichten in

einem ARRAY an den KEY *alternatives* gehängt.

Beispiel einer Alternative in einer Anwendung: Mittels der Reiter kann zwischen den Ansichten "Details", "Ähnliche Sehenswürdigkeiten" und "Suche" gewechselt werden.

Darstellung im Knowledge-Builder

Im Knowledge-Builder werden dem Anwender die verschieden konfigurierten und mit der Alternative verknüpften Ansichten eines Objekts durch Reiter zur Verfügung gestellt.



Beispiel einer Alternative im Knowledge-Builder: Mittels der Reiter kann zwischen der Ansicht "Details" und der Ansicht "Wiki-Verweise" gewechselt werden.

Anmerkung

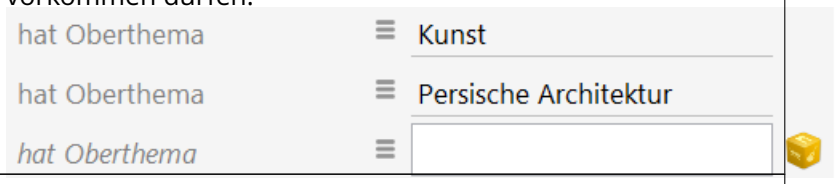
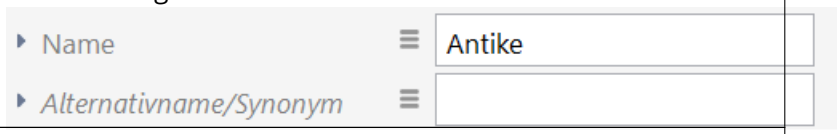
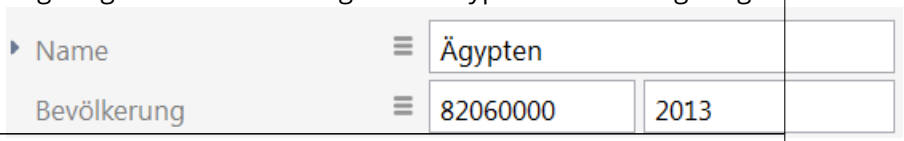
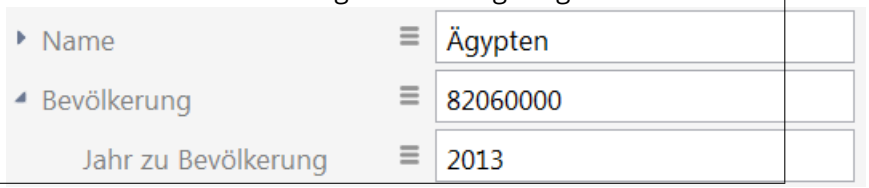
Diese Konfiguration ist ähnlich der Konfiguration *Gruppe*. Allerdings werden bei einer *Gruppe* alle Unteransichten gleichzeitig angezeigt.

1.7.2.1.2 Eigenschaft

Mit der View-Konfiguration *Eigenschaft* kann definiert werden, wenn beispielsweise eine abstrakte Eigenschaft benutzt wird, die eine Menge von Eigenschaften zusammenfasst, angezeigt werden soll.

Einstellungsmöglichkeiten

Name	Wert
Absteigend sortieren	Steuert, ob die Eigenschaften auf- oder absteigend nach ihrem Namen sortiert werden. Ist dieser Parameter nicht gesetzt, wird *aufsteigend* sortiert.
Anzeigeart	Steht in zwei Fällen zur Verfügung: **1.** Die Eigenschaft ist eine Relation: Auswahlmöglichkeit für die Darstellung der Beschriftung eines Relationsziels. Diese Einstellungsmöglichkeit steht nur dann zur Verfügung, wenn die Einstellung *Relationszielansicht* den Wert *Auswahl* oder *Relationsstruktur hat*. **2.** Die Eigenschaft ist ein Datei-Attribut: Auswahlmöglichkeit für die Darstellung des Wertes eines Datei-Attributs. Auswahlmöglichkeiten: • *Symbol (topic*Icon): Icon des Relationsziels/die Datei als Icon • *Symbol und Zeichenkette* • *Zeichenkette (Namensattribut)*: Name des Relationsziels/Name der Datei
Beschriftung	Anzeigename der Eigenschaft. Ist keine Beschriftung angegeben wird der Name des Eigenschaftstyps ausgegeben.
Eigenschaft	Verknüpfung zu einem Eigenschaftstyp, der angezeigt werden soll.
Skript für virtuelle Eigenschaften	Alternativ zu 'Eigenschaft': Skript zur Berechnung anzuzeigender Werte
Einblendung des Relationsziels	Standardmäßig wird lediglich der Name des Relationsziels angezeigt. Wird der Name angeklickt, öffnet sich das Relationsziel in einem weiteren Editor. Wird hingegen die Option *Einblendung des Relationsziels* gewählt, werden die Relationsziele direkt angezeigt, d.h. nicht nur ihr Name, sondern alle ihre Eigenschaften.

Einblendung zusätzlicher Eigenschaften	Nur bei der Ansicht zum Bearbeiten von Objekten relevant: Ist diese Option gesetzt, wird zu der Eigenschaft eine weitere Eigenschaft eingeblendet, sodass diese bequem und schnell ausgefüllt werden kann. Die Eigenschaft muss mehrfach vorkommen dürfen. 
Einblendung zusätzlicher neuer Eigenschaften	Nur bei der Ansicht zum Bearbeiten von Objekten relevant: Ist diese Option gesetzt, wird die Eigenschaft nur dann eingeblendet, wenn die Eigenschaft noch nicht angelegt wurde. So kann sie bequem und schnell ausgefüllt werden und wird nicht so leicht vergessen. 
Einblendungsfilter	Nur bei der Ansicht zum Bearbeiten von Objekten relevant: Mit dieser Option kann eine Verknüpfung zur einer Abfrage, die entscheidet, ob diese Konfiguration angezeigt wird, angelegt werden. Die Abfrage wird mit dem Objekt dieser Eigenschaft gefüllt. Nur, wenn die Abfrage ein Ergebnis enthält, wird die Eigenschaft zum bearbeiten angezeigt.
Konfiguration für eingebettete Metaeigenschaften	Angabe der Konfiguration, die verwendet werden soll, um Metaeigenschaften anzuzeigen. Die Metaeigenschaften werden eingebettet, d.h. hinter dem Wert der Eigenschaft angezeigt. Der Name des Eigenschafttyps wird nicht angezeigt. 
Konfiguration für Metaeigenschaften	Angabe der Konfiguration, die verwendet werden soll, um Metaeigenschaften anzuzeigen. Die Metaeigenschaften werden unter dem Wert der Eigenschaft angezeigt. 

Relationszielansicht	Wird als Eigenschaft eine Relation gewählt, kann mit diesem Parameter die Ansicht der Relationsziele definiert werden: • *Auswahl*: Alle Relationsziele werden aufgelistet und mit einer vorangestellten Checkbox angezeigt. Bei bestehenden Relationen ist die CheckBox mit einem Häkchen versehen. • *Drop Down*: Diese Einstellung ist nur sinnvoll, wenn die Relation nur einmal vorkommen darf. Es wird eine Drop Down-Liste mit allen möglichen Relationszielen zur Auswahl angezeigt. • *Relationsstruktur*: Alle Relationsziele werden im linken Bereich aufgelistet, ähnlich einer Hierarchie, angezeigt. Im rechten Bereich ist dann die Detailansicht des selektierten Relationsziels zu sehen. Diese Ansicht kommt nur zur Geltung, wenn die Konfiguration direkt einer Top-Level-Konfiguration untergeordnet ist. • *Tabelle*: Tabellenansicht der Relationen. Die Tabellenansicht kann nicht im Knowledge-Builder angewendet werden. Für die Tabellenansicht muss die Einstellung *Tabelle* ausgefüllt werden. • *Tabelle (Relationsziele)*: Tabellenansicht der Relationsziele. Diese Tabelle kann im Knowledge-Builder angewendet werden.
Skript für Sortierung	Anhand des Skriptes wird ein Wert, nach dem sortiert wird, ermittelt. Siehe Beispiel unten.
Suche zur Zielauswahl	Die Suche bestimmt, welche Wissensnetzelemente als mögliche Relationsziele angeboten werden. Zur Konfiguration der Suche, siehe auch Kapitel Suchen/Abfragen.
Tabelle	Steht nur zur Verfügung, wenn *Relationszielansicht* den Wert *Tabelle*, bzw. *Tabelle (Relationsziele)* hat und ist dann obligatorisch. Die hier angegebene Tabellenkonfiguration gibt an, welche Eigenschaften der Relationsziele tabellarisch ausgegeben werden sollen. Um das Relationsziel anzuzeigen muss mindestens sein Name in der Tabelle konfiguriert werden. Zur Konfiguration einer Tabelle, siehe Kapitel Tabelle.

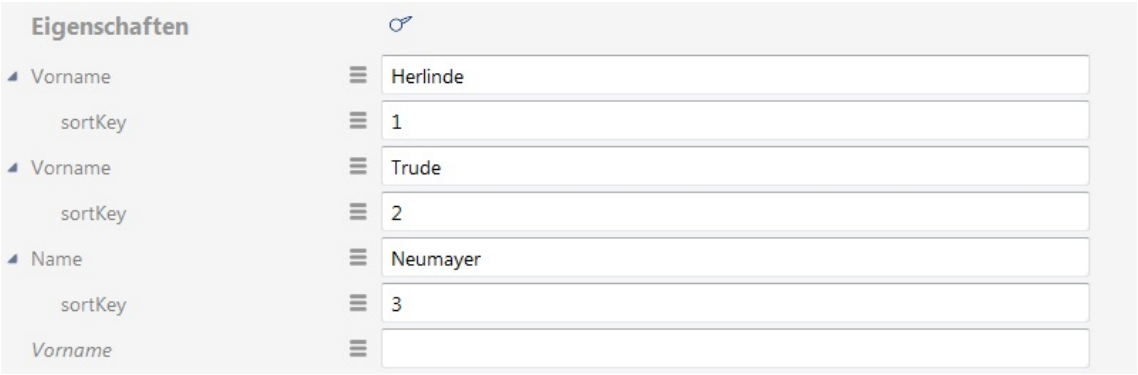
Beispiel für *Skript für Sortierung*

Vorbedingung: An allen Eigenschaften kann das Attribut *sortKey* angebracht werden.

```
function sortKey(element)
{
    if (element instanceof $k.Property)
    {
        var attribute = element.attribute('sortKey')
        if (attribute)
        {
```

```
            return attribute.value();
        };
    };
    if (element instanceof $k.Domain)
    {
        var attribute = element.type().attribute('sortKey');
        if (attribute)
    {
            return attribute.value();
        };
    };
    return undefined;
}
```

Eigenschaftsanzeige einer Person



Anmerkung: Beim Attributtyp *Vorname* ist für *sortKey* der Wert 20 eingetragen, daher steht dieser vorläufige Wert am Ende der Liste.

1.7.2.1.3 Eigenschaften

Die Konfiguration *Eigenschaften* ist im Prinzip eine Liste von einzelnen Eigenschaften. Die Unterkonfigurationen können ausschließlich vom Typ *Eigenschaft* sein.

Einstellungsmöglichkeiten

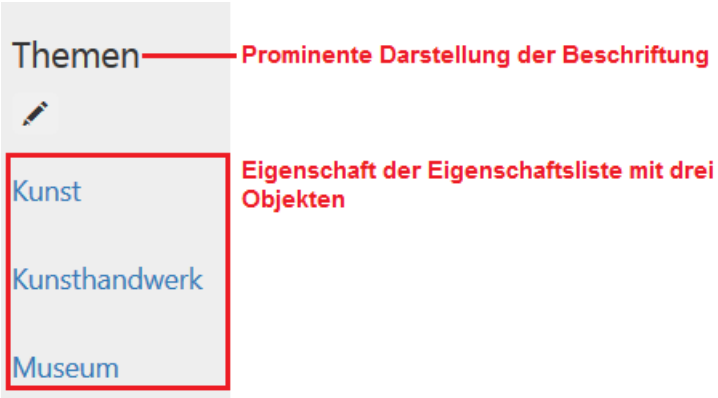
Name	Wert
Beschriftung	Anzeigename der Sammlung von Eigenschaften. Ist keine Beschriftung angegeben wird im Knowledge-Builder die Zeichenkette 'Eigenschaften' verwendet.
Initial ausgeklappt	Ist diese Konfiguration z.B. als Metakonfiguration eingehängt, kann mit diesem Parameter bestimmt werden, ob diese beim Öffnen des Knowledge-Builder-Editors bereits ausgeklappt sein soll. Achtung: Das Front-End stellt die betroffene Metaeigenschaft gar nicht dar, wenn der Haken hier nicht gesetzt ist.

Einstellungsmöglichkeiten für die Sortierung

Name	Wert
Absteigend sortieren	Steuert, ob auf- oder absteigend sortiert wird. Ist dieser Parameter nicht gesetzt, wird aufsteigend sortiert.
Primäres Sortierkriterium	Auswahlmöglichkeit für das Kriterium, nach dem die Eigenschaften sortiert werden: • *Position*: Die in der Konfiguration festgelegte Reihenfolge wird verwendet (Default). • *Wert*: Inhalt des Attributes bzw. Anzeigename des Relationszieles wird verwendet. • *Skript zur Sortierung*: Das in dem Attribut Skript zur Sortierung hinterlegte Skript wird zur Ermittlung des Sortierkriteriums verwendet.
Sekundäres Sortierkriterium	Sortierkriterium für Eigenschaften, die für das primäre Sortierkriterium den gleichen Wert haben. Einstellmöglichkeiten analog zum *Primären Sortierkriterium*
Skript zur Sortierung	Verweis auf ein registriertes Skript, das den Sortierschlüssel für das primäre bzw. sekundäre Sortierkriterium zurückgibt.

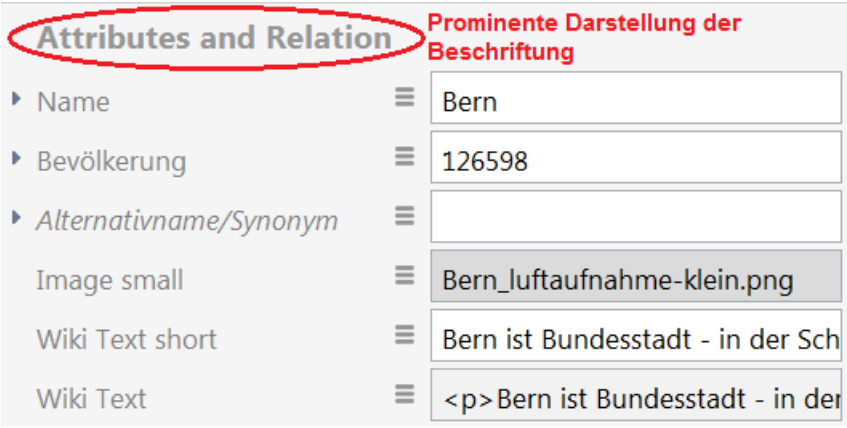
Darstellung in Anwendungen

Die Ansichten der Unterkonfiguration einzelner Eigenschaften werden in einem ARRAY abgelegt und mit dem KEY *properties* angehängt.



Darstellung im Knowledge-Builder

Die in der Konfiguration eingestellte Beschriftung wird prominent angezeigt. Ihm folgen die Ansichten der konfigurierten Eigenschaften.



Anmerkung

Meta-Eigenschaften werden mit dem gleichen Vorgehen angehängt.

1.7.2.1.4 Gruppe

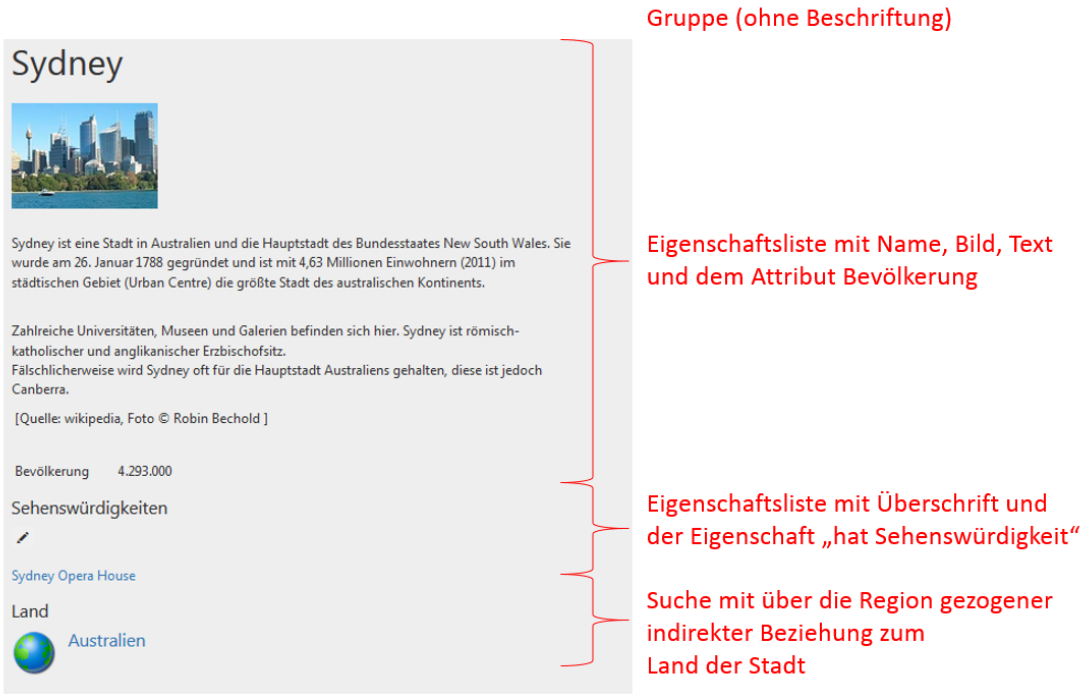
Mit Hilfe der Gruppe lassen sich verschiedenste Unterkonfigurationen in einer Ansicht zusammenfassen. Ausnahme, die nur für Front-End gilt: Das Element *Eigenschaft* kann keine direkte Unterkonfiguration von Gruppe sein. Hierfür braucht es zunächst die Konfiguration *Eigenschaften*.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.

Darstellung in einer Anwendung

Wird die Ansicht in JSON herausgeschrieben, werden die einzelnen Unteransichten in einem ARRAY an den KEY *group* angehängt.



Darstellung im Knowledge-Builder

Im Knowledge-Builder wird um eine Gruppe ein Rahmen gezeichnet. Die Ansichten der Unterkonfigurationen werden dann in diesem Rahmen angezeigt.

Eine Gruppe mit folgenden Unterkonfigurationen: der Eigenschaftsliste "Bild und Text", der Eigenschaftsliste "Eigenschaften" und der Suche "Ähnliche Sehenswürdigkeiten"

1.7.2.1.5 Hierarchie

Der Konfigurationstyp "Hierarchie" stellt Elemente eines semantischen Modells hierarchisch dar.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.

Skript für Beschriftung	Es ist auch möglich durch ein Skript eine Beschriftung festzulegen.
Banner der Hierarchiewurzel anzeigen	Betrifft nur den Knowledge-Builder: Banner wird angezeigt.
Detailansicht ausblenden	Standardmäßig wird die Detailansicht eines ausgewählten Objektes angezeigt (Knowledge-Builder) oder ausgegeben (json, als *subview*). Durch Aktivieren dieser Option wird keine Detailansicht angezeigt bzw. ausgegeben.
Subkonfiguration	Wird eine Detailansicht angezeigt, muss die View-Konfiguration für diese Detailansicht hier angegeben werden.
Unterelemente erzeugen ohne Frage nach Namen	Wenn neue Unterelemente in der Hierarchie erzeugt werden, wird standardmäßig gefragt, wie ihr Name lauten soll. Ein Häkchen hier, erzeugt ohne Frage nach Namen namenlose Objekte.
Verbiete manuelles Sortieren	Standardmäßig kann der Anwender im Knowledge Builder Elemente dem Schema entsprechend durch Drag&Drop umhängen. Wird diese Option aktiviert, ist dies nicht mehr möglich.

Einstellungsmöglichkeiten für die Sortierung

Name	**Wert**
Absteigend sortieren	Steuert, ob auf- oder absteigend sortiert wird. Ist dieser Parameter nicht gesetzt, wird aufsteigend sortiert.
Primäres Sortierkriterium	Auswahlmöglichkeit für das Kriterium, nach dem die Eigenschaften sortiert werden: • *Position*: Die in der Konfiguration festgelegte Reihenfolge wird verwendet (Default). • *Wert*: Inhalt des Attributes bzw. Anzeigename des Relationszieles wird verwendet. • *Skript für Sortierung*: Das in dem Attribut Skript zur Sortierung hinterlegte Skript wird zur Ermittlung des Sortierkriteriums verwendet.
Sekundäres Sortierkriterium	Sortierkriterium für Eigenschaften, die für das primäre Sortierkriterium den gleichen Wert haben. Einstellmöglichkeiten analog zum *Primären Sortierkriterium.*
Skript für Sortierung	Verweis auf ein registriertes Skript, das den Sortierschlüssel für das primäre bzw. sekundäre Sortierkriterium zurückgibt.

Ermittlungsmöglichkeiten der hierarchiebildenden Elemente

Name	Ermittlung von ...
Relation (absteigend)	Unterelementen
Relation (aufsteigend)	Oberelementen
Strukturabfrage (absteigend)	Unterelementen
Strukturabfrage (aufsteigend)	Oberelementen
Skript (absteigend)	Unterelementen
Skript (aufsteigend)	Oberelementen

Aktionen und Styles

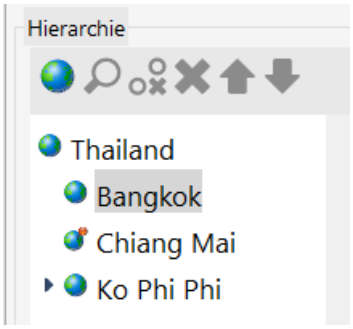
Es lassen sich sowohl für die gesamte Hierarchie als auch für die einzelnen Knoten Aktionen und Styles anbringen.

Darstellung in einer Anwendung

Erst ab Version 4.1 gibt es die JSON-Repräsentation einer Konfiguration vom Typ *Hierarchie*.

Darstellung im Knowledge-Builder

In der Detail-Anzeige eines Elements wird im linken Bereich eine Hierarchie eingeblendet. Im rechten Bereich wird das Element mit einer View-Konfiguration ohne Hierarchie angezeigt. Diese View-Konfiguration muss eigens definiert werden und unter *Verwendung* >> *anwenden in* muss der Konfigurationsname der Hierarchie angegeben werden. Die Subkonfiguration lässt sich alternativ auch direkt an der Hierarchie unter *Subkonfiguration* angeben.



Anmerkung

- Die Werte aller Eigenschaften, die für die Hierarchiebildung ausgefüllt werden können, sind Relationen.
- Die einzelnen Attribute wie z.B. *Relation - absteigend* können mehrfach vergeben werden.
- Für jeden Attributtyp werden die Relation oder Relationen ermittelt und aufgesammelt. Sind verschiedene Attributtypen angegeben, wird mit den Teilmengen eine Schnittmenge gebildet.

Beispiel - Anwendungsfall

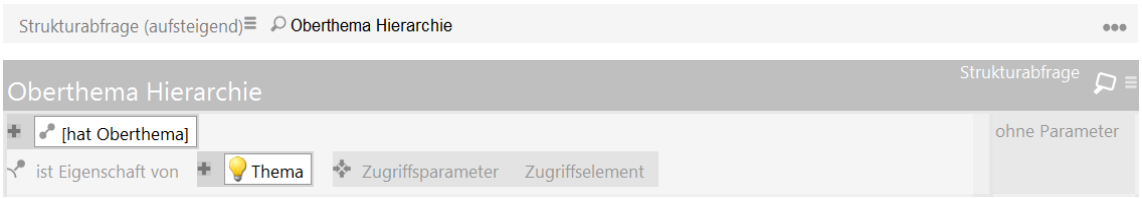
Typischerweise werden Hierarchien verwendet, um Ober-/Unterthema-Relationen oder Teil-von-Relationen darzustellen.

1. Hierarchiebildende Relation

Die direkteste Variante. Die Relationen, die die Hierarchie bilden, werden eingetragen.

2. Hierarchiebildende Strukturabfrage

Die Relationen lassen sich ebenfalls über eine Strukturabfrage ermitteln.



3. Hierarchiebildendes Skript

Auch durch ein Skript lassen sich die möglichen hierarchiebildenden Relationen aufsammeln.

Skript *hat Oberthema*

```
function relationsOf(element)
{
    return element.relations('hatOberthema');
}
```

1.7.2.1.6 Skriptgenerierter Inhalt

Eine durch ein Skript generierte HTML-Beschreibung wird an dieser Stelle angezeigt.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Skript	Zur Generierung einer HTML-Beschreibung

Beispiel

Skript

```
function render(topic, document)
{
    var writer = document.xmlWriter();
    writer.startElement("html");
        writer.startElement("body");
            writer.startElement("h2");
                writer.cdata(topic.name());
            writer.endElement();
        writer.endElement();
    writer.endElement();
}
```

Generierter Inhalt

```
<html>
    <body>
        <h2>Hermannn</h2>
    </body>
</html>
```

Ausgabe

Hermann

1.7.2.1.7 Statischer Text

Ein nicht veränderlicher Text wird ausgegeben.

Einstellungsmöglichkeiten

Name	Wert

Beschriftung	Eine Beschriftung wird nur ausgegeben, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.
Text	Text, der ausgegeben werden soll

1.7.2.1.8 Suche

Das View-Konfigurationselement "Suche" kann als Unterkonfiguration einer Alternative oder einer Gruppe ausgewählt werden. Eine beliebige Abfrage ist hier obligatorisch, deren Ergebnisse angezeigt werden. Ein Beispiel, wie man eine solche Suche anlegt wurde bereits in Kapitel Indirekte Eigenschaften konfigurieren gegeben. Auch Suchen zur Benutzereingabe können konfiguriert werden. Ein Beispiel hierfür findet sich in Kapitel Suchen konfigurieren.

Einstellungsmöglichkeiten

Name	Wert
Abfrage	Hier kann die Suche ausgewählt werden, die sofort ausgeführt wird. Das semantische Objekt für welches die View-Konfiguration angezeigt wird, kann als Zugriffselement in der Abfrage verwendet werden.
Beschriftung	Eine Beschriftung findet nur ihre Anwendung, wenn diese Konfiguration in einer anderen Konfiguration wie z.B. *Alternative* eingebettet ist.

Einstellungsmöglichkeiten für eine Abfrage

Die folgenden Parameter werden als Meta-Eigenschaften für eine *Abfrage* gepflegt.

Name	Wert
Parametername	Angabe eines Parameternamens, wie er in der Abfrage verwendet wird.

Einstellungsmöglichkeiten für einen Parameternamen

Die folgenden Parameter werden als Meta-Eigenschaften für einen *Parameternamen* gepflegt:

Name	Wert
Skript	Das Skript *parameterValue* wird zur Ermittlung des Suchwertes für den angegebenen Parameternamen verwendet.

Wertermittlung	Angabe über den Weg der Wertermittlung *Skript*: Der Wert wird aus dem Skript ermittelt und darf nicht von einem Benutzer überschrieben werden. *Skript, überschreibbar*: Das Skript ermittelt Wert. Der Benutzer darf überschreiben. *Benutzereingabe*: Keine Skriptauswertung. Nur Eingabe durch einen Benutzer.
Typ	xsd-Typ
Beschriftung	Beim Herausschreiben nach JSON landet dieser Wert in *label.*

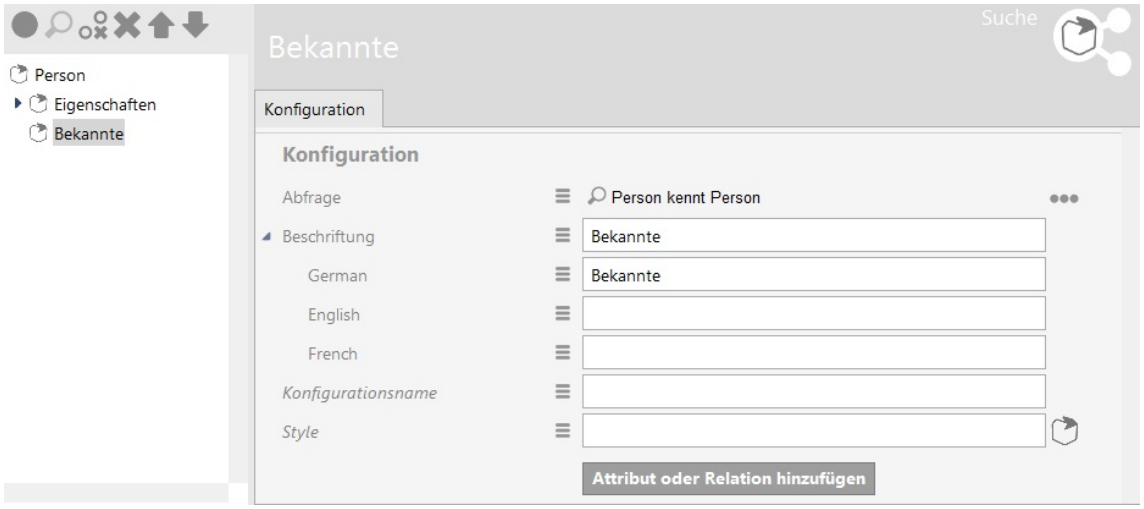
Darstellung in einer Anwendung

Eine Ausgabe für die Suchergebnisse gibt es erst ab Version 4.1 von i-views.

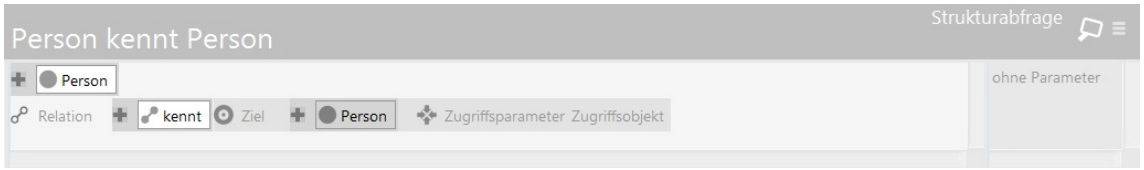
Darstellung im Knowledge-Builder

Die Ergebnisse einer beliebigen Abfrage werden im Knowledge-Builder stets in einer Objektliste angezeigt.

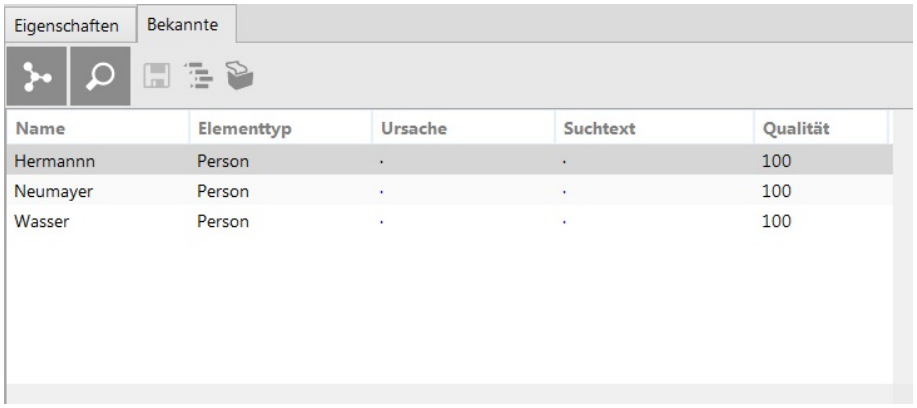
Beispiel:



In der View-Konfiguration werden die Reiter "Eigenschaften" und "Bekannte" definiert. "Bekannte" ist ein Konfigurationselement des Typs "Suche. Unter "Abfrage" kann eine Suche ausgewählt oder direkt neu angelegt werden.



Definition der Suche



Das Ergebnis der Abfrage wird im Reiter "Bekannte" bei Objekten des Typs "Person" im Knowledge-Builder angezeigt.

1.7.2.1.9 Facette

Anmerkungen:

- Dieses Feature steht ab dem Patch-Release 4.3.1 zur Verfügung.
- Es ist ein experimentelles Feature, d.h. in späteren Versionen kann es zu Modelländerungen kommen, die unter Umständen nicht in einem Upgrade-Code beachtet werden und somit von Hand durchgeführt werden müssen.
- Das Einschränken auf einen dynamisch ermittelten Term ist noch nicht implementiert.

Für die Ergebnismenge einer Suche, die in der übergeordneten Suche-Konfiguration angehängt ist, lassen sich Facetten auf Beziehungsziele definieren.

Mit der Aktion *Facette neu anlegen* oder *Facette hinzufügen* wird eine Facette zur der Suche-Konfiguration hinzugefügt.

Es können natürlich mehere Facetten an eine Suche-Konfiguration angebracht werden. Da sich keine Synergieeffekte bei der Facettenberechnung untereinander ergeben, verlängert sich dadurch natürlich der Berechnungsvorgang.

An einer Facetten-Konfiguration stehen folgende Einstellungsmöglichkeiten zur Verfügung.

Einstellungsmöglichkeiten

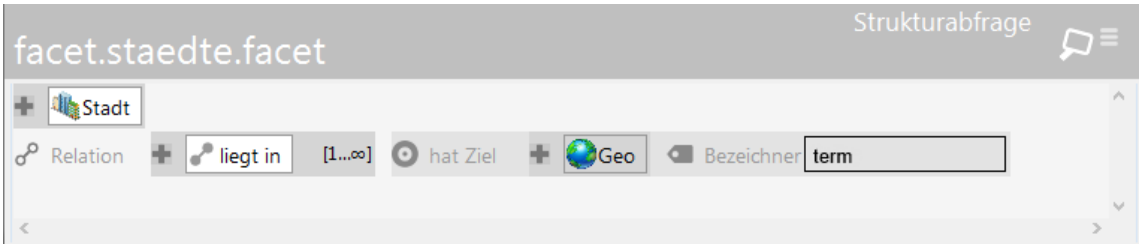
Name	Wert
Beschriftung	Idealerweise ist die Beschriftung angegeben. Ist dieser Wert nicht gesetzt, wird der Name des Eingangsobjektes der *Abfrage* verwendet.
Abfrage	Diese Abfrage ist obligatorisch. Sie wird benötigt um die Terme für eine Facette zu ermitteln. Das Eingangsobjekt ist vom Typ gleich den Suchergebnissen der Suche, die in der Suche-Konfiguration definiert ist. Die zu findenden Terme sind durch den Bezeichner *term* gekennzeichnet.
Abfrage für Ermittlung Elternelement	Soll eine Term-Hierarchie gebildet werden, muss durch diese Abfrage definiert werden, wie sich Eltern-Kind-Elemente finden. Das Eingangsobjekt ist dabei das Kind-Element. Der Bezeichner *parentTerm* kennzeichnet das Eltern-Element. Anmerkung: Alle Terme, die eine Hierarchie bilden sollen, müssen durch die *Abfrage* an der Facetten-Konfiguration gefunden werden.
Absteigend sortieren	Standardmäßig werden die Namen oder die Anzahlen aufsteigend sortiert. Mit diesem Flag lässt sich die Sortierung umdrehen.

Anzahl nicht anzeigen	Dieses Flag muss gesetzt werden, wenn die Anzahl nicht mit angezeigt werden soll.
Kindelemente initial anzeigen	Standardmäßig werden bei einer Hierarchie die Kindelemente erst angezeigt, wenn das dazugehörige Elternelement ausgewählt wurde. Wird dieses Flag gesetzt, werden initial alle Kindelemente mit ausgeliefert.
Leere Terme anzeigen	Standardmäßig werden Terme, die keine Anzahl haben, nicht eingeblendet. Durch setzen dieses Flags werden auch diese angezeigt.
Maximale Anzahl an Termen	Hier kann die Anzahl der Terme festgelegt werden, die angezeigt werden sollen. Standardmäßig werden immer alle Terme dargestellt.
Nach Anzahl sortieren	Standardmäßig werden die Terme nach dem Namen sortiert. Durch das setzen dieses Flags werden die Terme nach der Anzahl der Vorkommnisse in der Treffermenge sortiert.
Termart	Hier stehen folgende Einstellungsmöglichkeiten zur Verfügung: • *Dynamisch*:Die Wertebereiche der Terme werden automatisch ermittelt. Die Werte, die zur Termbildung verwendet werden, müssen mit dem Parameter *termValue* versehen sein. Mit dem Parameter *Maximale Anzahl an Termen* lässt sich festlegen, wie viel Terme ausgebildet werden. • *Statisch*:Es müssen alle Terme, die angezeigt werden sollen, für sich konfiguriert werden. Es muss für jeden Term eine Suche angelegt werden, die die möglichen Treffer in der Hauptsuche beschreibt. Wird keine Termart ausgewählt (Standardverhalten), werden die Terme anhand der Abfrage an der Facetten-Konfiguration ermittelt. In der Abfrage können nur Relationsziele als mögliche Terme ausgebildet werden.

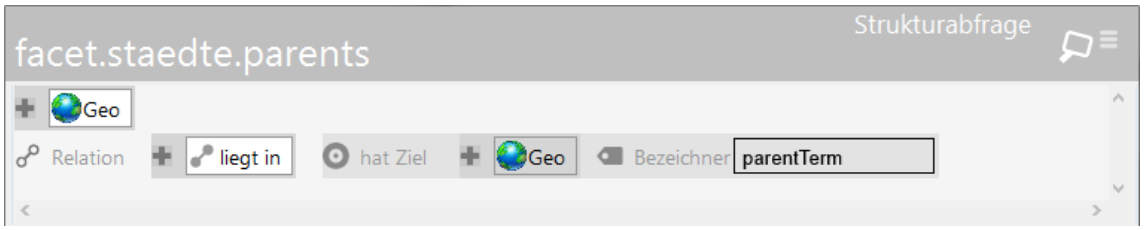
Beispiele:

- Keine Termart gewählt –> Suchen nach Relationsziel

Attribut *Abfrage*



Attribut *Abfrage für Ermittlung Elternelement*



- Termart *Dynamisch* ausgewählt

Attribut *Abfrage*

- Termart *Statisch* ausgewählt

Keine *Abfrage* an der Facetten-Konfiguration

Unter der Facetten-Konfiguration gibt es mehrere Term-Konfigurationen

Attribut *Abfrage* für Term-Konfiguration Hauptfigur

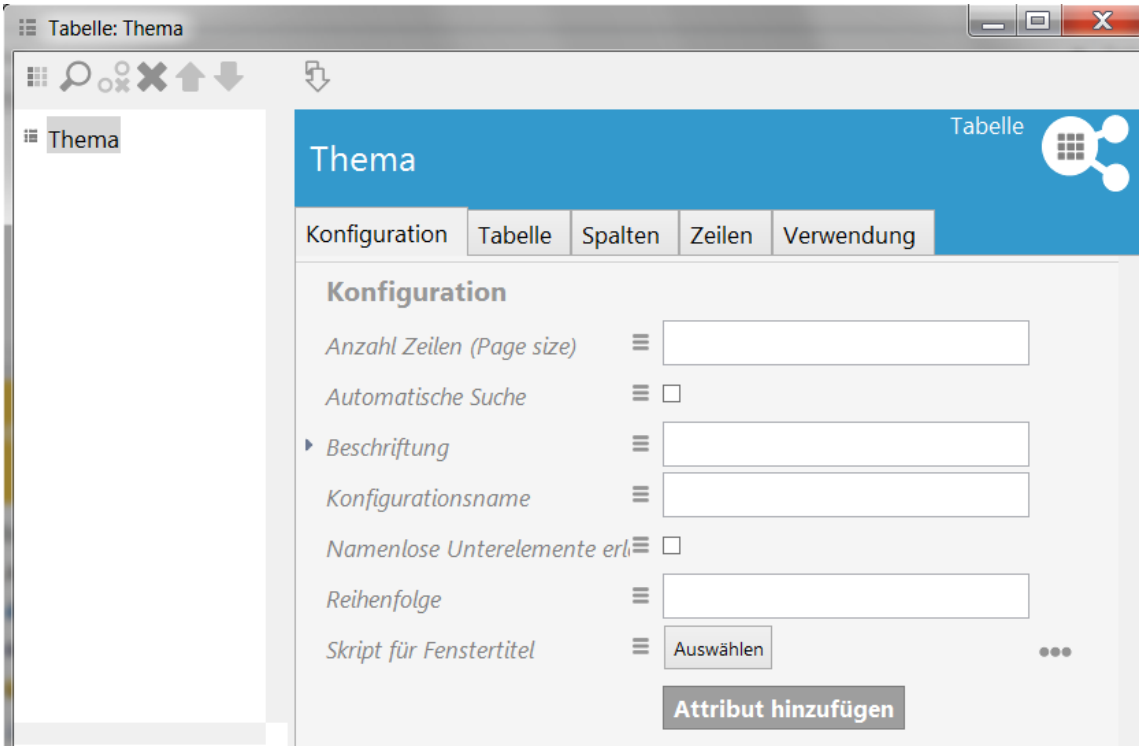
Attribut *Abfrage* für Term-Konfiguration Nebenfigur

1.7.2.2 Tabelle

Tabellen können als Unterkonfiguration für die Ergebnisanzeige von Abfragen des Konfigurationstyps "Suche" oder als eigenständige Konfiguration zur Darstellung der Objektlisten im Knowledge-Builder verwendet werden.

Eine Tabelle listet konkrete Objekte, Eigenschaften oder Untertypen eines bestimmten Typs auf. Ob alle Objekte, Eigenschaften oder Untertypen oder nur eine Auswahl angezeigt werden, lässt sich über die Eingabe in den Spaltenköpfen steuern. Mit den eingegebenen Werten wird eine Strukturabfrage nach passenden Objekten, Eigenschaften oder Untertypen ausgeführt und das Ergebnis tabellarisch dargestellt. Außerdem kann bei Objektlisten nach Eingabe von Werten in die Spaltenköpfe ein neues Objekt, ein neuer Eigenschaftswert oder ein neuer Untertyp mit den ausgefüllten Eigenschaften erzeugt werden.

Bestandteile der Konfiguration *Tabelle* sind *Spaltenkonfigurationen*. Diese wiederum beinhalten *Spaltenelemente*. Diese Aufteilung dient der Trennung von spaltenrelevanten Eigenschaften, wie Reihenfolge und Benennung der Spalte in der Tabelle, und der Zuordnung, welche Inhalte in der Spalte angezeigt werden sollen. *Spaltenelemente* wiederum erlauben die Zuordnung von Eigenschaften, zusätzlich können Skript-Bausteine und Strukturabfrage-Bausteine eingebettet werden.



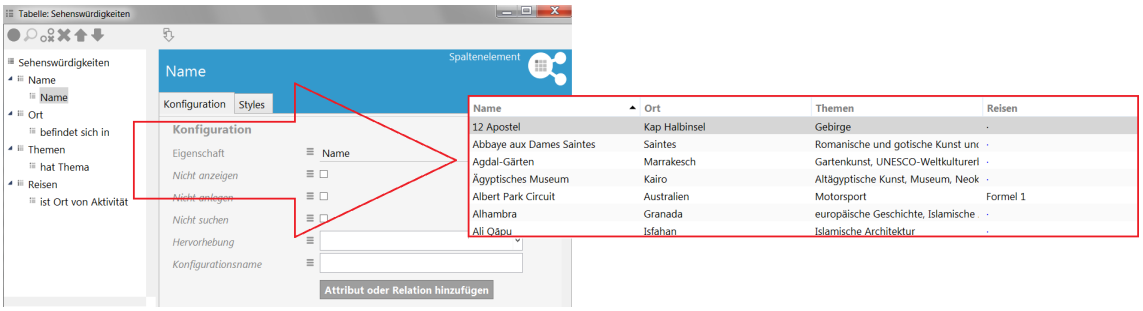
Die hierarchische Darstellung aller Unterkonfigurationselemente der Tabellenkonfiguration weist eine Menü-Zeile auf, die wie folgt mit Aktionen belegt ist:

- Neues Unterelement anlegen und verknüpfen.
- Bereits vorhandene mögliche Unterelement durchsuchen und verknüpfen
- Verknüpfung wieder löschen, Unterelement bleibt dabei als Objekt erhalten. und kann in anderen Konfigurationen wieder verwendet werden.
- Gewähltes Unterelement komplett löschen. Falls es in anderen Konfigurationen verwendet wird, öffnet sich vor der Löschung eine Warnung, die alle vorhandenen Verknüpfungen aufzeigt.

- Gewähltes Unterelement in der Liste nach oben schieben.
- Gewähltes Unterelement in der Liste nach unten schieben.
- Aktualisierung. Erst nach einer Aktualisierung werden alle Änderungen für die Anwendungen übernommen.

Beispiel einer einfachen Tabellenkonfiguration

Zu einer Liste von Sehenswürdigkeiten soll der Ort, an dem sie sich befinden, Themen, mit denen sie zu tun haben und Reisen, bei denen sie besucht werden in einer Tabelle zu sehen sein. Nicht zu vergessen ist das Namensattribut, durch das die Sehenswürdigkeiten in der ersten Spalte repräsentiert werden.



Einstellungsmöglichkeiten (Tabelle)

Name	Wert
Anzahl Zeilen (Page size)	Gibt an, wie viele Zeilen auf einer Seite angezeigt werden sollen. Standardwert: 20
Keine automatische Suche	Es wird keine automatische Suche durchgeführt.
Namenlose Unterelemente erlauben	Namenlose Unterelemente werden ebenfalls angezeigt.
Ohne Sortierung	Es findet keine Sortierung statt. Standardverhalten: es wird nach der ersten Spalte sortiert.
Reihenfolge	Durch Angabe einer Ganzzahl lässt sich steuern an welcher Stelle, falls mehreren Konfigurationen vom Typ *Tabelle* angezeigt werden sollen, die aktuelle Konfiguration angezeigt wird. Die Sortierung wird nach zwei Kriterien durchgeführt, die in der folgenden Reihenfolge überprüft werden: 1. Attribut *Reihenfolge* vorhanden, wenn ja, dann wird dieses als Sortierkriterium verwendet, wenn nein, werden erst die Konfigurationen für Typen und dann die für Objekte angezeigt. 2. Sortierung nach Anzeigename

Aktionen und Styles

Aktionen und Styles lassen sich für die gesamte Tabelle, aber auch für Zeilen festlegen.

Verwendung

Wo die Tabelle zur Verwendung kommt, wird auf dem Reiter *Verwendung* angegeben.

Unter *anwenden auf* wird der Objekttyp angegeben, auf den die Tabelle angewendet werden soll. Tabellen können in anderen View-Konfigurationen wieder verwendet werden. Falls die Tabelle Baustein einer anderen View-Konfiguration ist, wird dies unter *[inverse] anwenden in* angezeigt.

Die Eigenschaft *anwenden in* verweist auf eine Anwendung. Mehrere Verknüpfungen sind möglich.

Beispiele:

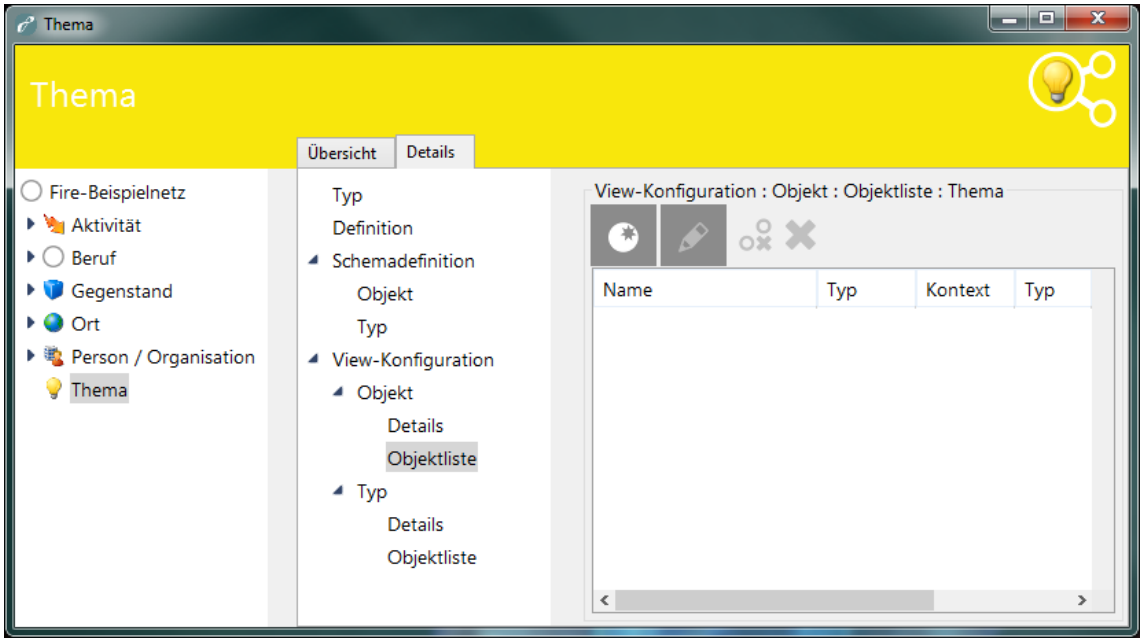
- Soll die Tabelle im Knowledge-Builder rechts im Hauptfenster bei der Navigation durch die Ordnerstruktur verwendet werden, dann muss die Tabellenkonfiguration mit dem entsprechenden Ordnerstrukturelement verknüpft sein.
- Sollen mögliche Relationsziele im Knowledge-Builder tabellarisch dargestellt werden, dann muss die Tabelle mit der Anwendung Knowledge-Builder verknüpft sein.

Tabellen / Objektslisten im Knowledge-Builder

Für die Konfiguration der tabellarische Darstellung von Objekten oder Typen im Knowledge-Builder findet sich im Reiter *Details* beim jeweiligen Typen der Abschnitt *View-Konfiguration* -> *Objekt/Typ* -> *Objektliste*. Das Erstellen und Pflegen der Tabellen-Konfiguration wird am Beispiel der Objekte des Typs *Thema* erläutert.

Objekte vom Typ *Thema* bieten selbst als zusätzliche Eigenschaften Synonym und die Relation *ist Thema von* an. Diese Eigenschaften, zusammen mit dem Namen des Themas sollen die Bestandteile der zu bauenden Tabelle ergeben.

Der Typ *Thema* wird bearbeitet, in der Ansicht wird der Reiter *Details* gewählt und hier der Punkt *View-Konfiguration* -> *Objekt* -> *Objektliste* angeklickt.



Noch wurde keine Tabellenkonfiguration mit diesem Typ verknüpft. Durch Klicken auf den Knopf *Neu*  wird eine neue, leere Konfiguration erzeugt. Diese kann dann selektiert und wie oben beschrieben bearbeitet werden.

Einstellungsmöglichkeiten für die Sortierung

Name	Wert
Absteigend sortieren	Steuert, ob auf- oder absteigend sortiert wird. Ist dieser Parameter nicht gesetzt, wird aufsteigend sortiert.
Primäres Sortierkriterium	Auswahlmöglichkeit für das Kriterium, nach dem die Eigenschaften sortiert werden: • *Position*: Die in der Konfiguration festgelegte Reihenfolge wird verwendet (Default). • *Wert*: Inhalt des Attributes bzw. Anzeigename des Relationszieles wird verwendet. • *Skript zur Sortierung*: Das in dem Attribut Skript zur Sortierung hinterlegte Skript wird zur Ermittlung des Sortierkriteriums verwendet.
Sekundäres Sortierkriterium	Sortierkriterium für Eigenschaften, die für das primäre Sortierkriterium den gleichen Wert haben. Einstellmöglichkeiten analog zum *Primären Sortierkriterium*
Skript zur Sortierung	Verweis auf ein registriertes Skript, das den Sortierschlüssel für das primäre bzw. sekundäre Sortierkriterium zurückgibt.

1.7.2.2.1 Spaltenkonfigurationen

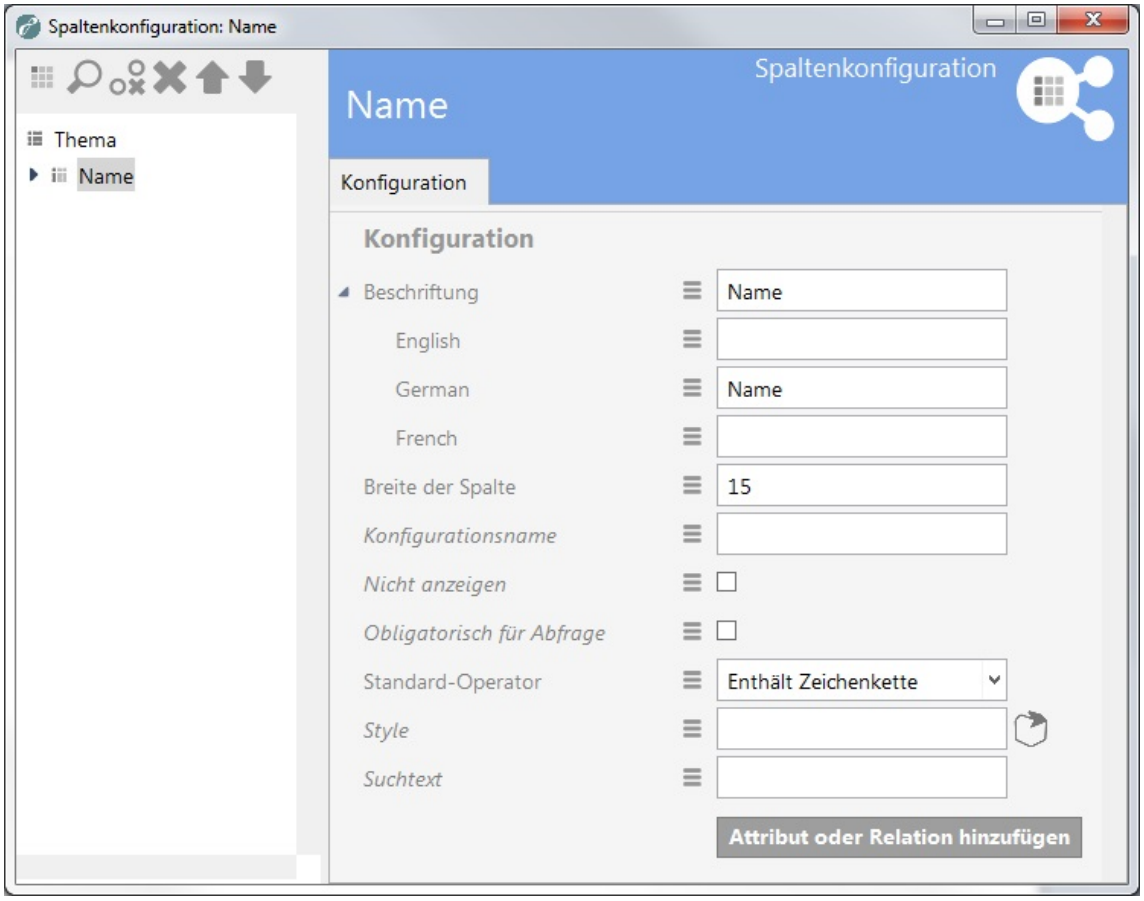
Wie bereits erwähnt, tragen *Spaltenkonfigurationen* Eigenschaften, die der Festlegung der Darstellung und des Verhaltens der Spalte in der Tabelle dienen. Erst wenn Eigenschaften an den in der Spaltenkonfiguration enthaltenen Spaltenelementen konfiguriert werden, wird die Spalte angezeigt.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Wird in der Titelzeile der Spalte angezeigt. Hierbei ist zu beachten, dass *Beschriftung* der Anzeige in der Tabelle dient, die Spaltenkonfiguration aber zusätzlich noch das Attribut *Konfigurationsname* enthält. Dieser Name dient allein der Verwaltung und dem Auffinden der Konfiguration in der semantischen Graph-Datenbank und wird nicht angezeigt oder ausgegeben.

Breite der Spalte (%)	Für die Breite der Spalte wird hier ein prozentualer Wert erwartet (für 60% muss also "60" eingegeben werden).
Nicht anzeigen	Ist dieser Wert gesetzt, wird die komplette Spalte nicht angezeigt. Dies dient dazu z.B. eine Suche mit den Werten dieser Spalte durchzuführen. Der Anwender soll aber nicht die Möglichkeit haben Veränderungen an dieser Spalte vorzunehmen.
Obligatorisch für Abfrage	Ist dieser Wert gesetzt, muss die Spalte ausgefüllt sein, um suchen zu dürfen.
Sortierpriorität	Nach der Spalte mit Sortierpriorität 1 wird primär sortiert. Bei gleichen Werten wird nach der Spalte mit der nächst höheren Sortierpriorität sortiert usw. Die Standardsortierreihenfolge ist aufsteigend. Um absteigend zu sortieren negiert man die Sortierpriorität oder setzt das Meta-Attribut "Absteigend sortieren".
Standard-Operator	Operator, der initial bei der Suche für einen Suchtext angewendet wird.
Suchtext	Initial kann eine Spalte mit einem Suchtext versehen werden.

Beispiel



Spaltenkonfiguration für die Spalte Name

1.7.2.2.2 Spaltenelemente

Ein *Spaltenelement* dient der Zuweisung, welche Inhalte eine Tabellenspalte darstellen soll und wie dies zu geschehen hat. Es können entweder an den semantischen Objekten definierte Eigenschaften wie Attribute und Relationen spezifiziert oder Strukturabfrage-Bausteine oder Skript-Bausteine verwendet werden.

Einstellungsmöglichkeiten

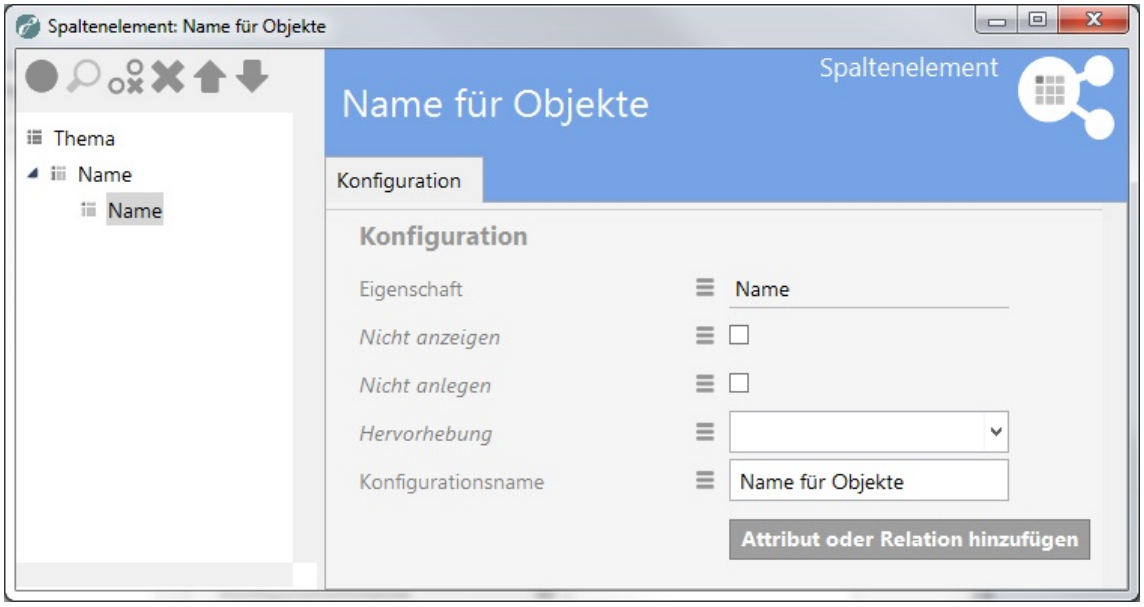
Name	Wert
Eigenschaft (obligatorisch oder Skript)	Verknüpfung zu einem Eigenschaftstyp, der angezeigt werden soll.
Skript (obligatorisch oder Eigenschaft)	Ausführen des Skripts *cellValues* zur Ermittlung der Werte, die angezeigt werden sollen.
Nicht anzeigen	Über dieses boolesche Attribut kann gesteuert werden, ob Werte der ausgewählten Eigenschaft angezeigt werden sollen. Standardmäßig werden alle Eigenschaften angezeigt.

Nicht anlegen	Dieses Attribut steuert, ob diese Eigenschaft beim Erzeugen eines neuen Objekts erzeugt werden soll, falls das entsprechende Eingabefeld der Spalte einen Wert enthält. Standardmäßig werden neue Eigenschaften erzeugt.
Nicht suchen	Hier kann eingestellt werden, dass die konfigurierte Eigenschaft nicht in die Suche übernommen wird. D.h. eingegebene Suchwerte werden nicht über diese Eigenschaft gesucht. Achtung: Wenn alle Spaltenelemente einer Spalte auf "Nicht suchen" geschaltet werden, hat dies denselben Effekt wie "Nicht anzeigen"!
Hervorhebung	Hier können für das Anzeigen von Werten Formatierungsvorgaben gemacht werden, zur Wahl steht derzeit nur *Unterstreichen*.
Hits verwenden	Standardmäßig werden Objekte erzeugt. Möchte man allerdings in dem Skript *cellValues* die Hits weiterverarbeiten, muss *Hits verwenden* eingeschaltet werden.
Relationszielansicht	Derzeit steht nur die Alternative *Drop down* zur Verfügung. Wird diese ausgewählt, so werden die möglichen Werte, die sich für die Filterung in der Tabelle für diese Spalte eintragen lassen, aus den möglichen Relationszielen gemäß Schema als Dropdown-Liste zusammengestellt, so dass ein möglicher Wert schnell spezifiziert werden kann. Dies empfiehlt sich für überschaubare Mengen an möglichen Relationszielen. Anmerkung: Dieser Parameter steht nur zur Verfügung, wenn eine Eigenschaft vom Typ Relation gewählt wurde.

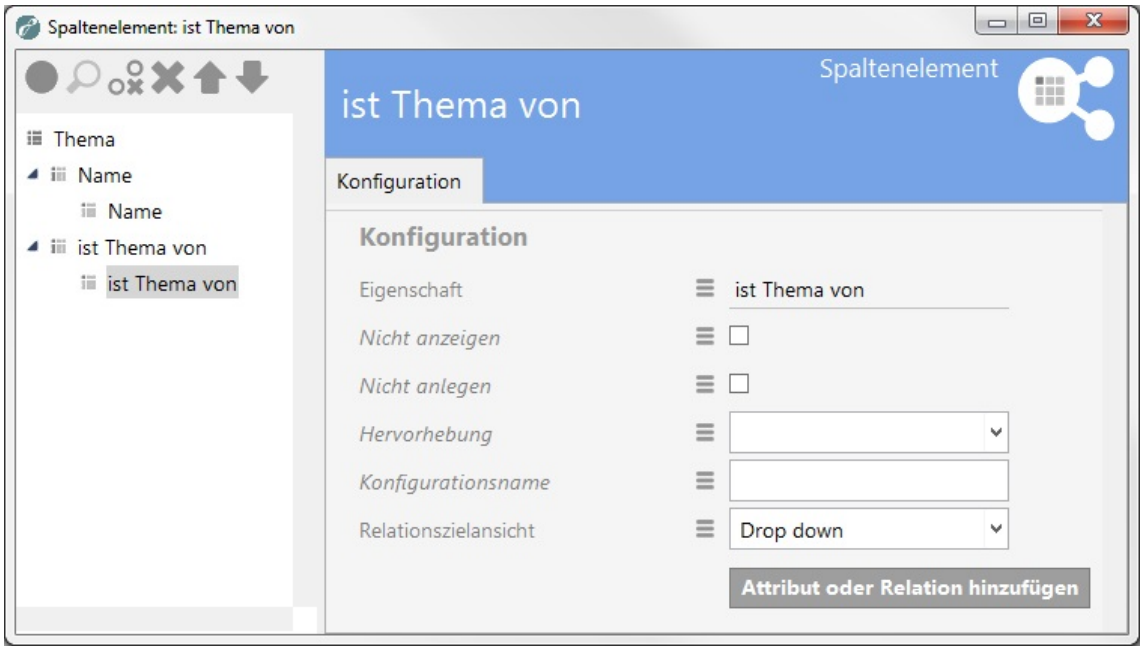
Es ist möglich für eine Spaltenkonfiguration mehrere Spaltenelemente zu definieren. Das ist z.B. dann sinnvoll, wenn mehrere Attribute in der Suche berücksichtigt werden sollen wie beispielsweise das Attribut Name und Synonym, aber nur eines davon angezeigt werden soll.

Beispiel

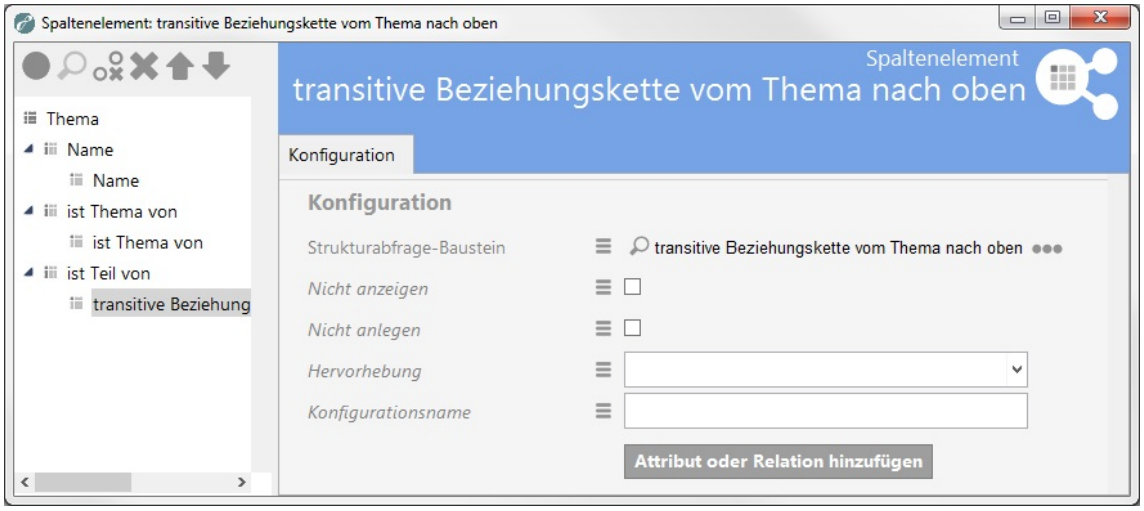
Im ersten Spaltenelement der Spaltenkonfiguration *Name* wurde das Attribut *Name* hinterlegt.



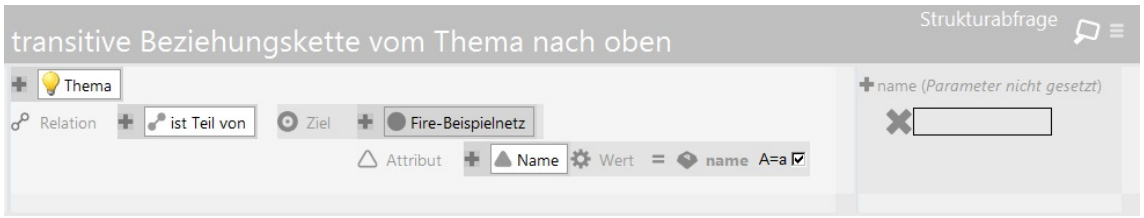
Auf der zweiten Spalte wurde im Spaltenelement die Beziehung *ist Thema von* hinterlegt.



Auf der dritten Spalte wurde im Spaltenelement der Strukturabfrage-Baustein *transitive Beziehungskette vom Thema nach oben* hinterlegt.



Hinterlegte Struktur-Abfrage



Um Werte aus dem Eingabefeld der Spalte übernehmen zu können, muss die hinterlegte Strukturabfrage konfigurierte Parameter haben. Es können mehrere Parameter angebracht werden, diese sind beim Auswerten der Strukturabfrage alle mit demselben Wert belegt.

Vorsicht: Hier liegt ein Unterschied zu sonstigen Fällen, in denen die Strukturabfrage verwendet wird. Normalerweise bestimmt das Ausgangsobjekt (in diesem Fall wäre dies "Thema") die Ergebnisse, hier sind es jedoch die Objekte oder Eigenschaften, an denen der Parameter angebracht ist (in diesem Fall das Namensattribut).

Der in der Spalte gezeigte Wert ist, wenn keine weiteren Anpassungen vorgenommen werden, der Wert des zum Filtern verwendeten Attributes. Wenn sich der angezeigte Wert nicht aus dem zur Filterung verwendeten Attribut ergibt, so gibt es zwei Möglichkeiten:

- der Bezeichner "*renderTarget*" kann an Relationszielen angebracht werden. Die hiermit markierten Objekte werden als Spaltenwert in der Tabelle angezeigt.
  "renderTarget" bewirkt außerdem, dass bei einer Ausgabe über die JavaScript API die Eigenschaften zur Darstellung als Link mit ausgegeben werden.
- der Bezeichner "*renderProperty*" kann an Attributen angebracht werden. Die hiermit markierte Eigenschaften werden als Spaltenwerte in der Tabellenspalte angezeigt.

Wird der Suchbaustein nicht zur Filterung verwendet, so muss das anzuzeigende Element mittels renderTarget oder renderProperty bestimmt werden!

Die Strukturabfragen, die in den Baustein des Spaltenelements eingefügt werden, können aus einer Liste bereits registrierter Strukturabfragen ausgewählt werden, sie können aber auch für genau diesen Baustein neu angelegt werden, was auch die Vergabe eines Registrierungsschlüssels mit sich bringt. Die Eigenschaft *Nicht anlegen* hat auf Spalten, die mit einem Strukturabfrage-Baustein belegt sind, keine Wirkung.

Auf die vierte Spalte wurde ein Script-Baustein abgebildet

Es sollen die Verantwortlichen für die Objekte, mit denen das in der Tabelle gelistete Thema mit *ist Thema von* verknüpft ist, angezeigt werden. Wie bei der Strukturabfrage kann das zugeordnete Skript aus einer Liste von bereits registrierten Skripten ausgewählt oder aber im Dialog neu angelegt (und registriert) werden. Der Skript-Editor öffnet sich nach Klicken auf den Skript-Baustein-Namen.

```
/*
 * Returns matching elements for colum search value "objectListArgument"
*  Note: "elements" may be undefined if no partial query result is available.
 * Return undefined if the script cannot provide any partial result itself.
 */
function filter(elements, queryParameters, objectListArgument) {
    return elements;
}

// Returns cell values for the given element
function cellValues(element, queryParameters) {
    var result = new Array();
    var firstTargets = element.relationTargets("istThemaVon") ;
    if ( firstTargets.length == 0 ) { return result ;
    }
    else {
        for (var i = 0; i < firstTargets.length; i++) {
        var secondTargets = firstTargets[i].relationTargets("hatVerantwortlichen");
        for (var j = 0; j < secondTargets.length; j++) {
            result.push(secondTargets[j].name());};
            };
        };
    return result.join(', ');
}
```

In diesem Fall ist die Sprache des Skriptbausteins JavaScript. Hier müssen zwei Teile gepflegt werden, der obere Teil dient der Filterung aller Elemente der Tabelle anhand des in der Spalte eingetragenen Wertes *objectListArgument*, der zweite Teil gibt an, wie für ein Element ein auszugebender Wert berechnet wird. Der erste Teil ist im Moment nicht ausgeführt. Zu bei-

den Teilen wird ein Code-Muster beim Erzeugen eingefügt, auf das beim Erstellen aufgebaut werden kann.

Wenn KScript als Sprache im Skript-Baustein gewählt wurde, um die Ausgabe einer Spalte zu steuern, dann muss das ausgewählte (registrierte) Skript zu jedem Objekt, das eine Zeile bildet, einen Rückgabewert für die Spalte liefern.

Da in KScript im Prinzip nur eine Ausgabe vorgesehen ist, wurde für die Filterung folgende Konvention getroffen:

Wenn es in dem ausgewählten Skript eine Funktion mit Namen *objectListScriptResults* und einem deklarierten Parameter gibt, so wird diese Funktion mit dem Argument der zugehörigen Sucheingabe aufgerufen, um die Menge der passenden Objekte zurückzuliefern. Die Funktion wird auf dem Wurzelbegriff oder der bisherigen Treffermenge als Ausgangsobjekt aufgerufen - je nach dem, wie die Suche am besten gelöst werden kann. Damit diese Variante wirklich effizient wird, ist es empfehlenswert, die Sucheingabe entsprechend auszuwerten und mit dem Ergebnis eine registrierte Strukturabfrage aufzurufen, um deren Ergebnis an die Objektliste weiterzuleiten.

1.7.2.3 Menüs und Aktionen

1.7.2.3.1 Menü

Die Menüs bedienen hauptsächlich zwei Funktionalitäten beim Umgang mit Aktionen. Zum Einen können mit ihnen Aktionen gegliedert werden, zum Anderen kann festgelegt werden, wo die Menüs zum Einsatz kommen. Im Knowledge-Builder und ViewConfigMapper gibt es viele Orte, an denen die Inhalte von Menüs angezeigt werden, beispielsweise Knöpfe am Kopf eines Editors oder das Kontextmenü an einer einzelnen Eigenschaften. Derzeit (4.3) lassen sich noch nicht an alle Stellen, an denen Menüs theoretisch möglich sind, Menüs anbringen.

Im Folgenden werden die direkten Einstellungsmöglichkeiten an einem Menü und die bereits existierenden Menüarten und deren Verwendung beschrieben.

Einstellungsmöglichkeiten

Name	Wert
Beschriftung	Ob die Beschriftung angezeigt wird, richtet sich nach der Menüart und dem Interface, das sich um die Anzeige kümmert.
Ersetzt Standardmenü	Dieser Parameter hat bisher nur Auswirkungen auf den Knowledge-Builder. Bei einigen Editoren, wie z.B. für eine Tabelle, werden Standardmenüs angezeigt. Mit Hilfe dieses Parameters können diese ausgeschaltet werden.
Konfigurationsname	Findet keine Verwendung im Userinterface. Der Ersteller einer Konfiguration hat hier die Möglichkeit, einen für ihn verständlichen Namen zu vergeben, um diese Konfiguration später besser wiederfinden und in anderen Konfigurationen wieder verwenden zu können.
Menüart	Die Menüart beschreibt die Verwendung des Menüs in den einzelnen Komponenten. Die Menüarten werden weiter unten beschrieben.

Werkzeugleiste

Name	Wert
Standardaktionen hinzufügen	Dieses Icon wird nur angezeigt, wenn Standardaktionen hinzugefügt werden können. Dies ist aktuell bei einer Tabellen- und Suche-Konfiguration möglich. Diese Funktion bietet die Möglichkeit, beim gesetzten Parameter *Ersetzt Standardmenü* wieder einige oder alle Standardmenüeinträge zu reaktivieren und die Reihenfolge der einzelnen Aktionen zu ändern.

Anmerkungen

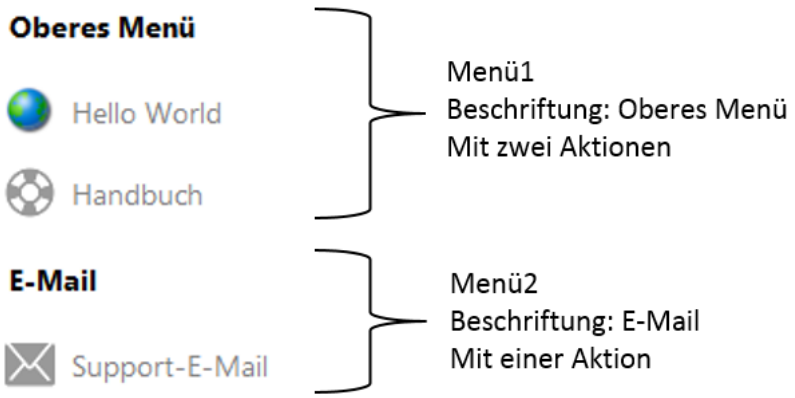
- Ist der Parameter *Ersetzt Standardmenü* nicht gesetzt, so werden die Aktionen, die in den Menüs enthalten sind, der Reihe nach hinten angefügt.
- Soll die Reihenfolge der Standardaktionen geändert werden, so muss der Parameter *Ersetzt Standardmenü* gesetzt sein. Anschließend können die Standardaktionen mit der Aktion *Standardaktionen hinzufügen* ergänzt werden. Die Standardaktionen können nun beliebig sortiert und mit eigenen Aktionen gemischt werden.

Kontextmenü

Icon	
Knowledge-Builder	Derzeit lassen sich Kontextmenüs für eine Tabellenzeile und einen Objekteditor erweitern oder neu definieren. **Objektkonfiguration**: In einer beliebigen Top-Konfiguration eines Elementes können unter dem Reiter *Menü* Menüs angelegt werden. Auch hier kann das Standardmenü durch das Setzen des Parameters *Ersetzt Standardmenü* ausgeschaltet werden. **Tabellen-Kofiguration**: Im Kontextmenü für eine Tabellenzeile gibt es zwei Abschnitte. Der erste bezieht sich auf das ausgewählte Element, der zweite bezieht sich auf die Tabelle. Für die beiden Abschnitte gibt es zwei unterschiedliche Konfigurationsorte. Für den ersten Fall muss das Menü für ein Element mit einer beliebigen, am besten neuen Konfiguration verknüpft werden, die wiederum über *anwenden in* an die Tabelle, die das Kontestmenü anzeigen soll, gehängt wird. Im zweiten Fall kann das Menü direkt an der Tabelle angebracht werden.
ViewConfigMapper	*Findet derzeit keine Verwendung im ViewConfigMapper.*

JSON	<pre>"label" : "Menü (Kontext)",
	"actions" : [{...}],
	"type" : "contextMenu"</pre>

Liste

Icon	
Knowledge-Builder	Findet nur Anwendung in der Startansicht-Konfiguration. Es werden die konfigurierten Aktionen in einer Liste dargestellt. Werden für die Menüs Beschriftungen vergeben, werden diese mit angezeigt und bieten somit eine Strukturierungsmöglichkeit. 
ViewConfigMapper	*Findet derzeit keine Verwendung im ViewConfigMapper.*

JSON	<pre>"label" : "Menü (Liste)",
	"actions" : [{...}],
	"type" : "listMenu"</pre>

Werkzeugliste

Icon	

Knowledge-Builder	Die Aktionen, die in den Menüs enthalten sind, werden der Reihe nach angefügt. Eine Unterteilung nach Menüs und eine Beschriftung der Menüs werden derzeit nicht berücksichtigt.
ViewConfigMapper	Die Aktionen, die in den Menüs enthalten sind, werden der Reihe nach angefügt. Eine Unterteilung nach Menüs und eine Beschriftung der Menüs werden derzeit nicht berücksichtigt.

JSON	`"label" : "Menü (Werkzeugleiste)",`
	`"actions" : [{...}],`
	`"type" : "toolbar"`

1.7.2.3.2 Aktion

Mit Hilfe von Aktionen lassen sich Zusatzfunktionalitäten an Detail- und Tabellenansichten anbringen.

Im Knowledge-Builder werden die vollständig konfigurierten Aktionen als zusätzliche Schaltflächen angezeigt. Bei einer Selektion wird das enthaltene Skript ausgeführt.

Bei einer Kommunikation mit einer anderen Applikation über JSON wird eine Beschreibung der möglichen Aktion herausgeschrieben. Die Gegenstelle hat dann die Möglichkeit, über einen passenden Request diese Aktion auszuführen zu lassen.

Aktion an einer Objektliste

Die Beschriftung wird als Tooltip im Knowledge-Builder angezeigt. Das ausgewählte Symbol (eine beliebige Bilddatei) wird auf Buttongröße skaliert.
Achtung: Ist kein Symbol angegeben, wird kein Button angezeigt.

1.7.2.3.3 Skripte einer Aktion

Aktuell lassen sich die vier folgenden Skript-Typen ausführen.

Die drei letzten Skript-Typen finden bisher nur bei der Ausgabe über JSON Verwendung.

Attribut Skript

Dieses Skript wird ausgeführt, wenn der benutzerdefinierte Knopf gedrückt oder ein Action-Request gesendet wird.

"Verweis auf Skript"-Attribut **Skript**

```
function onAction(element, context)
{
```

```
    return element;
}
```

Variablen

Das Skript einer Aktion kann auf folgende vordefinierte Variablen, in *context* enthalten, zugreifen:

Detaileditor

Variable	Wert
selectedElement	Ausgewähltes Objekt oder ausgewählter Typ
type	Objekttyp. Falls das Element ein Typ ist, wird der Typ selbst verwendet

Objektliste

Variable	Wert
selectedElement	Ausgewähltes Objekt oder ausgewählter Typ. nil, falls kein Element oder mehrere Elemente ausgewählt wurden.
selectedElements	Ausgewählte Elemente
elements	Alle Elemente der Objektliste
type	Typ der Objektliste

Attribut Skript (actionResponse)

"Verweis auf Skript"-Attribut **Skript (actionResponse)**

```
function actionResponse(element, context, actionResult)
{
    var actionResponse = new $k.ActionResponse();

    actionResponse.setData(actionResult);
    actionResponse.setFollowup("new");
    actionResponse.setNotification("Erledigt","warn");

    return actionResponse;
}
```

Die ActionResponse kann um Werte für *Followup* und *Nofication* erweitert werden. Diese Werte können von anderen Anwendungen wie z.B. dem ViewConfigMapper ausgewertet wer-

den.

Attribut Skript (visible)

"Verweis auf Skript"-Attribut **Skript (visible)**

```
function actionVisible(element, context)
{
    return true;
}
```

Das Skript darf nur einen booleschen Wert zurückliefern. Anhand des Wertes wird entschieden, ob der Knopf angezeigt werden soll oder nicht. Bei der Ausgabe über JSON wird die Aktion nicht weiter betrachtet und somit auch nicht ausgegeben.

Attribut Skript (enabled)

"Verweis auf Skript"-Attribut **Skript (enabled)**

```
function actionEnabled(element, context)
{
    return true;
}
```

Das Skript darf nur einen booleschen Wert zurückliefern. Anhand des Wertes wird entschieden, ob der Knopf aktiv ist oder nicht. Bei der Ausgabe über JSON wird der Marker *enabled* auf *false* gesetzt.

1.7.2.3.4 UI-spezifische Funktionen

Das die Aktion realisierende Skript kann auf UI-spezifische Funktionen zurückgreifen. Wegen dieser Möglichkeit wird das Skript nicht innerhalb einer einzigen Transaktion ausgeführt. Werden Transaktionen aufgrund von Schreibzugriffen benötigt, dann müssen diese im Skript aufgerufen werden.

UI-spezifische Funktionen in JavaScript

```
$k.UI.alert(message, windowTitle)
```

Zeigt eine Meldung an.

```
$k.UI.requestString(message, windowTitle)
```

Benutzer kann eine Zeichenkette eingeben.

```
$k.UI.confirm(message, windowTitle)
```

Öffnet einen Abbrechen-Dialog.

`$k.UI.choose(objects, message, windowTitle, stringFunction)`

Objekt aus einer Menge auswählen lassen.

`$k.UI.openEditor(element)`

Standardeditor für das Objekt öffnen.

`$k.UI.notificationDialog(notificationFunction, parameters, windowTitle)`

Es wird ein Warte- bzw. Benachrichtigungsdialog geöffnet. Dieser kann, je nachdem wie er konfiguriert ist, abgebrochen werden.

Mögliche Parameter:

Parameter	Beschreibung	Standardwert
autoExpand	Ist der Anzeigebereich des Dialogs initial geöffnet.	true
canCancel	Kann der Dialog abgebrochen werden.	true
stayOpen	Bleibt der Dialog nach Beendigung der Funktion geöffnet.	true

Beispiel:

```
ui.notificationDialog(
    function() {
        ui.raiseNotification("start");
        for (var i = 0; i < 10; i ++)
            ui.raiseNotification("" + i + "*" + i + "=" + (i*i));
        ui.raiseNotification("end");
        return undefined;
    },
    { "canCancel" : false },
    "Ein Wartedialog"
)
```

Mit der folgenden Function *raiseNotification* können Meldungen auf dem Anzeigebereich ausgegeben werden.

`$k.UI.raiseNotification(message)`

Diese Benachrichtigung wird nur von der Function *notificationDialog* gefangen und die Nachricht wird nur dort im Anzeigebereich ausgegeben.

UI-spezifische Funktionen in KScript

<UIWarn message="${name() + ' lebt'}"/>

Zeigt eine Meldung an.

<UIRequestString message="'Name'"/>

Benutzer kann eine Zeichenkette eingeben.

<UIConfirm message="'Sind Sie versichert?'"/>

Öffnet einen Abbrechen-Dialog.

<UIChooseObject message="'Person'" objects="//Person/instances()"/>

Objekt aus einer Menge auswählen lassen.

<UIOpenEditor/>

Standardeditor für das Objekt öffnen.

<UICreateTopic/>

Objektliste: Ein neues Topic samt Eigenschaften für die Suchfeldeingaben wird angelegt. Editor: Keine Wirkung, liefert null.

<UIOpenTopicList title="Personen" objects="//Person/instances()"/>

Öffnet ein Objektlistenfenster mit den durch das Argument *objects* festgelegten Wissensnetzelementen.

1.7.2.4 Graph-Editor

Die Graph-Editor-Konfiguration ermöglicht es nur bestimmte Typen und Relationen im Graphen anzuzeigen. So kann verhindert werden, dass unerwünschte Typen und Relationen im Graphen zu sehen sind. Die Graph-Editor-Konfiguration kann ebenfalls über JavaScript-Funktionen angefragt werden. Sie findet beispielsweise Verwendung im Net-Navigator.

1.7.2.4.1 Die Standard Graph-Editor-Konfiguration

Sollte keine Graph-Editor-Konfiguration angelegt sein, wird im Graph-Editor des Knowledge-Builders auf eine Fallback-Graph-Editor-Konfiguration zurückgegriffen.

Diese unterscheidet sich für Administratoren und Anwender um einen sinnvollen Standard

zu bieten.

Konfigurierte Typen:

Administratoren	Alle Typen und Individuen
Anwender	Alle Typen und Individuen ohne Technik

Konfigurierte Relationen für Administratoren und Anwender:

Alle Benutzerrelationen außer:

- abstrakte Relationen
- inverse Einweg Relationen

SystemRelationen:

- erweitert Objekte von
- Objekte sind Domäne von
- Inverser Relationstyp
- ist Objekt von
- ist Untertyp von
- ist Obertyp von
- An Attributen: ist Eigenschaft von

1.7.2.4.2 Konfiguration des Graph-Editors

Der Graph-Editor wird im Technik-Bereich unter *View-Konfiguration* >> *Objekttypen* >> *Konfigurationselement* >> *Graph-Editor-Konfiguration* konfiguriert.

Graph-Editor-Konfiguration

Eine Graph-Editor-Konfiguration besteht aus einem Graph-Editor-Konfigurations-Objekt. An diesem können Beschriftung und die Anwendung, in welcher diese Graph-Editor-Konfiguration gilt (Bsp. Net-Navigator), konfiguriert werden.

Knotenkategorie

Über den Button *"Knotenkategorie neu anlegen"* kann eine Knotenkategorie neu angelegt werden. Eine Knotenkategorie definiert ein oder mehrere Typen bzw. dessen Objekte, welche im Graphen angezeigt werden sollen. Diese Typen können über die Relation *"anwenden auf"* im Reiter Verwendung festgelegt werden. Die Option *"anwenden auf Untertypen"* bewirkt dabei, dass der Typ und nicht seine Objekte angezeigt werden. Möchte man sowohl den Typ als auch dessen Objekte angezeigt bekommen, legt man zwei Relationen *"anwenden auf*" mit selbem Ziel an, wobei sich nur *"anwenden auf Untertypen"* unterscheidet.
Ein Knoten wird im Graphen angezeigt, wenn wenigstens eine Knotenkategorie diesen Typen konfiguriert. Dabei wird auch Vererbung berücksichtigt. Sollte ein Knoten keiner direkten Knotenkategorie angehören, gehört er der Knotenkategorie an, die den ersten Obertypen konfiguriert.
Die Option "an konkreten Typ anpassen" bildet automatisch Knotenkategorien für alle Untertypen der konfigurierten Typen. Dies wird in JavaScript-Funktionen berücksichtigt (siehe

Kapitel Graph-Editor Konfiguration in JavaScript), findet im Knowledge-Builder jedoch keine Verwendung.

Verknüpfungskonfiguration

Eine Knotenkategorie kann Verknüpfungskonfigurationen haben. Diese definieren, welche Relationen an der Knotenkategorie erwünscht sind. Dies beeinflusst das Ausklappen von Relationen, sowie das Anlegen von Relation im Graphen. Eine neue Verknüpfungskonfiguration kann über den Button *"neues Anlegen"* >> *"Verknüpfungskonfiguration"* angelegt werden. Konfigurierte Relationen können entweder über das Feld "Relation" direkt angegeben, oder durch ein Skript bzw. eine Strukturabfrage ermittelt werden. Auch konfigurierte Relationen vererben die Konfiguration an ihre Untertypen.

Eine Relation kann angezeigt, ausgeklappt und erstellt werden, falls sowohl Domain als auch Inversedomain des Relationstyps eine Knotenkategorie haben, und der Relationstyp an der Knotenkategorie des Topics konfiguriert ist.

1.7.2.4.3 Graph-Editor-Konfiguration in JavaScript

Man kann einen konfigurierten Graphen auch über JavaScript-Funktionen anfragen. Dazu gibt es einige Funktionen die im Folgenden näher beschrieben werden.

Beispiel	Beschreibung
`$k.GraphConfiguration.for("ApplicationName");`	Liefert ein Graph-Editor-Konfigurationsobjekt für einen optionalen Kontext.
`$k.GraphConfiguration.fromConfigurationTopic(topic);`	Liefert die Graph-Editor-Konfiguration welche durch das configurationTopic definiert wird.
`graph.configElement();`	Liefert das KonfigurationsTopic der Graph-Editor-Konfiguration.
`graph.addElements([node1, node2]);`	Fügt dem Graphen ein Array von Topics hinzu.
`graph.expand([node1, node2], 2);`	Fügt dem Graphen alle Nachbarn bis zu einer Tiefe 2 der angegebenen Knoten hinzu.
`graph.expandNodes(2);`	Fügt dem Graphen alle Nachbarn bis zu einer Tiefe 2 aller Knoten hinzu.
`graph.collapse([node1, node2]);`	Entfernt die Nachbarn der angegebenen Knoten aus dem Graphen.
`graph.categories();`	Liefert alle Knotenkategorien als JSON.

`graph.possibleRelationTargets(node1);`	Liefert die erstellbaren Relationstypen von einem Knoten node1 zu allen anderen Knoten sortiert nach Zielknoten als JSON.
`graph.render(["categories"]);`	Liefert den Graphen als JSON. Die angegebenen Filter werden nicht gerendert.
`graph.semanticElements();`	Liefert alle Knoten im Graphen.

1.7.3 Knowledge-Builder-Konfiguration

Die hier beschriebenen View-Konfigurationen betreffen ausschließlich den Knowledge-Builder. Weitere View-Konfigurationen, die den Knowledge-Builder betreffen, finden sich auch an anderen Stellen in Kapitel 7, können dann aber auch zusätzlich jeweils die Ausgabe in JSON betreffen.

1.7.3.1 Ordnerstruktur

Der linke Teil des Hauptfensters im Knowledge-Builder dient der Navigation durch das semantische Modell. Dazu wird dort eine hierarchische Ordnerstruktur angezeigt. Diese lässt sich in mehrere Hauptbereiche gliedern, die dann als Balken angezeigt werden. Klickt man einen solchen Balken an, dann klappt die darunter liegende Ordnerstruktur auf, über die man Inhalte (Elemente, Abfragen, Import/Export-Abbildungen usw.) erreichen kann. Die Inhalte werden auf der rechten Seite aufgelistet und können dort bearbeitet werden.

1.7.3.1.1 Die Standard-Ordnerstruktur

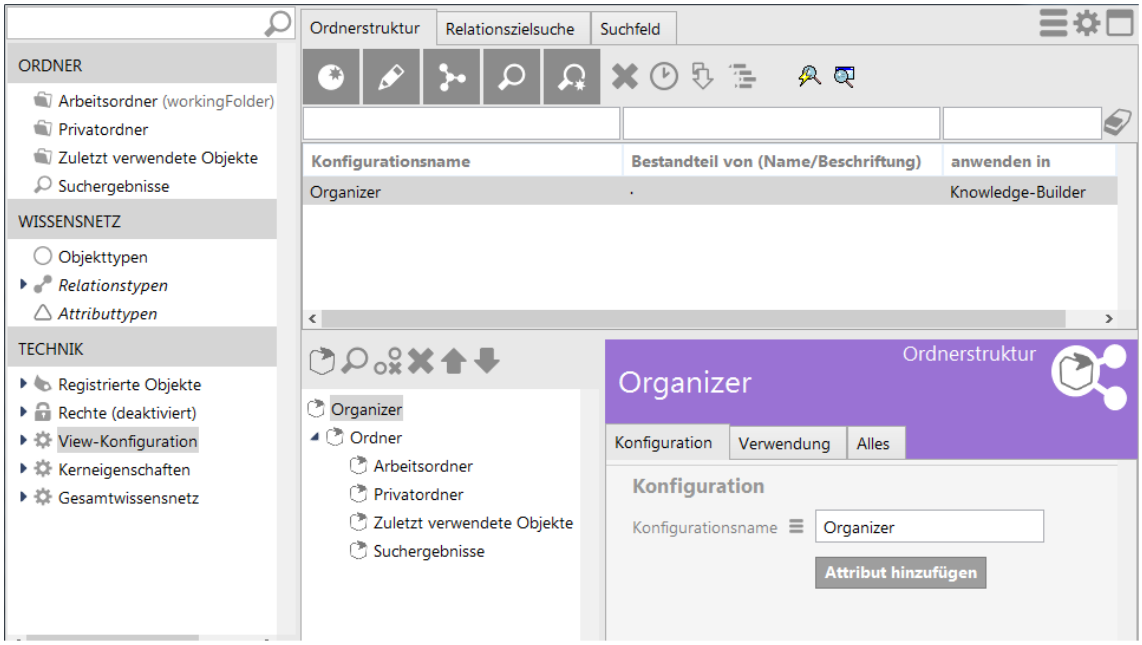
Die Konfiguration der Standard-Ordnerstruktur stellt Ordner zur Verfügung, so dass im semantischen Modell navigiert und Inhalte abgelegt werden können. Für Administratoren werden drei Hauptbereiche zur Verfügung gestellt.

Der obere Hauptbereich **"Ordner"** stellt Ordner für die Anlage weiterer Ordner und das Verwalten von Inhalten zur Verfügung. Das sind der Arbeitsordner, der Privatordner, der Ordner "Zuletzt verwendete Objekte" und der Ordner "Suchergebnisse".

Der zweite Hauptbereich **"Wissensnetz"** ermöglicht die Navigation zu den Elementen über die Hierarchie der Typen. Die hier zu erreichenden Elemente sind Typen, Objekte und auch Attribute und Relationen. Dafür enthält der Bereich drei Ordner:

- Objekttypen für die Hierarchie der Objekttypen und ihrer konkreten Objekte
- Relationstypen für die Hierarchie der Relationen
- Attributtypen für die Hierarchie der Attribute

Der dritte Hauptbereich **"Technik"** ermöglicht es Administratoren Änderungen, Einstellungen und Konfigurationen verschiedenster Art im sem. Netz vorzunehmen. Dazu gehören u.a. Registrierte Objekt, das Rechtesystem und Trigger.



Die Konfiguration dieser Standard-Ordnerstruktur kann im Technik-Bereich >> View-Konfiguration überprüft, verändert und den Bedürfnissen der Anwender angepasst werden.

Anmerkung: Für Administratoren wird immer die Standard-Ordnerstruktur angezeigt. Konfiguriert man eine View-Konfiguration für Ordner, so werden diese nur für Nicht-Administratoren angezeigt. Möchte man als Administrator auch die konfigurierte Sicht der Ordnerstruktur angezeigt bekommen, so kann das in den persönlichen Einstellungen des Knowledge-Builder auswählt werden: Unter "Einstellungen" > "Persönlich" > "View-Konfiguration" die Auswahl "Konfiguriert" wählen.

1.7.3.1.2 Konfiguration der Ordnerstruktur

Die Ordnerstruktur wird im Technik-Bereich unter *View-Konfiguration* >> *Objekttypen* >> *Knowledge-Builder-Konfiguration* >> *Ordnerstruktur* konfiguriert. Einen schnellen Zugriff auf die Konfigurationen erhält der Admin, wenn er im Technik-Ast den Knoten *View-Konfiguration* selektiert und im rechten Teilfenster auf dem Reiter *Ordnerstruktur* das Objekt *Organizer* auswählt.

In der Konfiguration werden Ordnerstrukturelemente hierarchisch miteinander verknüpft. Der Wurzelknoten dieser Hierarchie ist ein Objekt des Typs *Ordnerstruktur*. Initial ist eine Ordnerstruktur mit Namen *Organizer* enthalten. Alle Unterknoten und deren Unterknoten sind vom Typ *Ordnerstrukturelemente*. Die Hierarchie in der Konfiguration zeigt direkt die im Hauptfenster dargestellte Hierarchie. Die direkten Unterknoten des Wurzelknotens werden im Hauptfenster als Balken dargestellt, so dass sich eine optische Abgrenzung der verschiedenen Ordnerhierarchien voneinander ergibt.

Beschriftung ist ein Parameter, den alle Konfigurationstypen gemein haben. Ein Knoten, der durch eine Konfiguration beschrieben wird, wird mit diesem Wert beschriftet. Was im rechten Teil des Hauptfensters angezeigt wird, wenn man einen Knoten selektiert, hängt von den Parametern des Ordnerstrukturelements ab. Dazu muss der Parameter **Ordnertyp** belegt werden, für den eine Auswahl an Typen zur Verfügung steht. Diese Ordnertypen und deren zusätzliche Parameter werden in der folgenden Tabelle aufgeführt.

Ordnertyp (obligatorisch)	**Parameter**	**Beschreibung**
Attributtypen	Typ	Der angegebene Attributtyp und alle seine Untertypen werden in einem hierarchischen Baum angezeigt.
Privatordner	-	Anzeige des Ordners, den nur der Benutzer selbst sehen darf und der für jeden Benutzer unterschiedlich ist.
Relationstypen	Typ	Der angegebene Attributtyp und alle seine Untertypen werden in einem hierarchischen Baum angezeigt.
Strukturordner	Strukturordner	Ein beliebiger *Strukturordner* kann hier eingehängt werden.
Suchergebnisordner	-	Jeder Benutzer hat einen eigenen Suchergebnisordner, der die letzten Suchergebnisse des Benutzers speichert.
Typbasierte Ordnerstruktur	Ansicht "Ohne Vererbung", Typ	Der angegebene *Typ* und seine Untertypen werden tabellarisch aufgelistet. Ist der Parameter *Ansicht "Ohne Vererbung"* gesetzt, wird nur der angegebene Typ angezeigt. Anmerkung: Zur Steuerung welche Tabellenkonfigurationen auf der rechten Seite Anwendung finden, muss dort die Relation *anwenden in* mit diesem *Ordnerstrukturelement* verknüpft werden.
Virtueller Ordner	-	Ein Ordner, der zur Strukturierung der Ordner dient.
Zuletzt verwendete Objekte	-	Jeder Benutzer hat einen eigenen Ordner, in dem die zuletzt verwendeten Objekte für einen schnelleren Zugriff gespeichert werden.

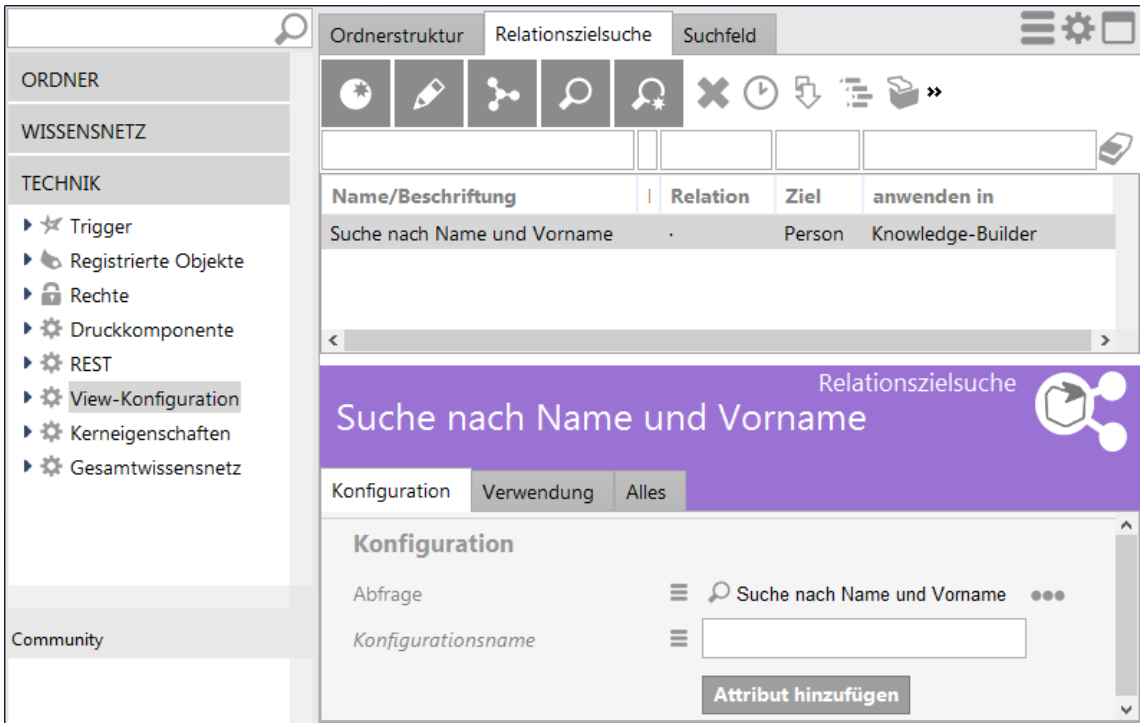
Nur der Konfigurationstyp *Virtueller Ordner* kann weitere Unterkonfigurationen enthalten bzw. nur beim ihm machen Unterkonfigurationen Sinn.

Anmerkung: Bei dem Ordnertyp Attributtypen, Relationstypen und Typenbasierte Ordnerstruktur dient der Parameter Typ zur Angebe des Attribut-, Relations- oder Objekttyp der und dessen Untertypen in dem Ordner angezeigt werden sollen.

1.7.3.2 Relationszielsuche

Die Konfiguration der Relationszielsuche ermöglicht es, auf die Strategie einzuwirken, mit der mögliche Relationsziele gesucht werden. Enthält ein semantisches Modell keine Relationszielsuche, dann wird auf eine Eingabe von "Egon" immer nach einem Objekt mit Namen "Egon" gesucht (d.h. das jeweilig definierte Namensattribut wird verwendet). Durch Angabe einer zuvor definierten Abfrage kann dieses Verhalten verändert werden. Beispielsweise kön-

nte man für die Suche nach Personen eine Abfrage definieren, die sowohl den Vornamen als auch den Nachnamen durchsucht. Sucht man dann nach einem Ziel für eine Relation, deren Zieldomäne Person ist, dann werden die Nachnamen und Vornamen von Personen nach der Eingabe von "Egon" durchsucht. Sinnvoll ist eine angepasste Relationszielsuche auch, wenn man gleichzeitig Namen und Synonyme von Objekten durchsuchen möchte, sodass beispielsweise das Objekt "Architektur" auch gefunden wird, wenn der Anwender "Baukunst" eingibt.



Relationszielsuche konfiguriert für die Suche nach Personen

Wie bei allen Konfigurationen muss der Kontext angegeben werden, in dem die Relationszielsuche verwendet werden soll. Zusätzlich kann die Verwendung auf einen Typ von Relationszielen oder auch auf einen speziellen Relationstyp eingeschränkt werden. Diese Angaben sind jedoch optional.

1.7.3.3 Startansicht

Mit der Konfiguration *Startansicht* lässt sich definieren, welches Hintergrundbild und welche Aktionen im Knowledge-Builder auf der rechten Seite angezeigt werden sollen. Die Anzeige lässt sich jederzeit durch Deselektion (Strg-Taste auf bestehende Selektion im linken Navigationsbaum) hervorrufen.

Einstellungsmöglichkeiten

Name	Wert
Hintergrundbild	Ein Bild
Farbwert für Schriftart einer Aktion	Je nach ausgewähltem Bild muss eine andere Farbe für die Beschriftung der Aktionen gewählt werden, um den Text lesen zu können.

Darüber hinaus lassen sich Aktionen definieren. Siehe dazu Kapitel Aktion. Zusätzlich kann eine Aktionsart festgelegt werden. Hier stehen folgende Einträge zur Verfügung:

Aktionsart	Aktion
Handbuch (spezialisierter Web-Link)	Web-Handbuch wird im Browser geöffnet
Homepage (spezialisierter Web-Link)	Die Homepage wird im Browser geöffnet.
Support-E-Mail (spezialisierter Web-Link)	Ein Fenster für eine neue E-Mail wird mit der Support-E-Mail-Adresse geöffnet.
Web-Link	Frei definierbarer Web-Link
<keine Aktionsart>	Konfigurierte Aktion (mit Skript) ausführen

Ein Web-Link muss vollständig konfiguriert sein, sonst wird er nicht angezeigt.

Abweichend dazu muss dies bei den drei oberen Aktionsarten (spezialisierte Web-Links) nicht so sein. Diese verwenden, falls eine Eigenschaft fehlt, ihre Standardwerte. Es besteht die Möglichkeit die Standardwerte zu überschreiben.

Konfigurationsmöglichkeit Web-Link

Name	Wert
Beschriftung	Anzeigename hinter dem Icon
Symbol	Icon, welches vor der Beschriftung angezeigt wird
URL	URL die geöffnet werden soll

1.7.3.4 Suchfeld

Das Schnellsuchfeld findet sich in der linken oberen Ecke des Hauptfensters. Dieses Feld ermöglicht den schnellen Zugriff auf Abfragen. Diese werden vom Administrator zur Verfügung gestellt oder auch vom Anwender hinzugefügt. Alle Abfragen, die hier verwendet werden, dürfen nur eine Suchzeichenkette oder keine Sucheingabe erwarten.

Keine Sucheingabe macht bei solchen Abfragen Sinn, deren Ergebnis sich von Zeit zu Zeit ändert. Das Ausführen einer solchen Suche im Schnellsuchfeld zeigt dann das aktuelle Ergebnis, ohne dass man die entsprechende Abfrage jedes Mal beispielsweise in einem Ordner aufsuchen muss. Beispielsweise könnte es eine Suchabfrage geben, die alle Lieder anzeigt, die der aktive Anwender schon gehört hat.

1.7.3.4.1 Suchfeldkonfiguration für Administratoren

Die "Suchfeld"-Konfiguration legt fest, welche Abfragen vom Administrator im Schnellsuchfeld des Knowledge-Builders zur Verfügung gestellt werden.

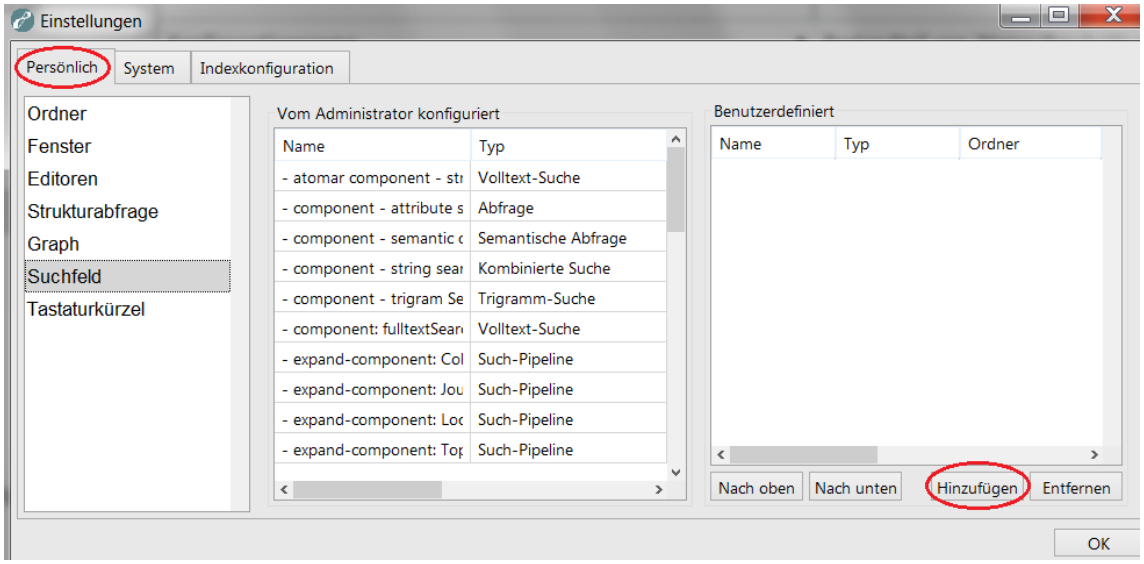
Neu angelegte Netze verfügen über eine Suchfeld-Konfiguration, die für alle Nutzer gleich ist. Der Administrator kann diese Suchfeld-Konfiguration erweitern, um allen Nutzern weitere Abfragen zugänglich zu machen. Zusätzlich kann jeder Nutzer seinem Schnellsuchfeld weitere Abfragen hinzufügen, die dann aber nur für ihn persönlich sichtbar sind.

Eine Suchfeld-Konfiguration besteht aus "Schnellsuchelementen", die eine Referenz auf eine Abfrage enthalten müssen und optional mit einer Beschriftung versehen werden können. Die Reihenfolge der Schnellsuchelemente bestimmt die Reihenfolge der Menüeinträge am Schnellsuchfeld.

1.7.3.4.2 Suchfeldkonfiguration für Anwender

Der Anwender kann Abfragen durch ziehen einer existierenden Abfrage auf das Schnellsuchfeld hinzufügen.

Das Hinzufügen kann ebenfalls über die *Einstellungen* erfolgen. Auf dem Reiter *Persönlich* befindet sich der Punkt *Suchfeld*. Im rechten Bereich im Abschnitt *Benutzerdefiniert* steht neben dem *Hinzufügen* auch die Operationen *Entfernen* und die Möglichkeit die Reihenfolge zu ändern zur Verfügung.



1.7.4 Style

Aufgabe der View-Konfiguration ist die strukturelle Aufbereitung von Elementen des semantischen Modells für die Anzeige. Geht es darüber hinaus um die Festlegung rein optischer Eigenschaften bzw. kontextloser Informationen, wird das sogenannte "Style"-Element verwendet.

1.7.4.1 Style-Eigenschaften in Anwendungen

Es gibt eine Reihe von Style-Elementen, die bereits in i-views definiert sind. Um welche Elemente es sich handelt und wie diese Style-Elemente im Knowledge-Builder angelegt werden, sodass sie dann mit einzelnen Elementen der View-Konfiguration einer Anwendung verknüpft werden können, wird im Folgenden erläutert.

Zunächst muss das Element der View-Konfiguration ausgewählt werden, mit dem wir ein oder mehrere Style-Elemente verknüpfen wollen. Je nach Typ des View-Konfiguration-Elements

stehen verschiedene Reiter zur Konfiguration der Styles zur Verfügung ("Aktionen und Styles" -> "Styles" oder direkt "Styles"). Diesen Reiter wählen wir aus und können dann entweder ein neues Style-Element definieren oder ein bereits vorhandenes Style-Element verknüpfen. Wenn wir ein neues Style-Element definieren, müssen wir diesem zuerst einen Konfigurationsnamen geben. Auf der rechten Seite des Editors kann daraufhin die Konfiguration vorgenommen werden.

Im Folgenden werden die einzelnen Konfigurationsmöglichkeiten für ein Style-Element erläutert:

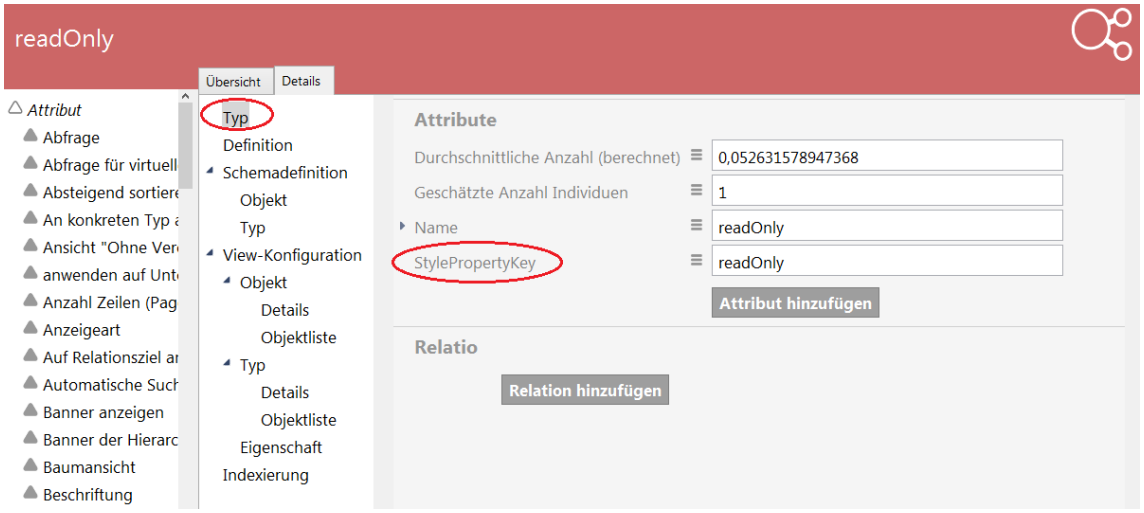
Name	**Attributtyp**	**StylePropertyKey**	**Konftyp**	**Beschreibung**
actionConfirmation	Verweis auf Skript	actionConfirmation		
actionConfirmation	Zeichenkette	actionConfirmation		
actionConfirmationTitle	Zeichenkette	actionConfirmationTitle		
actionType	Zeichenkette	actionType		
class	Zeichenkette	class		
class (skript)	Verweis auf Skript	class		
collapsed	Boolean	collapsed		

Datumsformat	Zeichenkette	Datumformat		
Do not print	Boolean	doNotPrint		
Download request	Zeichenkette	downloadRequest		
Element ID	Zeichenkette	elementId		
extra	Zeichenkette	downloadRequest		
group column grid	Zeichenkette	groupColumnGrid	Gruppe	Als Eingabe wird ein String mit Zahlen erwartet, die durch ein Leerzeichen oder ein Komma getrennt werden. Jede Zahl definiert die Anzahl der Spalten, wenn das Maximum 12 Spalten beträgt.
hide filters	Boolean	hideFilters		
hierarchyDetached	Boolean	hierarchyDetached		
href	Zeichenkette	href		
Label anzeigen	Boolean	hideLabel		
Panel	Zeichenkette	panel		
renderMode	Auswahl	headerMode		
renderMode	Zeichenkette	renderMode		

schreibgeschützt	Boolean	readOnly	Eigenschaften	Die Eigenschaften des View-Konfiguration-Elements können in der Anwendung nur gelesen und nicht bearbeitet werden. Auch ein "Bearbeiten-Button" wird darum nicht angezeigt.
vcmStateContext	Auswahl	vcmStateContext		global, none or page
vcmStateContext	Zeichenkette	vcmStateContext		
vcmTruncate	Ganzzahl	vcmTruncate		
Zahlenformat	Zeichenkette	numberFormat		
Zeichenkette für Konfigurationswert	Zeichenkette	style		
Skript für Konfigurationswert	Verweis auf Skript	style		
Zeilen markieren	Boolean	vcmMarkRowClick		
Ziel	Zeichenkette	class		
Zusätzliche Datumsformate	Zeichenkette	extraDatumFormats		

Falls wir ein neues Style-Element definiert haben, muss noch ein letzter Schritt vorgenommen werden. Damit die vorgenommene Konfiguration auch in die JSON-Ausgabe geschrieben wird, muss bei dem entsprechenden Konfigurationselement im Schema noch eine Ergänzung vorgenommen werden. Zum Schema gelangt man, indem man im Menü ☰ des entsprechenden Konfigurationselements auf "Schema" klickt. Im Schema muss man dann unter "Details"

-> "Typ" das Attribut "StylePropertyKey" hinzufügen und dort den Namen des Konfigurationselements eingeben.

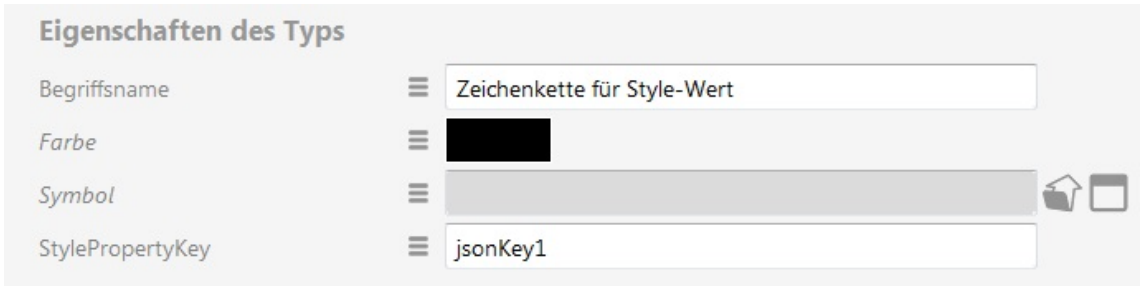


In der JSON-Ausgabe werden dann die Schlüssel-Wert-Paare (*StylePropertyKey* -> Style-Eigenschaft) als Array unter dem Schlüssel *additionalConfig* rausgeschrieben.

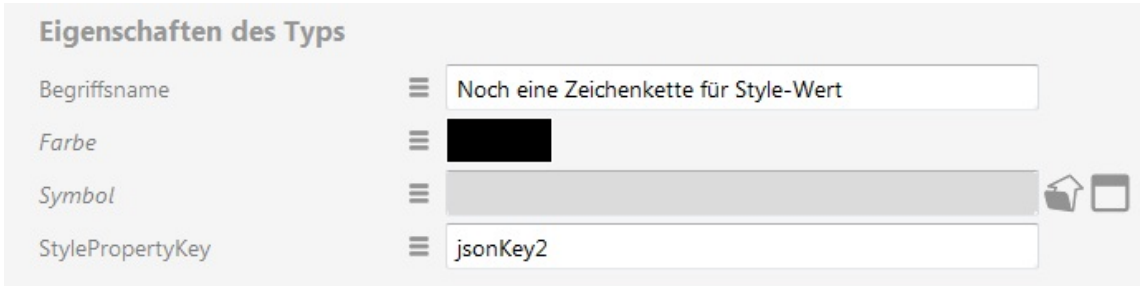
Auch ganz eigene Style-Eigenschaften können so definiert werden.

Beispiel

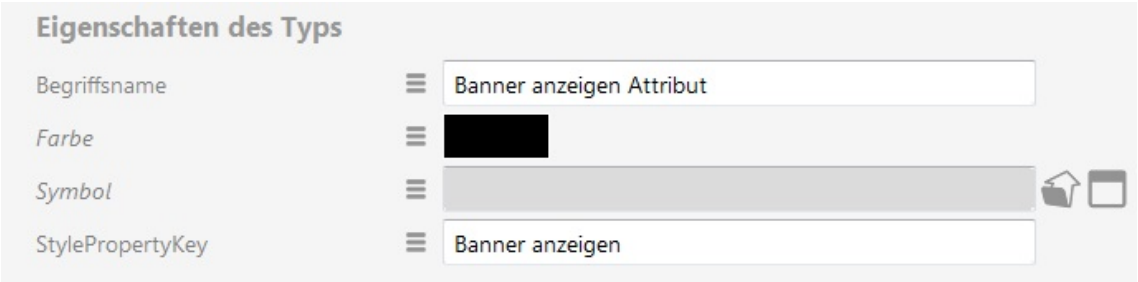
Konfiguration des Typs *Zeichenkette für Style-Wert*



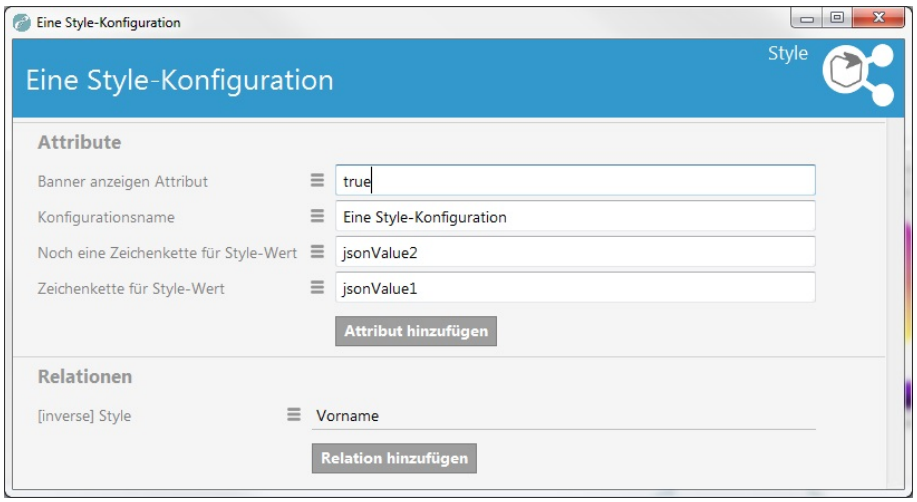
Konfiguration des Typs *Noch eine Zeichenkette für Style-Wert*



Konfiguration des Typs *Banner anzeigen Attribut*



Konfiguration des Objektes *Eine Style-Konfiguration* vom Typ *Style*



JSON-Ausgabe:

```
"properties": [{
    "values": [{ ... }],
    "label": "Vorname",
    "additionalConfig": {
        "jsonKey1": ["jsonValue1"],
        "jsonKey2": ["jsonValue2"],
        "Banner anzeigen": ["true"]
    },
    "viewId": "ID34304_461524079",
    "schema": { ... }
}
```

1.7.4.2 Style-Eigenschaften im Knowledge-Builder

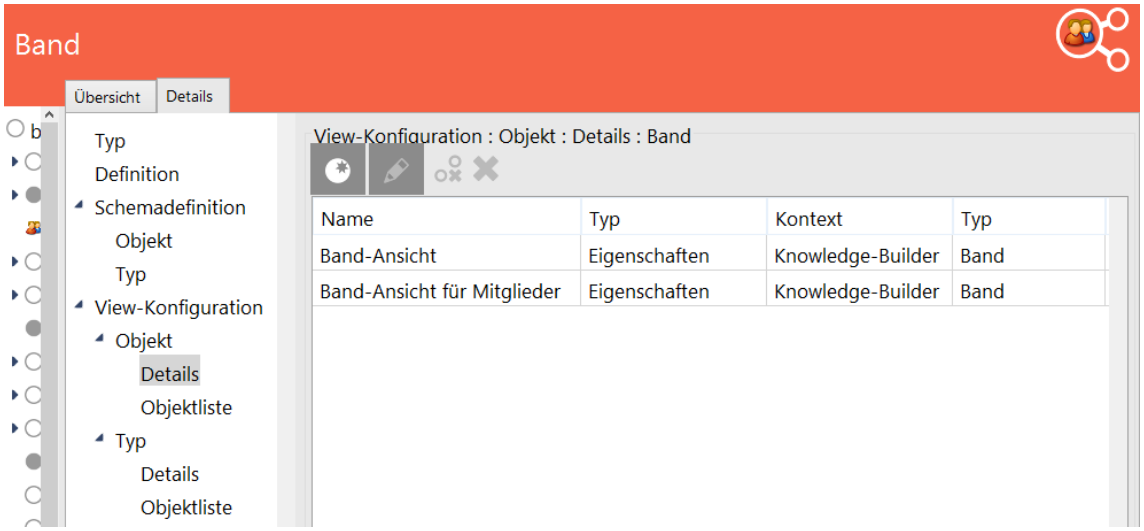
Im Knowledge-Builder bestimmen die Eigenschaften eines Styles über die Darstellung einiger Benutzungsoberflächenelemente. Nicht alle möglichen Eigenschaften eines Styles ergeben für alle Konfigurationen Sinn. Die folgende Tabelle zeigt, welche Eigenschaften von welcher Konfiguration unterstützt werden und was der Effekt ist.

Style-Eigenschaft	Konfigurationstyp	Effekt
Banner anzeigen	Objektkonfiguration	Banner mit Namen des Objekts und Schaltflächen zur Bearbeitung anzeigen. Standard ist *nein*.
Baumansicht	Gruppe	Elemente der Gruppe als Baum darstellen. Standard ist *nein*, d.h. die Gruppenelemente werden nebeneinander oder untereinander angezeigt.
Editorbreite (Pixel)	Eigenschaft	Breite in Pixeln einer Eigenschaft
Vertikale Anordnung	Gruppe	Elemente der Gruppe nebeneinander darstellen. Standard ist *untereinander* anzeigen.
Höhe	Gruppe	Höhe eines Gruppenelements in Pixeln bei nebeneinander dargestellten Gruppenelementen.
Höhe	Eigenschaft	Höhe in Zeilen bei Zeichenkettenattributen
Meta-Eigenschaften im Kontextmenü einblenden	(Meta-) Eigenschaft(en)	Metaeigenschaften werden im Kontextmenü der Eigenschaft eingeblendet. Somit kann man entweder einzelne Meta-Eigenschaften oder alle Meta-Eigenschaften einer Meta-Eigenschaftenkonfiguration einblenden. (Anmerkung: Der Menüpunkt *Metaeigenschaften hinzufügen* bleibt dabei unverändert).
Vorschau anzeigen	Tabelle	Steuert, ob unterhalb der Tabelle ein Editor angezeigt wird.

1.7.5 Detektorsystem zur Ermittlung der View-Konfiguration

Mit Hilfe des Detektorsystems können View-Konfigurationen an Bedingungen geknüpft werden. Das Detektorsystem bestimmt wann welche Konfiguration angezeigt werden soll. Im Folgenden wird die Funktionsweise des Detektorsystems und das Zusammenspiel mit View-Konfigurationen an einem Beispiel erläutert.

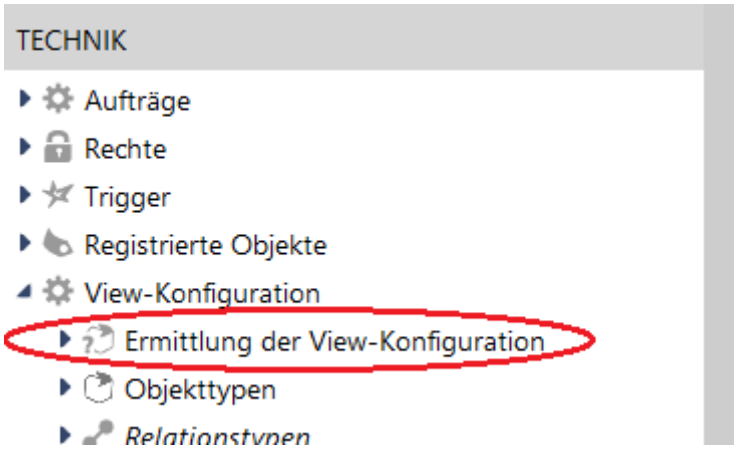
Für Objekte eines Objekttyps können, über Einstellungen in der View-Konfiguration, mehrere Anzeigen erstellt werden. Mit Hilfe des Detektorsystems können diese an Bedingungen geknüpft werden - wie beispielsweise an einen bestimmten Benutzer. Für das hier beschriebene Beispiel wurden für die Objekte eines beliebigen Typs mit Hilfe der View-Konfiguration zwei Ansichten konfiguriert.



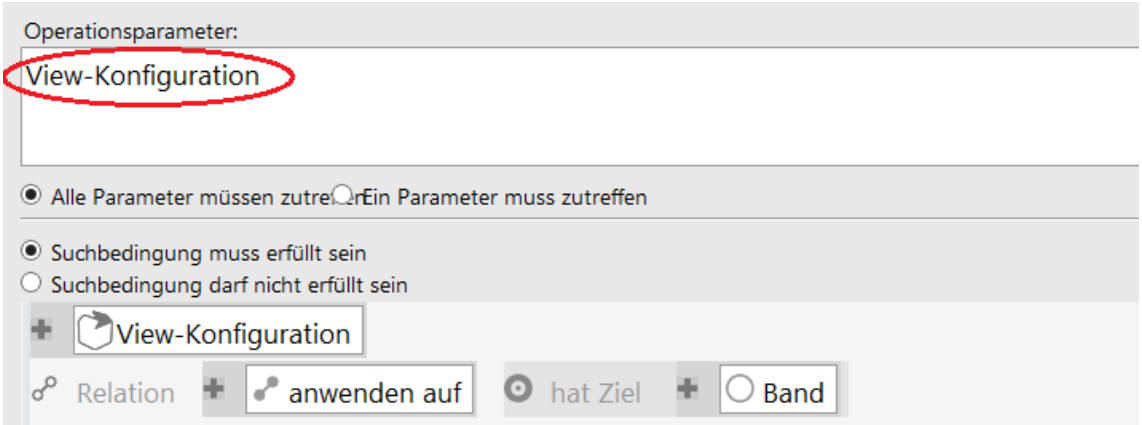
Benutzer, die Mitglied in der Band sind, auf die sie zugreifen wollen, sollen die "Band-Ansicht für Mitglieder" sehen. Alle Benutzer, die nicht Mitglied in der Band sind, auf die sie zugreifen wollen, sollen die "Band-Ansicht" sehen. Die Bedingungen, nach denen die Ansichten benutzt werden sollen, werden im Detektorsystem definiert.

Erstellung einer View-Konfigurations-Ermittlung

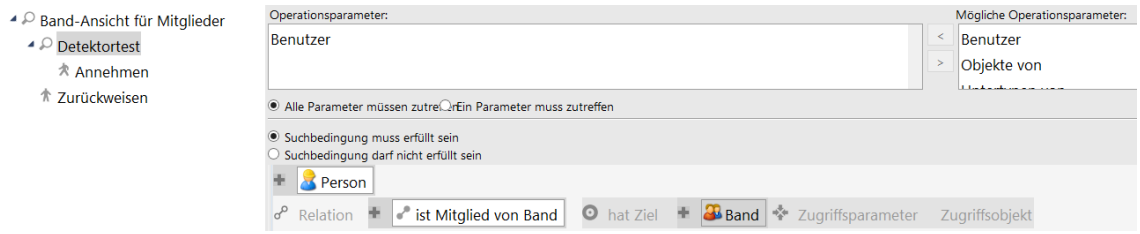
Das Detektorsystem befindet sich in der linken Ordnerhierarchie unter dem Abschnitt "Technik" und ist mit der Bezeichnung "Ermittlung der View-Konfiguration" unter "View-Konfiguration" versehen.



Im erste Schritt muss, über das Anlegen eines neuen Suchfilters (siehe Kapitel Suchfilter), der Ausgangspunkt definiert werden - das heißt, es muss definiert werden, wofür die noch folgenden Einstellungen gelten sollen. In diesem Beispiel ist unser Ausgangspunkt daher eine View-Konfiguration (hier: "Band-Ansicht für Mitglieder"), für die gleich eine Bedingung angelegt wird. Als Operationsparameter muss "View-Konfiguration" aus der Liste ausgewählt und eingetragen werden. Der Suchfilter sieht dann folgendermaßen aus:



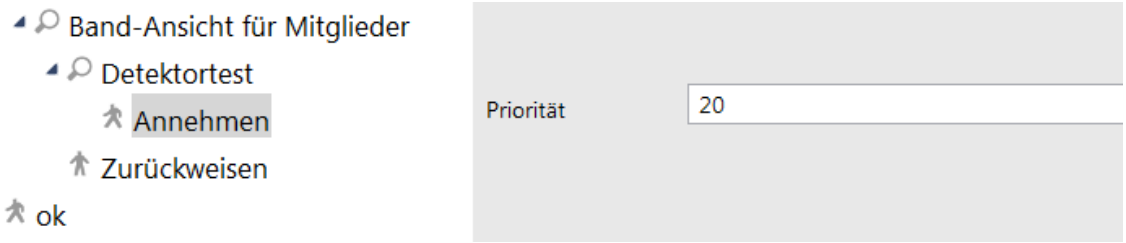
Unter dem Suchfilter, der nach der View-Konfiguration "Band-Ansicht für Mitglieder" sucht, muss nun ein neuer Suchfilter angelegt werden, der die Bedingung für diese View-Konfiguration beschreibt: Die View-Konfiguration "Band-Ansicht für Mitglieder" soll nur für Benutzer sichtbar sein, die Mitglieder der Band sind, die sie sich gerade ansehen. Der zweite Suchfilter prüft also, ob der aktive Benutzer ein Mitglied der Band ist. Über einen Klick auf wird der Menge der Suchergebnisse dann erlaubt, die Konfiguration "Band-Ansicht für Mitglieder" einzusehen. Die folgende Abbildung zeigt, den Suchfilter nach Benutzern, die Mitglied der Band sind, die sie sich gerade ansehen und die Ordnerhierarchie, die auf der linken Seite bisher aufgebaut wurde.



Die View-Konfiguration "Band-Ansicht" wird automatisch für diejenigen Nutzer verwendet, die nicht Mitglied der Band sind, die sie sich gerade ansehen.

Gewichtung der Konfigurationen im Detektorsystem

Die Konfigurationen im Detektorsystem "Ermittlung der View-Konfiguration" werden in der Anwendung von oben nach unten gewichtet. Das heißt, Zugangseinstellungen die weiter oben vorgenommen wurden, wiegen mehr als jene weiter unten. Um diese Standardeinstellung zu umgehen, können den Berechtigungen bzw. Verweigerungen Prioritäten gegeben werden.



Dabei ist Priorität 1 die höchste Priorität. Gibt es bei den Bedingungsanweisungen Überschneidungen, so wird die Berechtigungs- bzw. Verweigerungsbedingung mit der höchsten Priorität durchgesetzt. Sind keine Prioritätsangaben gemacht oder haben alle Prioritätszahlen den gleichen Wert, so wird die frühere Bedingungen im Detektorbaum durchgesetzt.

1.8 JavaScript-API

1.8.1 Einführung

Die JavaScript-API ist eine serverseitige API für semantische Netze. Sie wird u.a. in Triggern, REST-Services und Berichten verwendet.

Mit der API kann man sowohl lesend auf das Wissensnetz zugreifen (Suchen ausführen, Eigenschaften abfragen usw.) als auch Änderungen vornehmen (neue Objekte anlegen, Attribute ändern usw.).

Beispiel: Neues Objekt vom Typ "Person" erzeugen

```
var person = $k.Registry.type("Person").createInstance();
person.setAttributeValue("familyName", "Sinatra");
person.setAttributeValue("firstName", "Frank");
```

Beispiel: Artikel-Objekte mit Parameter "tag" = "Jazz" suchen und als JSON ausgeben

```
var articles = $k.Registry.query("articles").findElements({tag: "Jazz"});
var json = articles.map(function(element) {
    return {
        name: element.name(),
        type: element.type().name()
    }
});
return JSON.stringify(json)
```

1.8.1.1 API-Referenz

Die API-Referenz ist unter folgender Adresse erreichbar:

http://documentation.i-views.com/k-infinity-api/4.3/kinf-js-api

1.8.1.2 Der Namespace $k

Die meisten API-Objekte und -Funktionen sind im Namensraum $k definiert. Der Namensraum selbst definiert einige hilfreiche Funktionen, z.B.

```
$k.rootType()
```

das den Wurzeltyp des Netzes zurückgibt, oder

```
$k.user()
```

das den aktiven Benutzer zurückgibt.

1.8.1.3 Registry

Ein weiteres wichtiges Objekt ist das Registry-Objekt $k.Registry. Es erlaubt den Zugriff auf registrierte Ordnerelemente und Objekttypen.

Beispiele:

```
$k.Registry.type("Article")
```

gibt den Typ mit dem internen Namen "Article" zurück.

```
$k.Registry.query("articles")
```

gibt die Abfrage mit dem Schlüssel "articles" zurück.

Das Registry-Objekt ist ein Singleton, ähnlich wie das Math-Objekt von JavaScript.

1.8.1.4 Mit semantischen Elementen arbeiten

Auf Wissensnetzelemente greift man normalerweise über die Registry oder Abfragen zu.

```
// Personen-Typ anhand seines internen Namen ermitteln
var personType = $k.Registry.type("Person");


// Suche "articles" mit dem Suchparameter "tag" = "Vocal"
var sailingArticles = $k.Registry.query("articles").findElements({tag: "Vocal"});
```

Auf die Eigenschaften eines Elements kann man über ihren internen Namen zugreifen:

```
// Wert des Attributs "familyName"
var familyName = person.attributeValue("familyName");
// Ziel der Relation "bornIn"
var birthplace = person.relationTarget("bornIn");
```

Die Funktion name() liefert den Wert des Namensattributs:

```
var name = birthplace.name();
```

Bei übersetzten Attributen kann die Sprache als 2- oder 3-stelliger ISO 639 Sprachcode angegeben werden. Ohne Sprachangabe wird die aktuelle Sprache der Umgebung verwendet.

```
var englishTitle = book.attributeValue("title", "en");
var swedishTitle = book.attributeValue("title", "swe");
var currentTitle = book.attributeValue("title");
```

1.8.1.5 Transaktionen

Zum Anlegen, Ändern oder Löschen von Wissensnetzelementen wird eine Transaktion benötigt. Falls die Transaktionssteuerung durch das Script kontrolliert wird, kann man einen Block in eine Transaktion kapseln:

```
$k.transaction(function() {
    return $k.Registry.type("Article").createInstance();
});
```

Man kann konfigurieren, ob ein Script die Transaktionssteuerung übernimmt, oder ob das gesamte Script in einer Transaktion ausgeführt werden soll. Ausgenommen sind davon Trigger-Skripte, die immer als Teil der schreibenden Transaktion ausgeführt werden.

Bei Nebenläufigkeitskonflikten wird eine Transaktion vom Server zurückgewiesen. In diesem Falls wird eine optionale Callback-Funktion aufgerufen, die man als Argument an $k.transaction() übergibt:

```
$k.transaction(
    function() { return $k.Registry.type("Article").createInstance() },
    function() { throw "The transaction was rejected" }
);
```

1.8.1.6 Elemente modifizieren

1.8.1.6.1 Elemente anlegen

```
// Neues Objekt vom Typ "Person" erzeugen
var person = $k.Registry.type("Person").createInstance();
```

```
// Einen neuen Typ erzeugen
var blogType = $k.Registry.type("CommunicationChannel").createSubtype();
blogType.setName("Blog");
```

1.8.1.6.2 Attribute hinzufügen und ändern

Attributwerte können mit setAttributeValue() gesetzt werden. Es wird entweder der Wert des bestehenden Attributs geändert oder ein neues Attribut angelegt, falls noch kein Attribut vorhanden ist. Sollten bereits mehrere Attribute vorhanden sein wird eine Exception geworfen.

```
person.setAttributeValue("familyName", "Sinatra");
person.setAttributeValue("firstName", "Frank");
// Überschreibe den Attributwert "Frank" mit "Francis""
person.setAttributeValue("firstName", "Francis");
```

Mit createAttribute() können mehrere Attribute desselben Typs hinzugefügt werden, da bestehende Attribute nicht überschrieben werden:

```
// Zwei Attribute anlegen
person.createAttribute("nickName", "Ol' Blue Eyes");
person.createAttribute("nickName", "The Voice");
```

Die Attributwerte werden je nach Attributart durch unterschiedliche Objekte repräsentiert:

Art des Attributs	Objekttyp
Auswahl	$k.Choice
Boolesch	Boolean
Datei	$k.Blob
Datum	$k.Date
Datum und Uhrzeit	$k.DateTime
Farbwert	String (Hexwert)
Flexible Zeit	$k.FlexDateTime
Fließkommazahl	Number
Ganzzahl	Number
Geographische Position	$k.GeoPosition
Gruppe	- (kein Wert)
Internet-Verknüpfung (URL)	String
Intervall	$k.Interval
Zeichenkette	String
Zeit	$k.Time

1.8.1.6.3 Relationen hinzufügen

Mit createRelation() kann eine Relation zwischen zwei Elementen angelegt werden:

```
var places = $k.Registry.query("places").findElements({name: "Hoboken"});
if (places.length == 1)
    person.createRelation("bornIn", places[0]);
```

1.8.1.6.4 Elemente löschen

Elemente können mit der Funktion remove() gelöscht werden:

```
person.remove();
```

Dabei werden auch alle Eigenschaften des Elements gelöscht.

1.8.2 Beispiele

1.8.2.1 Abfragen

Suche nach Elementen:

```
// Abfrage "articles" mit dem Parameter tag = "Soccer" ausführen
var articles = $k.Registry.query("articles").findElements({tag: "Soccer"});
for (var a in articles)
    $k.out.print(articles[a].name() + "\n");
```

Suche nach Hits. Ein Hit kapselt ein Element und fügt einen Qualitätswert (zwischen 0 und 1) sowie weitere Metainformationen hinzu.

```
// Abfrage "mainSearch" mit dem Suchwert "Baseball" ausführen
var hits = $k.Registry.query("mainSearch").findHits("Baseball");
hits.forEach(function(hit) {
    $k.out.print(hit.element().name() + " (" + (Math.round(hit.quality() * 100))+ "%)\n");
});
```

Suchergebnis in JSON umwandeln:

```
var elements= $k.Registry.query("articles").findElements({tag: "Snooker"});
var json = elements.map(function(element) {
    return {
        name: element.name(),
        id: element.idString(),
        type: element.type().name()
    }
});
$k.out.print(JSON.stringify(json, undefined, "\t"));
```

1.8.2.2 Zur Laufzeit generierte Abfragen

Die Javascript-API erlaubt es auch, Abfragen dynamisch zu generieren. Hier einige Beispiele aus einem Filmnetz:

Suche nach Filmen mit Jahr + Name

```
var query = new $k.StructuredQuery("imdb_film");
query.addAttributeValue("imdb_film_year", "year");
query.addAttributeValue("name", "name");
query.findElements({year: "1958", name: "Vert*"});
```

Dem Constructor wird die Domain übergeben. Bei internen Namen wird automatisch nach Objekten dieses Types gesucht. Mehr Möglichkeiten bietet die Funktion setDomains()

Jahr + Anzahl Regisseure >= 3

```
var query = new $k.StructuredQuery("imdb_film");
query.addAttributeValue("imdb_film_year", "year");
```

```
query.addCardinality("imdb_film_regisseur", 3, ">=");
query.findElements({year: "1958"});
```

Jahr + Name des Regisseurs

```
var query = new $k.StructuredQuery("imdb_film");
query.addAttributeValue("imdb_film_year", "year", ">=");
var directorQuery = query.addRelationTarget("imdb_film_regisseur").targetQuery();
directorQuery.addAttributeValue("name", "director");
query.findElements({year: "1950", director: "Hitchcock, Alfred"});
```

Alternativen (Oder-Bedingungen)

```
var query = new $k.StructuredQuery("imdb_film");
query.addAttributeValue("imdb_film_year", "year");
var alternatives = query.addAlternativeGroup();
alternatives.addAlternative().addAttributeValue("name", "name");
alternatives.addAlternative().addAttributeValue("imdb_film_alternativeTitel", "name");
query.findElements({year: "1958", name: "Vert*"});
```

1.8.2.3 Elemente anlegen und ändern

Eine Person anlegen

```
// Personen-Typ über seinen internen Namen ermitteln
var personType = $k.Registry.type("Person");
// Neue Person anlegen
var person = personType.createInstance();
// Attribuwerte setzen
person.setAttributeValue("familyName", "Norris");
person.setAttributeValue("firstName", "Chuck");
```

Den vollständigen Namen einer Person setzen

```
var familyName = person.attributeValue("familyName");
var firstName = person.attributeValue("firstName");
if (familyName && firstName)
{
    var fullName = familyName + ", " + firstName;
    person.setAttributeValue("fullName", fullName);
}
```

Attributwerte setzen

```
// Boolean
element.setAttributeValue("hasKeycard", true);

// Auswahl - Internen Name des Auswahlwerts angeben
element.setAttributeValue("status", "confirmed");
```

```
// Auswahl - Auswahlwert angeben
var choiceRange = $k.Registry.attributeType("status").valueRange();
var choice = choiceRange.choiceInternalNamed("confirmed");
element.setAttributeValue("status", choice);

// Farbe
element.setAttributeValue("hairColor", "723F10");

// Datum / Uhrzeit
element.setAttributeValue("dateOfBirth", new $k.Date(1984, 5, 4));
element.setAttributeValue("lastModification", new $k.DateTime());
element.setAttributeValue("teatime", new $k.Time(15, 30, 0));

// Flexible Zeit - $k.FlexTime (ermöglich ungenaue Zeitangaben)
element.setAttributeValue("start", new $k.FlexTime(1984, 6));
// Flexible Zeit - Date (fehlende Werte werden durch Standardwerte aufgefüllt)
element.setAttributeValue("start", new Date(1984, 5, 3));

// Zahl (Ganzzahl / Fließkomma)
element.setAttributeValue("weight", 73);

// Intervall
element.setAttributeValue("interval", new $k.Interval(2, 4));

// Zeichenkette - nicht übersetzt
element.setAttributeValue("familyName", "Norris");
// Zeichenkette - übersetzt
// Sprache wird als ISO 639-1 or 639-2b Code übergeben
element.setAttributeValue("welcomeMessage", "Welcome", "en");
element.setAttributeValue("welcomeMessage", "Bienvenue", "fre");
```

Neues Attribut anlegen

```
person.createAttribute("nickName", "Ground Chuck");
```

Neue Relation anlegen

```
var places = $k.Registry.query("places").findElements({name: "Oklahoma"});
if (places.length == 1)
    person.createRelation("bornIn", places[0]);
```

Ein Element samt Eigenschaften löschen

```
person.remove()
```

Eine Zeichenkette in einen Attributwert konvertieren. Der ValueRange eines Attributtyps kennt die erlaubten Werte und kann die Zeichenkette einlesen. Bei ungültigen Zeichenketten wird ein Fehler geworfen.

```
var statusRange = $k.Registry.type("status").valueRange();
var statusConfirmed = statusRange.parse("Confirmed", "eng");
```

Änderungs-Metadaten ändern

```
element.setAttributeValue("lastChangeDate", new Date());
var userInstance = $k.user().instance();
// Verknüpfung mit Benutzer?
if (element.relationTarget("lastChangedBy") !== userInstance)
{   // bestehende Relationen löschen
    var relations = element.relations("lastChangedBy");
    for (var r in relations)
        relation[r].remove();
    // Relation zum Benutzer-Objekt ziehen
    element.createRelation("lastChangedBy", userInstance);
}
```

1.8.2.4 REST

Ein REST-Script muss eine Funktion respond() definieren, die als Argumente die HTTP-Anfrage, die geparsten Anfrage-Parameter und eine leere HTTP-Antwort entgegennimmt. Das Script füllt dann Header und Inhalt der Antwort.

```
function respond(request, parameters, response)
{
    response.setText("REST example");
}
```

Einen Blob herunterladen

```
function respond(request, parameters, response)
{
    var name = parameters["name"];
    if (name)
    {
        var images = $k.Registry.query("rest.image").findElements({"name": name});
        if (images.length == 1)
        {
            // Inhalt und content type aus dem Blob übernehmen
            response.setContents(images[0].value());
            // Inline anzeigen
            response.setContentDisposition("inline");
        }
        else
        {
            response.setCode($k.HttpResponse.BAD_REQUEST);
            response.setText(images.length + " images found");
    }
    }
    else
    {
        response.setCode($k.HttpResponse.BAD_REQUEST);
        response.setText("Name not specified");
```

```
    }
}
```

Neues Objekt mit einem hochgeladenen Blob anlegen

```
function respond(request, parameters, response)
{
    var formData = request.formData();
    var name = formData.name;
    var picture = formData.picture;
    if (name && picture)
    {
        var city = $k.Registry.type("City").createInstance();
        city.setAttributeValue("image", picture);
        city.setName(name);
        response.setText("Created city " + name);
    }
    else
    {
        response.setCode($k.HttpResponse.BAD_REQUEST);
        response.setText("Parameters missing");
    }
}
```

1.8.2.5 XML

Suchergebnisse in XML transformieren

```
function respond(request, parameters, response)
{
    var name = parameters["name"];
    if (name)
    {
        // Find points of interest
        var pois = $k.Registry.query("rest.poi").findElements({name: name});
        // Write XML
        var document = new $k.TextDocument();
        var writer = document.xmlWriter();
        writer.startElement("result");
        for (var p in pois)
        {
            writer.startElement("poi");
            writer.attribute("name", pois[p].name());
            writer.endElement();
        }
        writer.endElement();
        response.setContents(document);
        response.setContentType("application/xml");
    }
    else
    {
        response.setCode($k.HttpResponse.BAD_REQUEST);
```

```
        response.setContents("Name not specified");
    }
}
```

XML-Ausgabe

```
<result>
    <poi name="Plaza Mayor"/>
    <poi name="Plaza de la Villa"/>
    <poi name="Puerta de Europa"/>
</result>
```

Qualifizierte Namen verwenden

```
var document = new $k.TextDocument();
var writer = $k.out.xmlWriter();
writer.setPrefix("k", "http://www.i-views.de/kinfinity");
writer.startElement("root", "k");
writer.attribute("hidden", "true", "k");
writer.startElement("child","k").endElement();
writer.endElement();
```

XML-Ausgabe

```
<k:root xmlns:k="http://www.i-views.de/kinfinity" k:hidden="true">
    <k:child/>
</k:root>
```

Standard-Namespace definieren

```
var document = new $k.TextDocument();
var writer = $k.out.xmlWriter();
writer.startElement("root");
writer.defaultNamespace("http://www.i-views.de/kinfinity");
writer.startElement("child").endElement();
writer.endElement();
```

XML-Ausgabe

```
<root xmlns="http://www.i-views.de/kinfinity">
    <child/>
</root>
```

1.8.2.6 HTTP-Client

In einem Script können auch HTTP-Requests abgeschickt werden.

Beispiel: Bild über HTTP laden und als Blob im Wissensnetz speichern

```
var http = new $k.HttpConnection();
var imageUrl = "http://upload.wikimedia.org/wikipedia/commons/e/e7/2007-07-06_GreatBriain_Portree.
var imageResponse = http.getResponse(new $k.HttpRequest(imageUrl));
if (imageResponse && imageResponse.code() == $k.HttpResponse.OK )
{
    var portree = $k.Registry.type("City").createInstance();
    portree.setAttributeValue("image", imageResponse);
    portree.setName("Portree");
}
```

Wetterberichte aller Städte aktualisieren

```
var instances = $k.Registry.type("City").instances();
var http = new $k.HttpConnection();
for (var i in instances)
{
    var city = instances[i];
    var weatherUrl = "http://api.openweathermap.org/data/2.5/weather";
    var weatherRequest = new $k.HttpRequest(weatherUrl);
    weatherRequest.setQueryData({q: city.name()});
    try {
        var weatherResponse = http.getResponse(weatherRequest);
        if (weatherResponse.code() == $k.HttpResponse.OK)
        {
            var json = JSON.parse(weatherResponse.text());
            var weather = json.weather[0].description;
            city.setAttributeValue("weather", weather);
        }
    } catch (e) {
    }
}
```

Um zu verhindern, dass Skripte Requests zu beliebigen Hosts schicken, kann man in der Konfigurationsdatei einer Anwendung eine Whitelist definieren:

```
[script]
allowedOutgoingDomains=*.i-views.de,*.intelligent-views.com,ivinternal:8080
```

Die durch Komma getrennten Zeichenketten werden mit der Domain der URL verglichen, "*" als Wildcard ist dabei erlaubt. Optional kann auch ein Port angegeben werden. Falls Domain oder Port nicht passen wird bei der Ausführung des Requests ein URIError geworfen. Ohne Angabe des Port ist jeder Port gültig.

1.8.2.7 E-Mails versenden

E-Mails können mit dem MailMessage-Objekt verschickt werden. Dazu muss im Netz ein SMTP-Server konfiguriert werden (Einstellungen -> System -> SMTP).

```
var mail = new $k.MailMessage();
mail.setSubject("Hello from " + $k.volume());
mail.setText("This is a test mail");
mail.setSender("kinfinity@example.org");
```

```
mail.setReceiver("developers@example.org");
mail.setUserName("kinf");
mail.send();
```

Das Benutzerkonto "kinf" wird für die Authentifizierung verwendet. Das Passwort wird in den SMTP-Einstellungen hinterlegt.

1.8.2.8 Views

JSON-Strukturen können auch anhand der View-Konfiguration generiert werden, sowohl für einzelne Objekte als auch Objektlisten.

Im einfachsten Fall wird ein Objekt anhand der Standard-Konfiguration ohne weiteren Kontext in JSON umgewandelt:

```
var data = element.renderJSON();
```

Es werden dann alle durch die Konfiguration definierten Strukturen in JSON umgesetzt:

```
{
    "viewType" : "fieldSet",
    "label" : "Bern",
    "elementType" : "instance",
    "modNum" : 26,
    "elementId" : "ID17361_141538476",
    "type" : {
        "elementType" : "instance",
        "typeId" : "ID10336_319205877",
        "internalName" : "City",
        "typeName" : "Stadt"
    },
    "properties" : [{
            "values" : [{
                    "value" : "Bern",
                    "propertyId" : "ID17361_137824032"
                }
            ],
            "schema" : {
                "label" : "Name",
                "elementType" : "attribute",
                "internalName" : "name",
                "maxOccurrences" : 1,
                "attributeType" : "string",
                "viewId" : "ID20838_426818557",
                "typeId" : "ID4900_317193164",
                "minOccurrences" : 0
            }
        }, {
            "values" : [{
                    "typeId" : "ID4900_79689320"
                }
            ],
            "schema" : {
```

```
                "label" : "Alternativname/Synonym",
                "elementType" : "attribute",
                "internalName" : "alternativeName",
                "attributeType" : "string",
                "rdf-id" : "alternativeName",
                "viewId" : "ID20839_64952366",
                "typeId" : "ID4900_79689320",
                "minOccurrences" : 0
            }
        }, {
            "values" : [{
                    "target" : {
                        "typeId" : "ID10336_493550611",
                        "label" : "Kunstmuseum Bern",
                        "elementId" : "ID17362_205182965"
                    },
                    "propertyId" : "ID17361_395925739"
                }, {
                    "target" : {
                        "typeId" : "ID10336_493550611",
                        "label" : "Schweizerische Nationalbibliothek",
                        "elementId" : "ID20401_126870015"
                    },
                    "propertyId" : "ID17361_9264966"
                }
            ],
            "schema" : {
                "targetDomains" : [{
                        "elementType" : "instance",
                        "typeId" : "ID10336_493550611",
                        "internalName" : "point_of_interest",
                        "typeName" : "Sehenswürdigkeit"
                    }
                ],
                "label" : "beherbergt Sehenswürdigkeit",
                "elementType" : "relation",
                "internalName" : "contains_poi",
                "viewId" : "ID20840_182208894",
                "typeId" : "ID2052_332207092",
                "minOccurrences" : 0
            }
        }
    ]
}
```


Zusätzlich kann ein Kontext in Form eines Anwendungs- oder Konfigurationsobjekts angegeben werden. Es wird dann ein zu diesem Kontext passende Konfiguration gewählt. Im folgenden Beispiel wird die Anwendung "Android" vorgegeben:

```
var application =  $k.Registry.elementAtValue("viewconfig.configurationName", "Android");
var data = element.renderJSON(application);
```

Es ist aber auch möglich, eine Konfiguration vorzugeben und diese das Element umwandeln zu lassen. Dazu erzeugt man eine $k.ViewConfiguration aus dem Konfigurationsobjekt.

```
var configurationElement = $k.Registry.elementAtValue("viewconfig.configurationName", "Android Art
var data = $k.ViewConfiguration.from(configurationElement).renderJSON(element);
```

Da die JSON-Struktur recht umfangreich ist, kann man auch bestimmte Properties bei der Umwandlung weglassen, indem man die Schlüssel als zusätzlichen Parameter angibt:

```
var application =  $k.Registry.elementAtValue("viewconfig.configurationName", "Android");
var data = element.renderJSON(application, ["rdf-id", "viewId", "typeId", "propertyId", "modNum",
```


```
{
    "viewType": "fieldSet",
    "label": "Bern",
    "elementType": "instance",
    "elementId": "ID17361_141538476",
    "type": {
        "elementType": "instance",
        "internalName": "City",
        "typeName": "Stadt"
    },
    "properties": [
        {
            "values": [
                {
                    "value": "Bern"
                }
            ],
            "schema": {
                "elementType": "attribute",
                "label": "Name",
                "internalName": "name",
                "attributeType": "string",
                "maxOccurrences": 1
            }
        },
        {
            "schema": {
                "elementType": "attribute",
                "label": "Alternativname/Synonym",
                "internalName": "alternativeName",
                "attributeType": "string"
            }
        },
        {
            "values": [
                {
                    "target": {
                        "label": "Kunstmuseum Bern",
                        "elementId": "ID17362_205182965"
```

```
                    }
                },
                {
                    "target": {
                        "label": "Schweizerische Nationalbibliothek",
                        "elementId": "ID20401_126870015"
                    }
                }
            ],
            "schema": {
                "elementType": "relation",
                "targetDomains": [
                    {
                        "elementType": "instance",
                        "internalName": "point_of_interest",
                        "typeName": "Sehenswürdigkeit"
                    }
                ],
                "label": "beherbergt Sehenswürdigkeit",
                "internalName": "contains_poi"
            }
        }
    ]
}
```

1.8.2.9 Mustache-Templates

Die folgende Funktion erzeugt ein Dokument mit Hilfe der Mustache Template-Bibliothek. Es erwartet folgendes Schema:

- ein Zeichenketten-Attribut (interner Name "template.id"), um das Template zu identifizieren
- ein Dateiattribut (interner Name "template.file") mit dem Template, z.B. ein HTML-Dokument
- Eine Relation zu einem MediaType_objekt (interner Name "template.contentType")

Die Abfrage "rest.articles" gibt alle Element zurück die dargestellt werden sollen. Die Mustache-Bibliothek ist unter "mustache.js" registriert.

```
function respond(request, parameters, response)
{
    // Mustache einbinden
    $k.module("mustache.js");

    // Template ermitteln
    var templateId = parameters["templateId"];
    var template = $k.Registry.elementAtValue("template.id", templateId);
    var templateText = template.attributeValue("template.file").text("utf-8");

    // Darzustellende Elemente suchen
```

```
    var elements = $k.Registry.query("rest.articles").findElements(parameters);

    // Template-Parameter vorbereiten
    var templateParameters = {
                data: elements.map(function(element) {
                        return {
                            name: element.name(),
                            type: element.type().name()
                        }})
        };

    // Dokument mit Mustache erstellen
    var output = Mustache.render(templateText, templateParameters);

    // Dokument ausliefern
    response.setText(output);
    response.setContentType(template.relationTarget("template.contentType").name());
}
```

1.8.2.10 Java Native Interface

Mit Hilfe von JNI (Java Native Interface) können Java-Bibliotheken eingebunden werden.

Warnung: JNI ist ein experimentelles Feature und hat einige Einschränkungen:

- JNI kann nicht in Triggern verwendet werden
- Es ist nicht möglich, Klassen zu definieren (beispielsweise für Callbacks)
- Generics werden nicht unterstützt
- JNI erlaubt den Zugriff auf Systemressourcen (z.B. Dateien)

JNI muss in den Konfigurationsdateien aller Anwendungen eingerichtet werden, die diese Skripte verwenden. Der Klassenpfad kann nicht zur Laufzeit geändert werden.

```
[JNI]
classPath=tika\tika-app-1.5.jar
libraryPath=C:\Program Files\Java\jre7\bin\server\jvm.dll
```

Mit Hilfe der Funktion $jni.use() wird eine Liste von Klassen importiert. Für jede Klasse wird ein gleichnamiges Funktionsobjekt angelegt, das mit new instanziiert werden kann. Außerdem werden alle statischen Eigenschaften an dieses Funktionsobjekt übertragen. Der Namensraum der Java-Klassen kann optional auch weggelassen werden.

Beispiel

```
// Import the StringBuilder class, without namespace
$jni.use(["java.lang.StringBuilder"], false);
// Create a new instance
var builder = new StringBuilder();
```

```
// Javascript primitives and Strings are automatically converted
builder.append("Welcome to ");
builder.append($k.volume());
// toJS() converts Java objects to Javascript objects
$k.out.print(builder.toString().toJS());
```

Text/Metadaten-Extraction mit Apache Tika

```
$jni.use([
        "java.io.ByteArrayInputStream",
        "java.io.BufferedInputStream",
        "java.io.StringWriter",
        "org.apache.tika.parser.AutoDetectParser",
        "org.apache.tika.metadata.Metadata",
        "org.apache.tika.parser.ParseContext",
        "org.apache.tika.sax.BodyContentHandler"
    ], false);
// Get a blob
var blob = $k.Registry.elementAtValue("uuid", "f36db9ef-35b1-48c1-9f23-1e10288fddf6").attributeVal
// Blobs have to be explicitely converted to Java byte arrays
var bufferedInputStream = new BufferedInputStream(new ByteArrayInputStream($jni.toJava(blob)));
// Parse the blob
try {
    var parser = new AutoDetectParser();
    var writer = new StringWriter();
    var metaData = new Metadata();
    parser.parse(bufferedInputStream, new BodyContentHandler(writer), metaData, new ParseContext()
    var string = writer.toString().toJS();
    // Print extracted metadata
    var metaNames = metaData.names().toJS().sort(
        function(a,b) { return a.localeCompare(b) });
    for (n in metaNames)
        $k.out.print(metaNames[n] + " = " + metaData.get(metaNames[n])).cr();
    // Print extracted text (first 100 chars)
    $k.out.cr().cr().print(string.substring(1, 100) + " [...]\n\n(" + string.length + " chars)");
}
catch (e)  {
    $k.out.print("Extraction failed: " + e.toString());
} finally {
    bufferedInputStream.close();
}
```

1.8.3 Module

1.8.3.1 Module definieren

Ein Modul wird mit der Funktion define() definiert. Als Argument übergibt man entweder ein Modulobjekt oder eine Funktion, die ein Modulobjekt zurückgibt. Ein Script sollte nur ein Modul definieren.

Beispiel: Modul mit einer Funktion jsonify()

```
$k.define({
    // Create a JSON object array for the elements
    jsonifyElements: function(elements) {
        return elements.map(function(element) {
            return {
                name: element.name(),
                id: element.idString(),
                type: element.type().name()
            };
        });
    }
});
```

Module können auch von anderen Modulen abhängig sein. Das folgende Script definiert ein Modul, das ein anderes ("rest.common") verwendet.

```
$k.define(["rest.common"], function(common) {
    return {
        stringifyElements: function(elements) {
            return JSON.stringify(common.jsonifyElements(elements), undefined, "\t")
        }
    }});
```

1.8.3.2 Module verwenden

Ein Modul kann entweder mit require() oder module() verwendet werden.

require() erwartet einen Array von Modulnamen und eine Callback-Funktion. Die Callback-funktion wird mit den Modulen ausgeführt.

```
var elements = $k.Registry.query("rest.poi").findElements({name: "Madrid"});
var json = $k.require(["rest.common"], function(common) {
     return common.jsonifyElements(elements);
});
```

module() erwartet den Namen eines Moduls und gibt das Modulobjekt zurück.

```
var json = $k.module("rest.common").jsonifyElements(elements);
```

module() kann auch mit Skripten verwendet werden, die keine Module definieren. Das Script wird ausgeführt und alle deklarierten Funktionen instanziiert. Diese Funktionen können anschließend aufgerufen werden.

1.8.3.3 AMD

Um JavaScript-Bibliotheken einzubinden, die den AMD-Standard unterstützen, muss man vorher define() und require() global definieren:

```
this.define = $k.define;
this.define.amd = {};
this.require = $k.require;
```

Falls eine Bibliothek ein Modul mit einer bestimmten ID definiert und man diese Bibliothek unter einem anderen Namen registrieren möchte, kann man Module-IDs auf Registratur-IDs abbilden:

```
$k.mapModule("underscore", "lib.underscore");
```

Nun kann man underscore.js als "lib.underscore" registrieren und das fort definierte Modul "underscore" verwenden.

1.8.4 Debugger

1.8.4.1 Debuggen im Knowledge-Builder

Das folgende Script kann zum Testen von Restlet-Skripten im Knowledge-Builder verwendet werden. In der Skriptumgebung bearbeitet man dazu auf dem Reiter "Skript ausführen" das zusätzliche Test-Skript.

```
// Request und Response vorbereiten
var testRequest = new $k.HttpRequest("http://localhost");
var testResponse = new $k.HttpResponse();
// Restlet-Funktion respond() mit den Test-Parametern ausführen,
// die als Property k.testbenchParameters verfügbar sind
respond(testRequest, $k.testbenchParameters, testResponse);
// Response ausgeben
$k.out.print(testResponse.debugString());
```

Auf dem Reiter "Debug" kann man Breakpoints setzen und das Script in Einzelschritten ausführen.

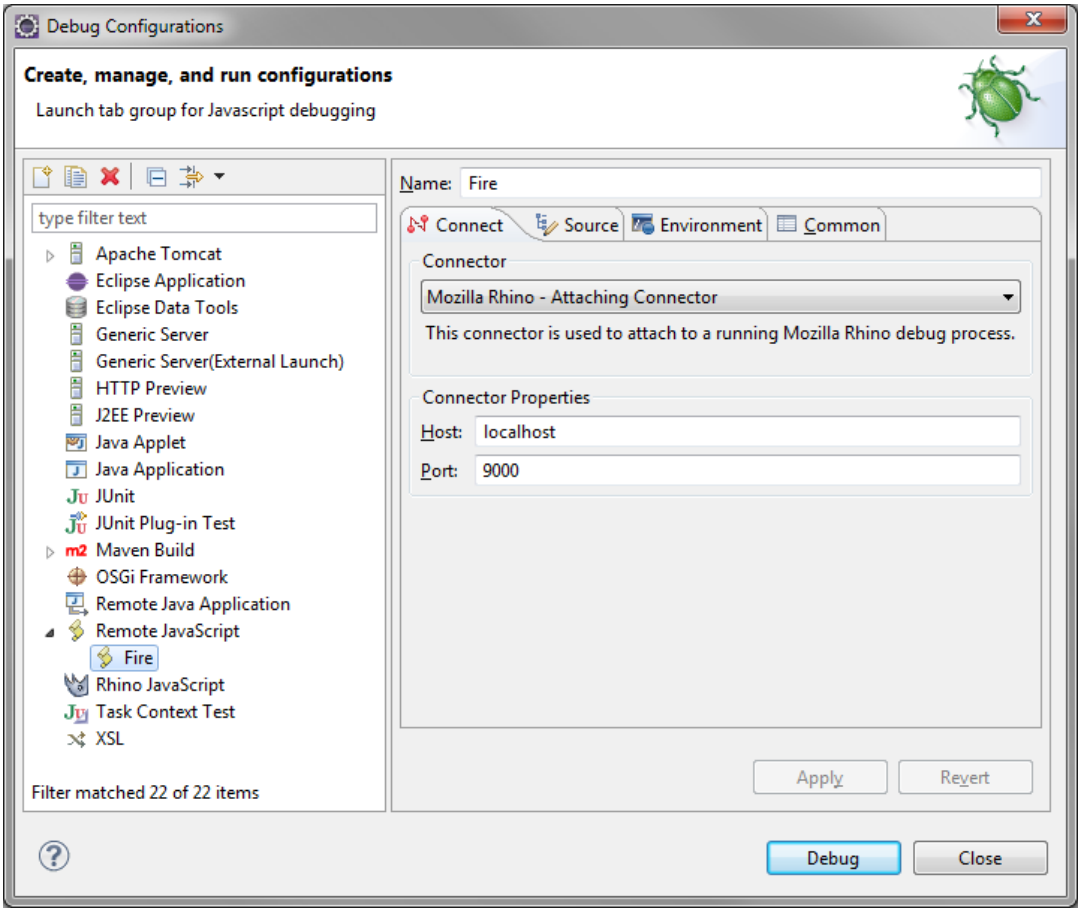
1.8.4.2 Remote Debugger

Restlet-Skripte können auch mit einem Remote Debugger analysiert werden.

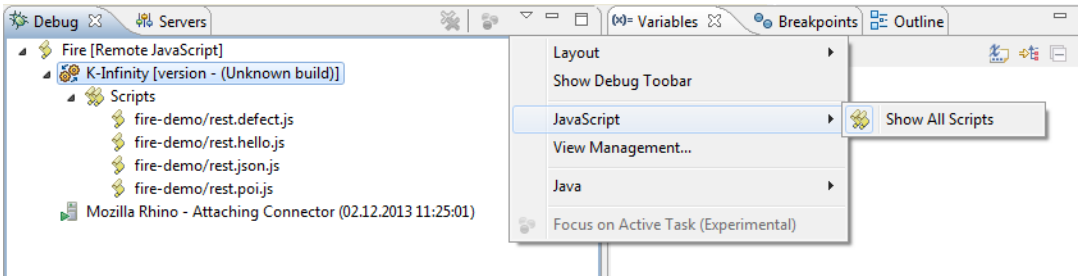
1. Debug-Port der Konfigurationsdatei der Bridge aktivieren

   ```
   [KRemoteDebuggerBridge]
   port=9000
   ```

2. Eclipse und JavaScript Development Tools installieren
3. In Eclipse eine JavaScript Debug-Konfiguration anlegen und als Connector Mozilla Rhino auswählen.



4. Debugger in Eclipse starten
5. Im Kontextmenü "Show all scripts" auswählen, damit alle Skripte der Bridge angezeigt werden.



6. "Open source" im Kontextmenü eines Skripts auswählen (Doppelklick scheint dagegen keine Funktion zu haben)
7. Im Skript können nun Breakpoints gesetzt werden

1.8.5 API-Erweiterungen

1.8.5.1 Zusätzliche Funktionen

Die API kann erweitert werden, indem eigene Funktionen bei Prototyp-Objekten hinzufügt.
Im folgenden Beispiel werden einige Prototypen erweitert, um Schemainformation mit printSchema() auszugeben.

```
// Print the schema of the instances and subtypes of a type
$k.Type.prototype.printSchema = function() {
    this.typeDomain().printSchema("Type schema of \"" + this.name() + "\"");
    this.domain().printSchema("Instance schema of \"" + this.name() + "\"");
    this.subtypes().forEach(function(subtype) {
        subtype.printSchema();
    });
}

// Print information about a property type
$k.PropertyType.prototype.logPropertySchema = function() {
    $k.out.print("\t" + this.name() + "\n");
}

// Attribute types print their type
$k.AttributeType.prototype.logPropertySchema = function() {
    $k.out.print("\t" + this.name() + " (Attribute of type " + this.valueRange().valueType() + ")\
}

// Relation types print their target domains
$k.RelationType.prototype.logPropertySchema = function() {
    $k.out.print("\t" + this.name());
    var inverse = this.inverseRelationType();
    if (inverse)
    {    var inverseDomains = inverse.domains();
        if (inverseDomains.length > 0 )
        {
            $k.out.print(" (Relation to ");
            var separate = false;
            inverseDomains.forEach(function(inverseDomain) {
                if (separate)
                    $k.out.print(", ");
                else
                    separate = true;
                $k.out.print("\"" + inverseDomain.type().name() + "\"");
            });
            $k.out.print(")");
        }
    }
    $k.out.cr();
}

// Print all properties defined for a domain
$k.Domain.prototype.printSchema = function(label) {
    var definedProperties = this.definedProperties();
    if (definedProperties.length > 0)
    {
```

```
        $k.out.print(label + "\n");
        definedProperties.sort(function(p1, p2) { return p1.name().localeCompare(p2.name()) });
        definedProperties.forEach(function(propertyType) {
            propertyType.logPropertySchema();
        });
    }
}

// Print the entire schema
$k.rootType().printSchema();
```

1.8.5.2 Eigene Prototypen definieren

Der Prototyp eines Wissensnetzelements ist normalerweise vordefiniert (Instance, Relation usw.). Es ist aber auch möglich, eigene Prototypen zu definieren und Objekten von bestimmten Typen mit der Funktion mapInstances(internalName, protoype) zuzuordnen.

Beispiel: Ein Warenkorb-Prototyp

```
// Definiere einen Prototyp mit der Funktion totalPrice()
function Basket() { }

Basket.prototype.totalPrice = function() {
    return this.relationTargets("contains").reduce(
        function(sum, item) {
            return sum + item.attributeValue("price");
        },
        0);
}

// Den Prototyp allen Objekten vom Typ "Basket" zuordnen
$k.mapInstances("Basket", Basket);

// Die neue Funktion verwenden
var baskets =  $k.Registry.type("Basket").instances();
for (var b in baskets)
    $k.out.print(baskets[b].totalPrice() + "\n");
```

1.9 REST-Services

Über die REST-Schnittstelle kann lesend und schreibend auf das Wissensnetz zugegriffen werden. Dazu definiert man im Wissensnetz **Ressourcen** (beschreiben das Verhalten der Schnittstelle beim Zugriff auf eine Ressource) und **Services** (fassen mehrere Ressourcen zusammen).

Das Verhalten einer Ressource wird über Skripte gesteuert. Zusätzlich können auch vordefinierte Ressourcen verwendet werden.

Der Zugriff erfolgt über HTTP-Requests, die nach dem Muster

```
https://<hostname>:<port>/<service-id>/<resource-pfad-und-parameter>
```

aufgebaut sind.

1.9.1 Konfiguration

Im Wissensnetz muss die REST-Komponente hinzugefügt werden. Diese definiert das notwendige Schema, das man im Knowledge-Builder im Bereich "Technik "-> "REST" findet.

Die REST-Schnittstelle wird normalerweise vom Bridge-Dienst bereitgestellt. Diese beantwortet HTTP-Anfragen anhand der REST-Konfiguration im Netz. In der Tryout-Variante des Knowledge-Builders ist die Schnittstelle bereits enthalten, man benötigt keinen Bridge-Dienst.

Konfigurationsänderungen im Wissensnetz wirken sich nicht automatisch auf bereits laufende Schnittstellen aus. Dies passiert, wenn man im Hauptmenü des Knowledge-Builders den Menüpunkt "Administrator -> REST-Schnittstelle aktualisieren" ausführt.

Der Bridge-Dienst benötigt eine passende Konfigurationsdatei (bridge.ini). In diese trägt man die Namen des Servers (host), des Wissensnetzes (volume) und der REST-Service-IDs ein. Die Zeile mit "services" kann auch komplett weggelassen werden, dann werden die Ressourcen aller vorhandenen Service-Objekte automatisch aktiviert.

```
[Default]
host=localhost
loglevel=10

[KHTTPRestBridge]
volume=demo
port=8086
services=core,extra
```

1.9.2 Services

Services fassen mehrere Ressourcen zusammen. Ressourcen können in mehreren Services enthalten sein.

Der Services-Editor im Knowledge-Builder zeigt in seiner Strukturansicht die Ressourcen an. Mit "Neues verknüpfen" wird eine neue Ressource angelegt und zum Service hinzugefügt. Mit "Bestehendes verknüpfen" wird eine bereits definierte Ressource zum Service hinzugefügt.

1.9.3 Ressourcen

Ressourcen beschreiben das Verhalten bei einer HTTP-Anfrage an die Schnittstelle. Es gibt folgende Arten von Ressourcen:

Ressource	Beschreibung
Script Resource	Durch Skripte definierbare Ressource.
Built-In Resource	Vordefinierte Ressource, deren Verhalten vom System definiert ist. Diese Ressourcen werden von der Komponente angelegt.

Static File Resource	Liefert Dateien aus dem Dateisystem aus.

Eine Ressource hat folgende konfigurierbare Eigenschaften:

Eigenschaft	**Beschreibung**

Path pattern	Definiert die URL der Ressource relativ zur Adresse des Services. Der Pfad kann parametrisiert werden, indem man Parameter in geschweiften Klammern hinzufügt:
	
	`albums/{genre}`
	
	Es können mehrere Parameter angegeben werden. Jeder Parameter muss aber ein kompletter durch "/" getrennter Teil sein:
	
	`albums/{genre}-{year}`
	
	ist nicht gültig,
	
	`albums/{genre}/{year}`
	
	schon

Part of service	Services, die diese Ressource verwenden
Description	Beschreibung zu Dokumentationszwecken
Requires authentication	Für den Zugriff auf die Ressource ist eine Authentifizierung notwendig

1.9.3.1 Methoden

Eine Ressource wird mit einer oder mehreren **Methoden** verknüpft. Diese definiert das Verhalten sowie die unterstützten Ein- und Ausgabetypen (Content-Type). Anhand der Methoden und Typen der HTTP-Anfrage wird eine passende konfigurierte Methode ausgewählt.

In der Strukturansicht werden Methoden als Unterelemente von Ressourcen angezeigt und können dort angelegt/gelöscht werden.

Methode	**Beschreibung**
HTTP Method	Unterstützte HTTP-Methoden (GET, POST, PUT, DELETE). Mehrfachangaben sind möglich.

Input media type	Nur POST/PUT: Erwarteter Content-Type des Inhalts der Anfrage.
Output media type	Content-Type der Antwort. Falls die Anfrage per "Accept" erwartete Content-Typen vorgibt, muss der Output media type dazu passen.
Script	Registriertes Skript zur Definition des Verhaltens (nur bei Script-Ressourcen relevant)
Transaction	Transaktionssteuerung (nur bei Script-Ressourcen relevant)

Die Transaktionssteuerung ist für schreibenden Zugriffe auf das Wissensnetz relevant, da diese nur innerhalb einer Transaktion möglich sind.

Transaktionssteuerung	Beschreibung
Automatic	Bei GET nur lesender Zugriff, bei POST/PUT/DELETE wird das Skript in einer Transaktion ausgeführt. Dies ist die Standardeinstellung.
Controlled by script	Keine Transaktion, das Skript muss diese selbst steuern.
Read	Nur lesender Zugriff, das Skript kann keine Transaktion starten.
Write	Das Skript wird in einer Transaktion ausgeführt.

1.9.3.2 Script-Ressource

Durch ein Skript wird bei einer Methode einer Script-Ressource die Antwort auf eine HTTP-Anfrage definiert. Von der Schnittstelle wird dazu die Funktion respond(request, parameters, response) aufgerufen, die im Skript definiert werden muss.

Argument	Typ	Beschreibung
request	$k.HttpRequest	Anfrage (URL, Header, usw.)
parameters	Object	Aus dem Request extrahierte Parameter
response	$k.HttpResponse	Antwort

Die Funktion füllt dann Header und Inhalt der Antwort. Einen Rückgabewert gibt es nicht.

Wenn für einen Parameter ein Typ definiert wurde (z.B. xsd:integer), wird der konvertierte Wert übergeben, ansonsten eine Zeichenkette. Bei Parametern, die laut Definition mehrfach vorkommen können, werden diese immer als Array übergeben.

Wenn in der Methode ein Output Content-Type für die Antwort definiert wurde, wird dieser automatisch gesetzt. Alternativ kann der Content-Type auch im Skript definiert werden.

Das folgende Skript sucht Alben und wandelt diese in JSON-Objekte um. Die Parameter der Ressource werden an als Suchparameter an die Abfrage weitergereicht.

```
function respond(request, parameters, response)
{
    var albums = $k.Registry.query("albums").findElements(parameters);
    var albumData = albums.map(function(album) {
        return {
            name: album.name(),
            id: album.idString(),
        }});
    response.setText(JSON.stringify(albumData, undefined, "\t"));
    response.setContentType("application/json");
}
```

Dieses Skript könnte man z.B. mit bei einer Ressource

```
albums/{genre}/{year}
```

verwenden und in der Abfrage "albums" die Suchparameter "genre" und "year" als Suchbedingungen verwenden.

1.9.3.3 Built-In Resourcen

Built-In Resourcen sind vordefinierte Ressourcen, deren Verhalten vom System vorgegeben ist. Jedes vordefinierte Verhalten kann durch einen zugeordneten Wert des Zeichenketten-Attributes *Rest resource ID* zugeordnet werden.

Rest resource ID	**Methode**	**Beschreibung**
BlobResource	GET	Gibt den binären Inhalt eines bestehenden Blob-Attributes zurück. Das Blob-Attribut wird über den Query-Parameter "blobLocator" identifiziert.
BlobResource	POST, PUT	Ändert den binären Inhalt eines Blob-Attributes. Das Blob-Attribut wird über den Query-Parameter "blobLocator" identifiziert. Je nach Typ des blobLocators wird ein neues Attribut angelegt oder ein bestehendes verändert.

EditorConfigResource	GET, POST, PUT	Ausgeben und Einlesen einer XML-Repräsentation eines semantischen Elementes.
ObjectListResource	GET	Gibt eine Tabelle von Instanzen oder Untertypen von dem angegebenen Typ zurück. Optional kann gefiltert, sortiert oder direkt die Menge der Objekte definiert werden.
ObjectListPrintTemplateResource	GET	Gibt eine Tabelle von Instanzen oder Untertypen in gedruckter Form zurück. Das Drucktemplate muss angegeben sein.
ObjectListPrintTemplateResource-WithFilename	GET	Gibt eine Tabelle von Instanzen oder Untertypen in gedruckter Form zurück. Das Drucktemplate muss angegeben sein. Der Parameter {filename} wird nicht ausgewertet und dient allein der besseren Handhabung im Browser.
TopicIconResource	GET	Gibt das Icon oder Bild des angegebenen semantischen Elementes zurück.

Ab i-views 4.1 kann noch ein Java-Script (*rest.preprocessScript*) an die Ressource angebracht werden. Die darin enthaltene Funktion **preprocessParameters (parameters, request)** kann die Parameter ergänzen. Aus den übergebenen Parametern kann so etwa der noch fehlende blobLocator (bzw. das zugehörige Blobattribut) ermittelt werden, was sonst einen zusätzlichen Script-Ressource-Aufruf benötigen würde.

BlobResource

Diese eingebaute Resource ermöglicht das Laden und Speichern der Inhalte von Datei-Attributen.

Download

Über die Methode "GET" kann man den binären Inhalt eines bestehenden Datei-Attributes herunterladen. Das Datei-Attribut wird über den Query-Parameter "blobLocator" identifiziert.

Upload

Beim Upload identifiziert der Parameter "blobLocator" entweder ein existierendes Datei-Attribut oder ein potentielles (d.h. neu anzulegendes) Datei-Attribut. Die Syntax für ein potentielles Attribut hat die Form: "PP∼ID1_115537458∼ID36518_344319903", wobei die erste ID das Wissensnetzelement und die zweite ID den Attribut-Prototyp repräsentiert.

Die Binärdaten können wahlweise als Multi- oder Singlepart übertragen werden. Bei Multipart können potentiell mehrere Dateien gleichzeitig hochgeladen werden, was natürlich nur Sinn macht, wenn jede Datei in ein neu anzulegendes Datei-Attribut geschrieben wird. In jedem Fall ist zu jeder übertragenden Datei der Dateiname zu setzen.

Der optionale Parameter "binaryKey" definiert den form-key, unter dem die Binärdaten im MultiPart übertragen werden.

Setzt man den optionalen booleschen Parameter "uploadOnly" auf "true", dann werden nur die Binärdaten hochgeladen jedoch nicht ins Datei-Attribut geschrieben. Dieser Modus wird

im Zusammenspiel mit dem ViewConfig-Mapper verwendet. Rückgabe ist in diesem Fall der JSON-Wert (fileName, fileSize, binaryContainerId), der dann in einem zweiten Schritt über den Mapper in das Attribut geschrieben werden kann. Der Content-Type der Rückgabe des JSON-Werts ist normalerweise "application/json", kann aber über den Parameter "overrideContentType" auf einen anderen Wert gesetzt werden, falls der Browser (z.B. IE) Probleme damit hat.

Topic Icon

Über den folgenden Pfad kann man die Bilddatei zu einem gegebenen Topic laden. Wenn ein Individuum keine eigene Bilddatei hat, wird auf die Bilddatei des Typs zurückgegriffen, welche wiederum vererbbar ist. Über den optionalen Parameter "size" kann man die Bilddatei mit der am ehesten passenden Größe selektieren, vorausgesetzt im Wissensnetz sind mehrere Bildgrößen hinterlegt.

http://{server:port}/baseService/topicIcon/{topicID}?size=10

Objektliste

Über den folgenden Pfad kann eine Objektliste im JSON-Format angefordert werden:

http://{server:port}/baseService/{conceptLocator}/objectList

Der Typ der Objektliste wird über den Parameter "**conceptLocator**" referenziert, dem Format für Topic-Referenzen in der Rest-URL folgt. (siehe Verknüpfung)

Alternativ kann der "conceptLocator" auch den einen Prototyp (Individuum oder Typ) des zu verwendenden Typs referenzieren.

Der optionale Parameter "**name**" bestimmt die Objektliste, die für die Ausgabe verwendet werden soll.

Filter

Über den optionalen und mehrwertigen Query-Parameter "**filter**" kann die Objektliste gefiltert werden. Ein Filter hat zwei mögliche Formen:

1. <Spalten-Name/Spalten-Nr.> ~ <Operator> ~ <Wert>
2. <Spalten-Name/Spalten-Nr.> ~ <Wert>

Mögliche Operatoren sind: equal, notEqual, greater, less, greaterOrEqual, lessOrEqual, equalCardinality, containsPhrase, covers, isCoveredBy, distance, fulltext, equalGeo, equalPresentTime, greaterOverlaps, greaterPresentTime, lessOverlaps, lessPresentTime, equalMaxCardinality, equalMinCardinality, overlaps, unmodifiedEqual.

Sortierung

Über den optionalen und mehrwertigen Query-Parameter "**sort**" kann die Objektliste sortiert werden. Die Reihenfolge der Sortierparameter bestimmt die Sortierpriorität. Die Angabe der Sortierung hat zwei mögliche Formen:

1. <Spalten-Name>
2. {-}<Spalten-Nr.>

Stellt man in Variante 2 ein Minus vor, wird absteigend sortiert, sonst aufsteigend.

Startmenge der Liste setzen

Über den optionalen QueryParameter "**elements**" kann eine Komma-separierte Liste von Topic-Referenzen übergeben werden, die als Listenelemente verwendet werden sollen.

Da die Liste der Element ggf. sehr lang ist, kann der Request auch als POST geschickt und die Parameter als Form-Parameter übergeben werden.

Startmenge der Liste über KPath setzen

Über die optionalen Query-Parameter "**elementsPath**" und "**startTopic**" können die initialen Elemente der Liste berechnet werden. Sind diese Parameter nicht gegeben, besteht die Ausgangsmenge aus allen Individuen bzw. allen Untertypen (im Falle einer Typ-Objektliste) des per "conceptLocator" festgelegten Typs.

Dabei ist "elementsPath" ein KPath-Ausdruck und "startTopic" eine Referenz auf das Topic, mit dem die Auswertung des KPath gestartet werden soll. Die Form des Parameters "startTopic" entspricht der des "conceptLocator".

Vererbung

Über den optionalen Query-Parameter "**disableInheritance**" kann die Vererbung unterdrückt werden. Der Paramater macht nur Sinn, wenn kein "elementsPath" gesetzt ist.

JSON-Ausgabeformat (Beispiel)


```
{
 rows: [{
  topicID: "ID123_987654321",
  row: ["MM",
  "Mustermann",
  "Max",
  "111",
  "m.mustermann@email.net",
  "10",
  "6",
  "2000-01-01",
  "Projekt A, Projekt B"]
 },
 {
  topicID: "ID987_123456789",
  row: ["MF",
  "Musterfrau",
  "Maxine",
  "222",
  "m.musterfrau@email.net",
  "10",
  "8",
  "2000-01-01",
  "Projekt X, Projekt Y, Projekt Z"]
 }],
 columnDescriptions: [{
  label: "Login",
  type: "string",
  columnId: "1"
 },
 {
  label: "Nachname",
  type: "string",
  columnId: "2"
 },
```

```
 {
  label: "Vorname",
  type: "string",
  columnId: "3"
 },
 {
  label: "Telefondurchwahl",
  type: "string",
  columnId: "4"
 },
 {
  label: "Email",
  type: "string",
  columnId: "5"
 },
 {
  label: "Verfügbarkeit",
  type: "number",
  columnId: "6"
 },
 {
  label: "Aufwand",
  type: "string",
  columnId: "7"
 },
 {
  label: "erstellt am",
  type: "dateTime",
  columnId: "8"
 },
 {
  label: "Projekt",
  type: "string",
  columnId: "9"
 }]
}
```


Objektlistendruckvorlage

Über den folgenden Pfad kann eine Objektliste in eine 'Druckvorlage für Liste' gefüllt und das Resultat heruntergeladen werden:

http://{server:port}/baseService/{conceptLocator}/objectList/printTemplate/

{templateLocator}/{filename}

Der Service funktioniert genau wie der Abruf einer Objektliste, trägt aber als zuätzlichen Parameter eine Referenz auf das Individuum des Typs 'Druckvorlage für Liste' im Wissensnetz.

"**templateLocator**" muss eines der unter "Allgemeines" beschriebenen Formate haben

Der optionale Pfad-Parameter "filename" wird nicht ausgewertet und dient dem besseren Browser-Handling.

Über das Header-Field "**Accept**" wird gesteuert, in welches Ausgabeformat konvertiert werden soll. Fehlt das Header-Field oder ist der Wert "***/***", findet keine Konvertierung statt. Mehrwertige Accept werden nicht unterstützt und resultieren in einer Fehlermeldung.

Über den optionalen Query-Parameter "**targetMimeType**" kann der Wert des "Accept" Header-Fields überschrieben werden. Dies ist notwendig, wenn man den Request aus einem Browser aufrufen möchte und dort keinen Einfluss auf die Header-Fields hat.

Topic drucken

Über den folgenden Pfad kann ein Topic in ein Drucklistentemplate gefüllt und das Resultat heruntergeladen werden:

http://{server:port}/baseService/{topicLocator}/printTemplate/

{templateLocator}/{filename}

"**templateLocator**" muss eines der unter "Allgemeines" beschriebenen Formate haben

Der optionale Pfad-Parameter "filename" wird nicht ausgewertet und dient dem besseren Browser-Handling.

Über das Header-Field "**Accept**" wird gesteuert, in welches Ausgabeformat konvertiert werden soll. Fehlt das Header-Field oder ist der Wert "***/***", findet keine Konvertierung statt. Mehrwertige Accept werden nicht unterstützt und resultieren in einer Fehlermeldung.

Über den optionalen Query-Parameter "**targetMimeType**" kann der Wert des "Accept" Header-Fields überschrieben werden. Dies ist notwendig, wenn man den Request aus einem Browser aufrufen möchte und dort keinen Einfluss auf die Header-Fields hat.

Dokumentformatkonvertierung

Über den folgenden Pfad kann ein Dokument in ein anderes Format umgewandelt werden (z.B. odt in pdf):

http://{server:port}/baseService/jodconverter/service

Der Service bildet den JOD-Konverter (siehe http://sourceforge.net/projects/jodconverter/) ab und dient der Abwärtskompatibilität für Installationen, die bisher mit dem JOD-Konverter betrieben wurden.

Damit der Service funktioniert muss Open/LibreOffice (ab Version 4.0) installiert sein und die Konfigurationsdatei "bridge.ini" muss einen Eintrag enthalten, der auf die Datei "soffice" verweist:

```
[file-format-conversion] sofficePath="C:\Program Files (x86)\LibreOffice 4.0\program\soffice.exe"
```

1.9.3.4 Static File Resource

Liefert Dateien aus dem Dateisystem aus.

Bei dieser Art von Ressource legt man lediglich per *Path pattern* das Verzeichnis fest, unterhalb dessen Dateien ausgeliefert werden. Die Adressierung des Verzeichnisses erfolgt relativ zum Installationsverzeichnis der REST-Bridge.

Beispiel:

Gegeben sei ein Verzeichnis *icons* mit der Datei *bullet.png*. Das Path-Pattern der Ressource lautet *icons*, der dazugehörige Service hat die *Service ID test*. Der Zugriff auf die Datei *bullet.png* lautet dann:

http://localhost:8815/test/icons/bullet.png

1.9.3.5 Ressourcen Parameter

Unterhalb von Methoden kann man die **Parameter** der Ressource definieren. Dies ist nicht zwingend erforderlich, hat aber einige Vorteile:

- Durch Typangaben kann man Parameter prüfen und konvertieren (z.B. in Zahlen oder Objekte)
- Dokumentation für Kunden

Folgende Parameter-Eigenschaften können konfiguriert werden:

Parameter name	Name des Parameters
Style	Art des Parameters • path (Teil des Pfads der URL) • query (Query-Parameter der URL) • header (HTTP-Header)
Type	Datentyp des Parameters. Parameter werden validiert und umgewandelt an das Skript übergeben.
Repeating	Parameter darf mehrfach vorkommen. Wenn aktiviert wird immer eine Array von Werten an das Skript übergeben, selbst wenn nur ein Parameterwert in der Anfrage vorhanden ist.
Required	Parameter muss angegeben werden
Fixed value	Standardwert, falls kein Parameter angegeben wurde.

1.9.4 CORS

Bei OPTIONS-Requests antwortet die REST-Schnittstelle standardmäßig mit

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Headers: Origin, X-Requested-With,Content-Type, Accept
```

In der Konfigurationsdatei (bridge.ini) können diese Header konfiguriert werden:

```
[KHTTPRestBridge]
accessControlAllowOrigin=http://*.i-views.de
accessControlAllowHeaders=Origin, X-Requested-With,Content-Type, Accept
```

1.10 Berichte und Drucken

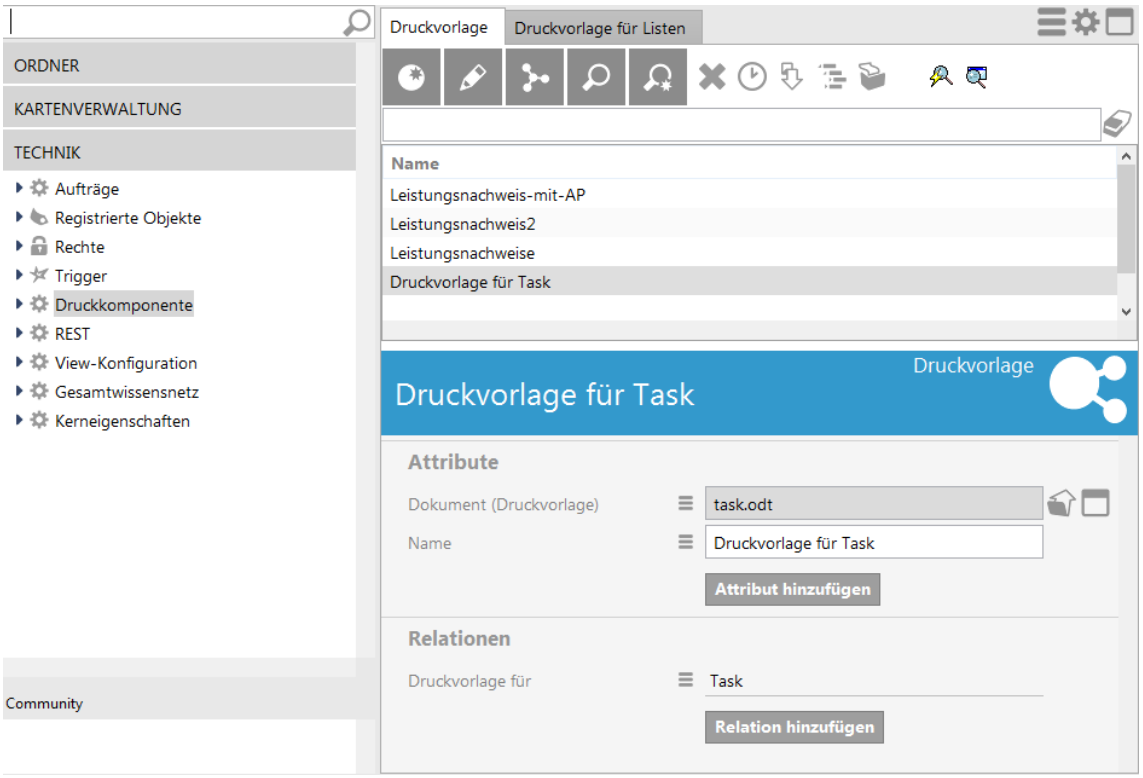
Mit Hilfe der Druckkomponente kann man Dokumentvorlagen (ODT/DOCX/XLSX/RTF-Dateien) mit KPath-Ausdrücken auf Objekten oder Objektlisten anwenden und daraus eine angepasste Ausgabe-Datei generieren, die entweder gedruckt oder gespeichert werden kann.

Das Hinzufügen der Druckkomponente über das Admin-Tool legt im Wissensnetz Konfigurationsschema für Objekte ("Druckvorlage") und Listen ("Druckvorlage für Listen") an. Die Existenz dieser Komponente ist Voraussetzung dafür, dass die Druckfunktionalität im Knowledge-Builder bzw. über die REST-Schnittstelle zur Verfügung steht.

1.10.1 Druckvorlagen erstellen

Druckvorlagen werden im Knowledge-Builder im Bereich "TECHNIK/Druckkomponente" angelegt. Jedes Druckvorlagen-Objekt enthält ein Druckvorlagen-Dokument (ODT,DOCX,RTF) und eine Relation, die angibt auf welche Objekte die Druckvorlage angewendet werden soll.

Das folgende Beispiel zeigt eine ODT-Druckvorlage für Objekte des Typs "Task".



Die folgenden Kapitel erläutern das Erstellen der Druckvorlagen-Dokumente.

1.10.1.1 RTF-Vorlagen erstellen

Die RTF-Vorlagedateien können auswertbare KPath-Ausdrücke mit den Schlüsselworten **KPATH_EXPAND** und **KPATH_ROWS** sowie Aufrufe registrierter KSkripte mit den Schlüsselworten **KSCRIPT_EXPAND** und **KSCRIPT_ROWS** enthalten. Die Pfadausdrücke bzw. der Name des aufzurufenden Skriptes stehen immer zwischen spitzen Klammern und nach dem Schlüsselwort durch ein Leerzeichen getrennt.

KPATH_EXPAND

Der KPath-Ausdruck nach diesem Schlüsselwort sollte ein einzelnes semantisches Objekt oder einen einfachen Wert (Datum, Zeichenkette etc.) zurückliefern. Bei der Auswertung wird der ursprüngliche Ausdruck durch das Ergebnis ersetzt. Die Formatierung des Ausdrucks bleibt erhalten, Umbrüche des Wertes werden in Zeilenumbrüche umgesetzt.

Beispiel:
Die Vorlage sei:

```
Absender:
<KPATH_EXPAND @$adresse$/rawValue()>
```

Nach Auswertung steht in der Ausgabedatei:

```
Absender:
intelligent views gmbh
Julius-Reiber-Str. 17
64293 Darmstadt
```

KSCRIPT_EXPAND

Alternativ zum Pfadausdruck kann mit KSCRIPT_EXPAND ein registriertes KSkript aufgerufen werden. Die Ausgabe dieses Skriptes (Skriptelemente mit <Output>) wird in das Dokument übernommen. Die Registrierung von Skripten erfolgt im Knowledge-Builder im Ordner TECHNIK/Registrierte Objekte/Skri.

Beispiel:
Die Vorlage sei:

```
<KSCRIPT_EXPAND einSkriptDas1bis9Ausgibt>
```

Nach Auswertung steht in der Ausgabedatei:

```
123.456.789
```

KPATH_ROWS

Dieser Ausdruck muss in einer Tabelle stehen. Der KPath-Ausdruck nach diesem Schlüsselwort muss eine Liste semantischer Objekte liefern. Bei der Auswertung wird die Tabellenzeile des KPATH_ROWS Ausdrucks für jedes Ergebnis des KPath-Ausdrucks einmal ausgewertet. Somit können Tabellen dynamisch ergänzt werden. Es spielt übrigens keine Rolle, in welcher Spalte der KPATH_ROWS Ausdruck steht.

Beispiel:
Die Vorlage sei:

Teile (<KPATH_EXPAND topic()/~$hatTeile$/size()> Stück)	**Bemerkung**
<KPATH_EXPAND topic()><KPATH_ROWS topic()/~$hatTeil$/target()/sort(@$name$, true)>	<KPATH_EXPAND topic()/@$bemerkung$>

Nach Auswertung in der Ausgabedatei:

Teile (3 Stück)	Bemerkung
RTF-Druck	
ODT-Druck	Ersetzt den RTF-Druck
Konvertierungsservice	Optionaler Dienst

KSCRIPT_ROWS

Bei KSCRIPT_ROWS werden die Objekte für die Tabellenzeilen durch ein registriertes KSkript ermittelt. Der Name des registrierten Skriptes wird direkt hinter KSCRIPT_ROWS angegeben. Das Skript muss vom Typ KSkript sein und die auszugebenden Objekte zurückgeben.

Beispiel:
Die Vorlage sei:

Spalte1	Spalte2
<KSCRIPT_ROWS allePersonen><KPATH_EXPAND @$nachname$>	<KPATH_EXPAND @$Vorname$>

Nach Auswertung in der Ausgabedatei:

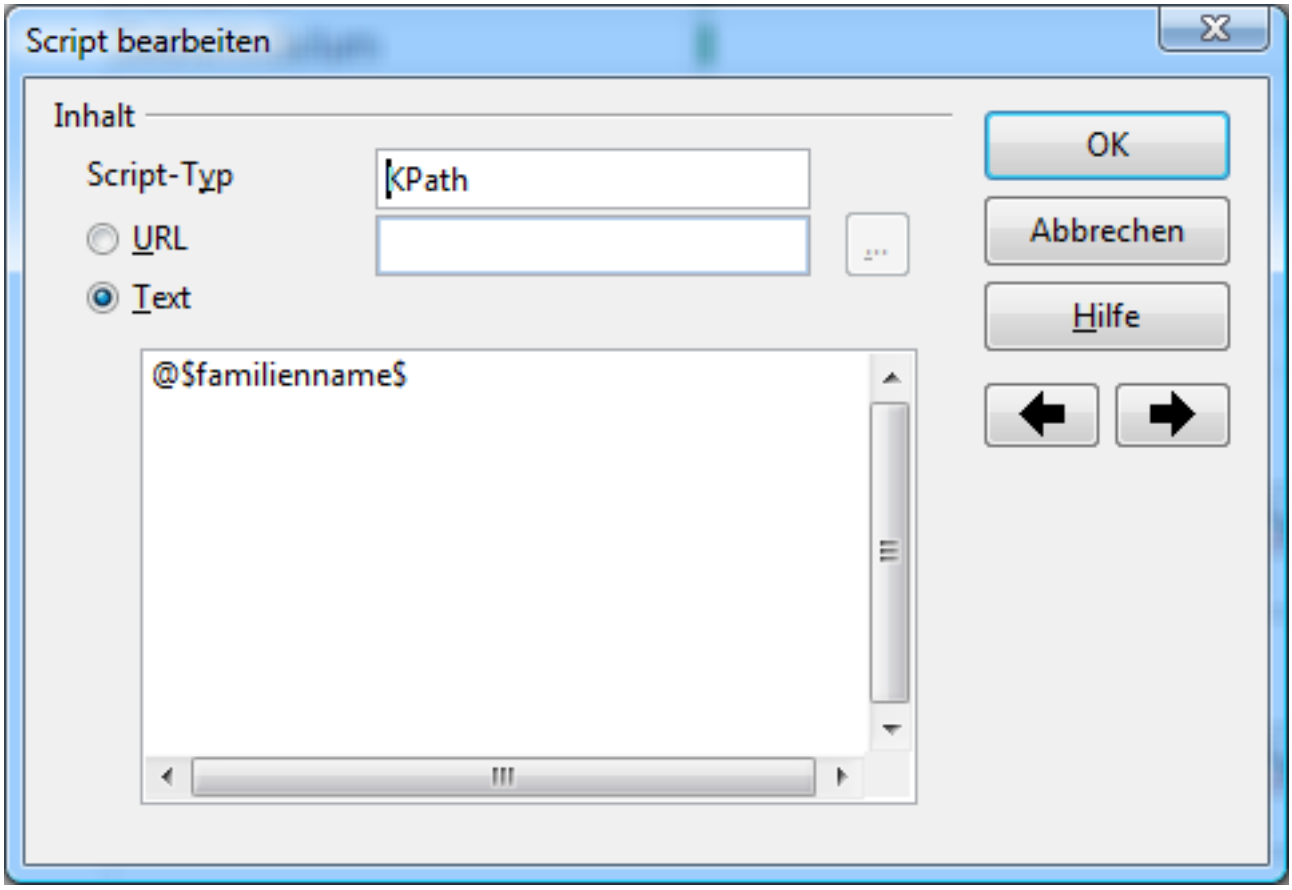
Spalte1	Spalte2
Meier	Peter
Schulze	Helmut

1.10.1.2 ODT-Dokumente (OpenOffice) erstellen

Der Druck über das ODT-Format (Open Document Text, offener Standard) hat viele Vorteile gegenüber dem RTF-Format:

- Die eingebetteten Skript-Anweisungen sind nicht Teil des Textes sondern werden in speziellen Script-Elementen abgelegt. Somit macht man sich seine Formatierung nicht durch längliche Skripte kaputt.
- Das ODT-Format unterstützt eine große Menge an Formatanweisungen (vergleichbar mit MS-Word), die RTF nicht kennt.
- RTF hat als Format keine einheitliche Normierung (MS-Word kann z.B. "mehr" RTF als der Standard).
- Die Bearbeitung der RTF-Vorlagen ist sehr fragil. Vor allem MS-Word neigt dazu, die Vorlagen mit Steuerelementen (wie z.B. die aktuelle Cursorposition bei der letzen Bearbeitung) zu ergänzen , sodass die Skripte nicht mehr verlässlich erkannt werden können.

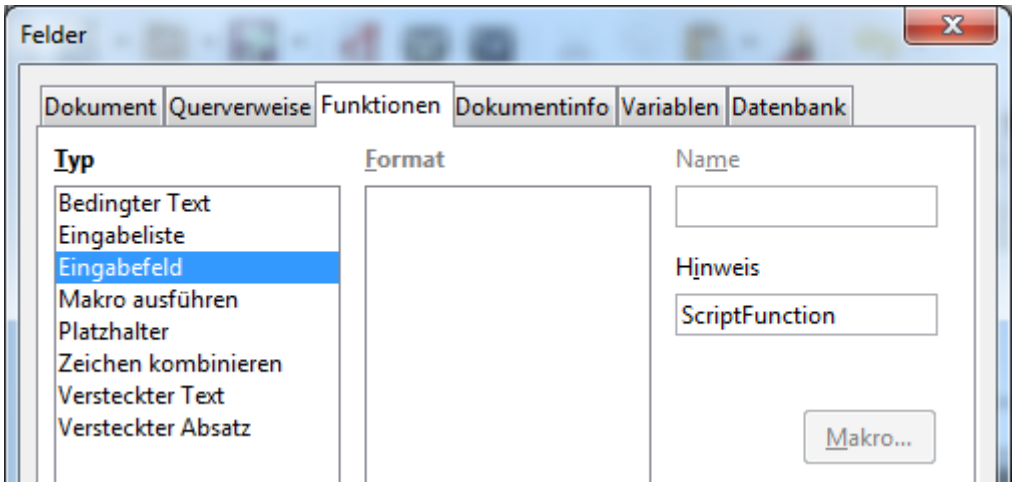
ODT-Vorlagen können mit OpenOffice oder LibreOffice erstellt werden. Die Erstellung erfolgt analog zu der Erstellung von RTF-Vorlagen mit dem Unterschied, dass die Path-/Script-Anweisungen in Script-Elementen abgelegt werden, wie in der folgenden Abbildung gezeigt.



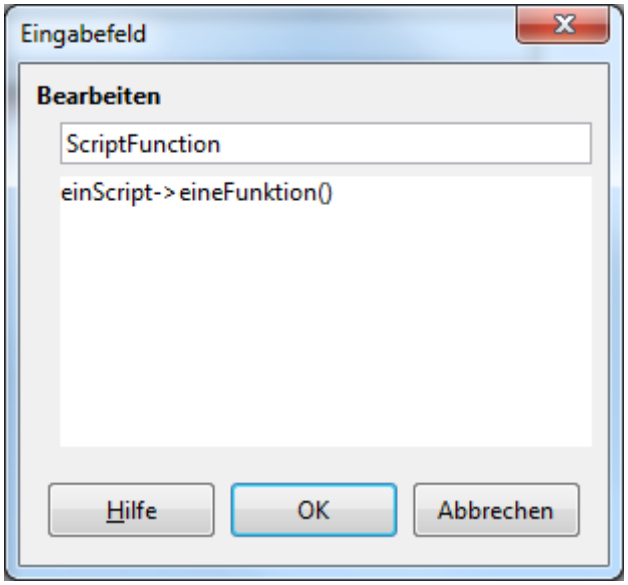
In LibreOffice 5 lässt sich das Skriptfeld nicht länger einbinden. Als Alternative hierzu kann das Feld "Eingabefeld" benutzt werden:

Einfügen > Feldbefehl > Weitere Feldbefehle (alternativ Tastenkombination Strg+F2)

Dort findet sich im Reiter "Funktionen" das Eingabefeld.



"Hinweis" entspricht dem vorherigen "Script-Typ"; nach einem Klick auf Einfügen öffnet sich ein weiteres Fenster, in das das Skript eingetragen werden kann.



Verfügbare Skript-Typen

Als Skript-Typen gibt es:

- **KPath** : analog zu **KPATH_EXPAND**
- **KScript** : analog zu **KSCRIPT_EXPAND**
- **KPathRows** : analog zu **KPATH_ROWS**
- **KPathImage** : zur Einbettung von Bildern
- **ScriptFunction**: Aufruf einer Funktion eines registrierten Scripts. Als Text wird eine Zeichenkette mit folgendem Format erwartet:

  `ScriptID->Funktionsname()`
  Der Funktionsaufruf wird automatisch um zwei Argumente erweitert: das semantisch Elemente und die durch die Umgebung vorgegebenen Variablen

  Ein Beispiel für ein aufgerufenes Skript:

  ```
  function headerLabel(element, variables)
  {
     return element.name().toLocaleUpperCase();
  }
  ```

- **ScriptRowsFunction**: Analog zu ScriptFunction. Für die zurückgegebenen Objekte werden analog zu KPathRows Tabellenzeilen erzeugt.
- **ScriptImageFunction:** zum Einfügen von Bitmap-Images
- **ScriptSVGImageFunction:** zum Einfügen von SVG-Zeichnungen
- **DataPath:** An der Druckvorlage muss das "Skript zur Erzeugung von JSON-Inhalten" gesetzt sein. Auf die Werte des JSON-Objekts kann nun über den entsprechenden

Schlüssel zugegriffen werden.

Beispiel zum Erzeugen des JSON-Objekts:

```
function templateData(element)
{
    return {
            name: element.name(),
            idNumber: element.idNumber(),
            someData: { idString: element.idString() }
    }
}
```

Um zum Beispiel auf den Wert idString zuzugreifen, muss als Text

```
someData.idString
```

gesetzt sein.

- **DataRowsPath:** Analog zu DataPath. Als Text wird ein Schlüssel erwartet, dessen Wert im JSON ein Array von Objekten ist. Nachfolgende DataPath-Felder beziehen sich auf die Objekte in diesem Array.

Zur Einbettung von Bildern können Datei-Attribute oder URLs verwendet werden. Bei der Verwendung von URLs wird versucht, ein Bild von der angegebenen Adresse zu laden.

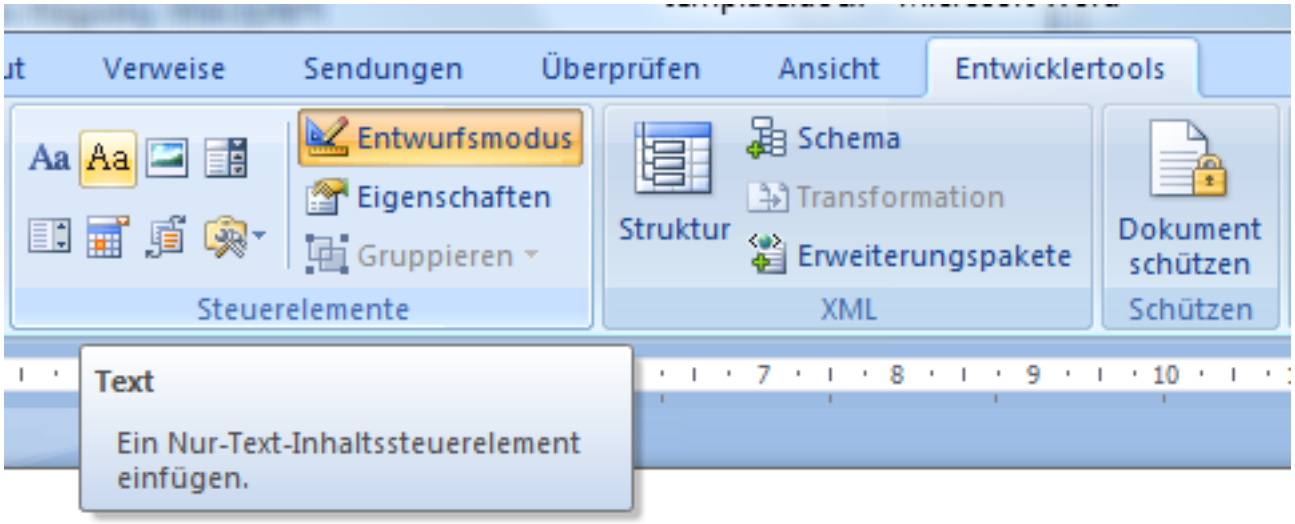
Eingebettete Bilder werden immer auf ihre Originalgröße (bei 96d dpi) gezogen. Möchte man im Ausdruck eine andere Größe erhalten, muss man einen Rahmen mit den gewünschten Ausmaßen (unbedingt absolute Maße in cm verwenden!) um das Skript-Element bauen. Das resultierende eingebettete Bild wird dann so in den Rahmen eingepasst, dass das Rahmenmaß unter Beibehaltung der Bild-Seitenverhältnisse nicht überschritten wird.

1.10.1.3 DOCX-Dokumente (Microsoft Word) erstellen

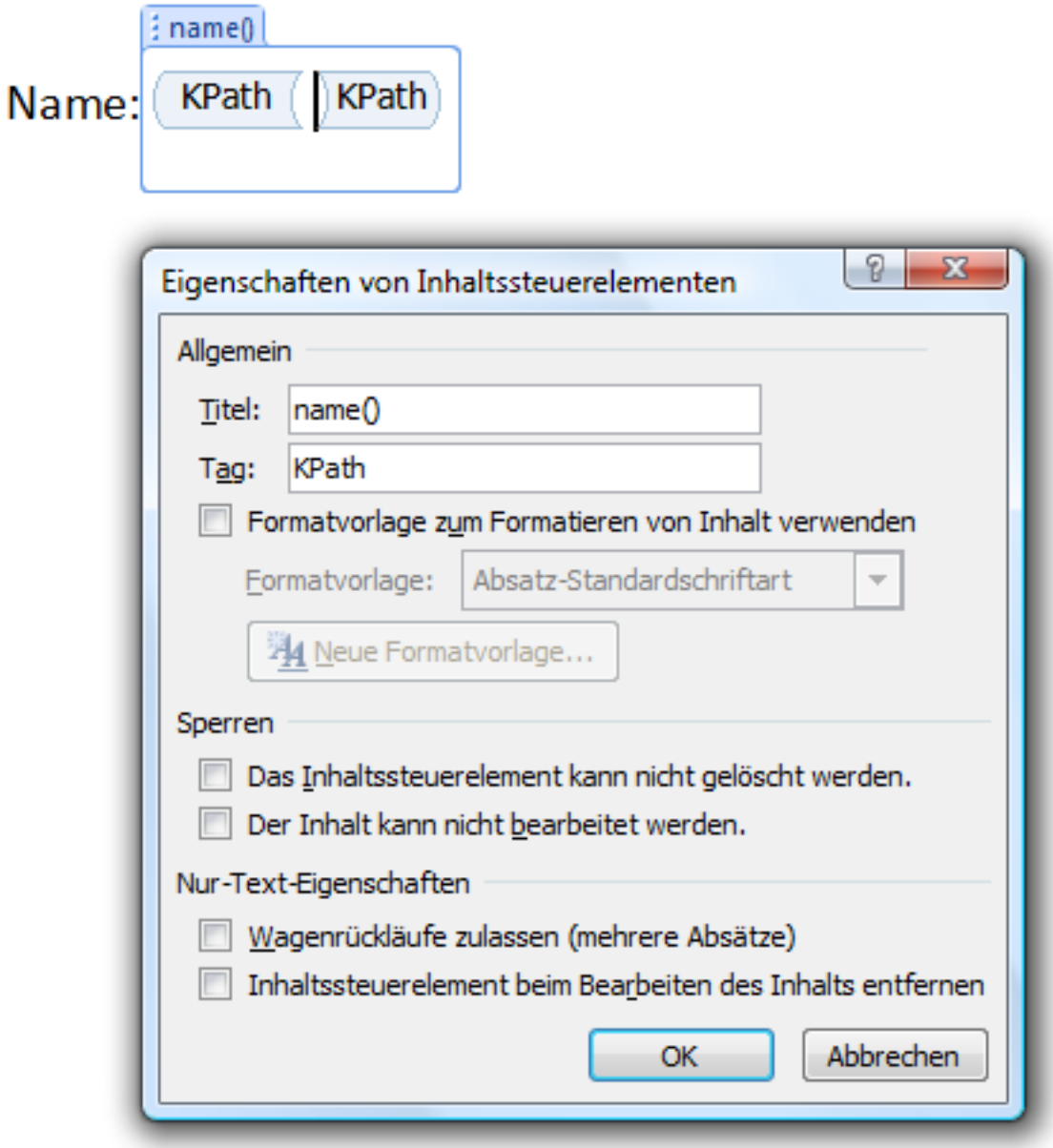
DOCX-Vorlagen können mit Microsoft Word 2007 oder neuer erstellt werden.

Die Erstellung erfolgt analog zu der Erstellung von RTF-Vorlagen mit dem Unterschied, dass die Path-/Skript-Anweisungen in Text-Inhaltssteuerelementen abgelegt werden.

Zum Einfügen der Steuerelement müssen in Word zuerst die Entwicklertools aktiviert werden. Dazu im Office-Menü die **Word-Optionen** öffnen und in der Kategorie **Häufig verwendet** die Option **Entwicklerregisterkarte in der Multifunktionsleiste anzeigen** aktivieren. Nun aktiviert man auf der Registerkarte **Entwicklertools** den **Entwurfsmodus**.



Um KSkript/KPath-Ausdrücke einzufügen, fügt man ein **Nur-Text-Inhaltssteuerelement** ein. Der Text des Steuerelements wird durch den berechneten Text ersetzt. Bei den Eigenschaften des Steuerelements (erreichbar über das Kontextmenü auf der schließenden Klammer) gibt man bei **Titel** das KSkript bzw. den KPath an. Falls man den Titel leer lässt, wird stattdessen der Text des Steuerelements verwendet. Als **Tag** gibt man den Skript-Typ an. Als Skript-Typen gibt es alle Typen, die bei ODT zur Verfügung stehen, mit Ausnahme von KPathImage**.**



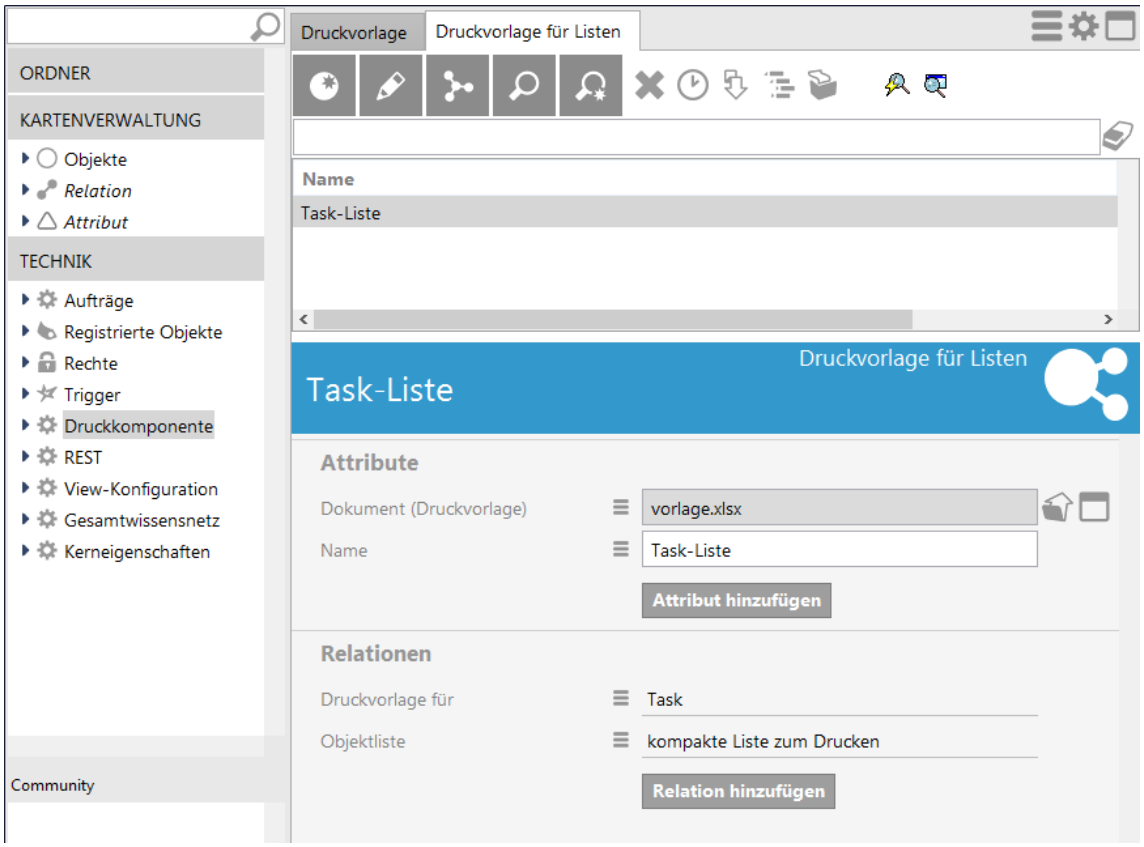
1.10.2 Druckvorlagen für Listen erstellen

Druckvorlagen für Listen werden im Knowledge-Builder im Bereich "TECHNIK/Druckkomponente" angelegt. Jedes "Druckvorlagen für Listen"-Objekt enthält ein Druckvorlagen-Dokument (XSLX) und eine Relation, die angibt auf welche Objekte die Druckvorlage angewendet werden soll. Optional kann eine Objektliste angegeben werden, die zur Generierung der Ausgabe verwendet werden soll. Auf diese Weise kann man bewirken, dass sich das Format der Liste, die der Anwender am Bildschirm sieht und das der ausgegebenen Liste unterscheiden.

Wenn man das Attribut "Dokument (Druckvorlage)" nicht angelegt hat, so wird bei der Dokumentgenerierung eine Excel-Datei erzeugt, die ein Arbeitsblatt mit den Daten der Objektliste und den Spaltenüberschriften aus der Objektlistenkonfiguration enthält, d.h. man muss nicht

zwangsläufig eine Excel-Datei als Druckvorlage angeben.

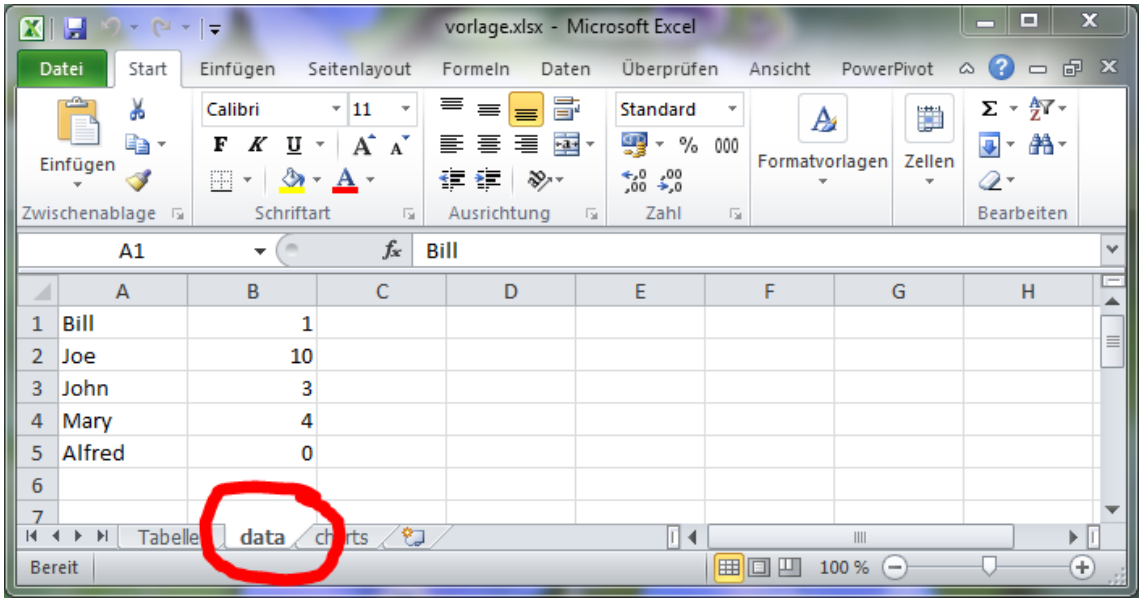
Das folgende Beispiel zeigt eine Druckvorlage für Listen mit Objekten des Typs "Task".



XLSX-Vorlagen können mit Microsoft Excel 2007 oder neuer erstellt werden. Diese Vorlagen funktionieren nur mit Objektlisten.

Erstellen der Excel-Datei

Als Vorlage dient eine gewöhnliche Excel-Datei, die ein zusätzliches Arbeitsblatt namens "data" enthalten muss. Die Objektlistendaten werden später dann in dieses Arbeitsblatt gefüllt und zwar ohne Überschriften und beginnend mit der Zelle A1.



Die anderen Arbeitsblätter können Daten aus dem Blatt "data" in Formeln referenzieren. i-views sorgt dafür, dass alle Formeln neu berechnet werden, sobald die ausgefüllte Excel-Datei das nächste Mal mit Excel geöffnet wird.

1.10.3 Dokumentformatkonvertierung mit Open/LibreOffice

Das Ausgabeformat des Druckvorgangs entspricht dem des verwendeten Templates. Möchte man ein anderes Ausgabeformat erhalten, muss man einen Konverter einrichten.

Dazu benötigt man eine Installation von Libre- oder OpenOffice ab Version 4.0 auf dem Rechner, der die Konvertierung durchführen soll - gewöhnlich dort, wo die Bridge oder der Jobclient läuft, der auch den Druckvorgang durchführt.

Zusätzlich muss in der Konfigurationsdatei (bridge.ini, jobclient.ini, etc.) der Pfad zum "soffice"-Programm angegeben werden, welches Teil der Libre/OpenOffice-Installation ist und sich dort im Unterverzeichnis "program" befindet. Diese Angabe muss als absoluter Pfad erfolgen, relative Pfade (..\LibreOffice\etc.) sind hier nicht möglich.

```
[file-format-conversion]
sofficePath="C:\Program Files (x86)\LibreOffice 4.0\program\soffice.exe"
```

Konvertierungsservice

Möchte oder kann man nicht an allen Arbeitsplätzen bzw. Serverinstallationen, von denen aus die Formatkonvertierung durchgeführt werden soll, eine Libre/Office-Installation vorhalten, kann eine entsprechend konfigurierte REST-Bridge die Konvertierung durchführen.

Das .ini-File der REST-Bridge muss dazu folgendes Format aufweisen:

```
[Default]
host=localhost

[KHTTPRestBridge]
port=3040
volume=kartenverwaltung
```

```
services=jodService

[file-format-conversion]
sofficePath="C:\Program Files (x86)\LibreOffice 4.0\program\soffice.exe"
```

Im Admin-Tool trägt man unter Systemkonfiguration/Komponenten/Konvertierungsservice ein, über welche Adresse der Konvertierungsservice erreichbar ist.

Beispiel:

```
http://localhost:3040/jodService/jodconverter/service
```

Dokument-Formate

Damit die Ausgabeformate verfügbar sind, müssen entsprechend konfigurierte Objekte vom Typ "Konverter-Dokumentformat" im Netz vorhanden sein.

Wichtig ist, dass nicht alle Formate ineinander Konvertiert werden können. Die wichtigsten sind:

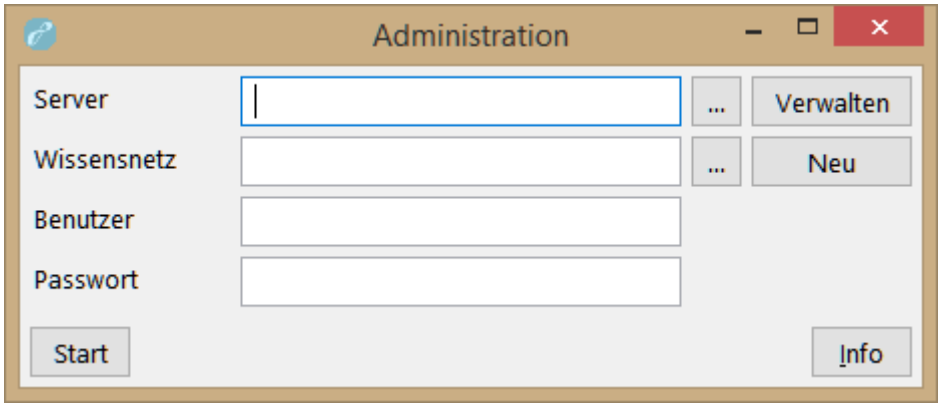
Name	Exten-sion	Mime-Type
Portable Document Format	pdf	application/pdf
OpenDocument Text	odt	application/vnd.oasis.opendocument.text
Microsoft Word	doc	application/msword

2 Admin-Tool

Mit dem Admin-Tool können neue Wissensnetze angelegt, alle Wissensnetze eines Mediators verwaltet und einzelne Wissensnetze konfiguriert werden.

2.1 Startfenster

Nach dem Start des Admin-Tools (Windows: *admin.exe*, Mac OS: *admin*, Linux: *admin-64.im*) erscheint das **Startfenster**.



2.1.1 Server

Im Freitextfeld **Server** wird die URL des Servers eingegeben. (Falls kein Protokoll angegeben wird, wird das Protokoll cnp:// verwendet). Gültige URLs verwenden eines der Protokolle *[cnp://,cnps://,http:// oder https://] gefolgt von [Rechnername oder IP-Adresse]:[Portnummer]*. Dieses Format entspricht der interface einstellung am Mediator.

Läuft der Mediator, über den Wissensnetze administriert werden sollen, auf dem gleichen Rechner wie das Admin-Tool, kann er auch über den Rechnernamen *localhost* angesprochen werden.

Bleibt das Feld leer, wird auf die Wissensnetze zugegriffen, die relativ zur Position des Admin-Tools im direkten Unterordner *volumes* liegen. Für diese Art des Zugriffes ist kein Mediator notwendig.

Einmal eingegebene Einträge im Freitextfeld werden gespeichert. Über die Schaltfläche **...** können sie in einem separaten Fenster aus einer Liste ausgewählt werden.

Mit der Schaltfläche **Verwalten** gelangt man zur **Serververwaltung**, bei der eine Authentifizierung mit dem Serverpassword benötigt wird.

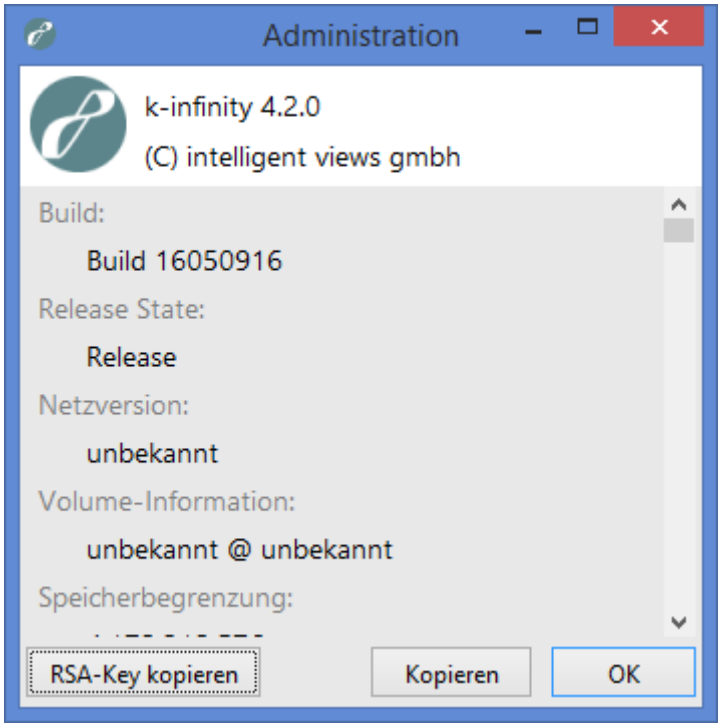
2.1.2 Wissensnetz

Im Freitextfeld **Wissensnetz** wird das Wissensnetz angegeben, das administriert werden soll.

Einmal eingegebene Einträge im Freitextfeld werden gespeichert. Über die Schaltfläche **...** können sie in einem separaten Fenster aus einer Liste ausgewählt werden. Zur Anzeige aller Wissensnetze wird man gegebenenfalls aufgefordert, das Serverpasswort einzugeben.

2.1.3 Info

Über die Schaltfläche **Info** lassen sich in einem eigenen Fenster versionsspezifische Informationen über das Admin-Tool abrufen.



Konkret handelt es sich dabei um

- die Versionsnummer des Admin-Tools (*Build*),
- den Veröffentlichungsstatus des Admin-Tools (*Release State*),
- die vom Admin-Tool maximal nutzbare Menge an Systemarbeitsspeicher in Byte (*Speicherbegrenzung*),
- die Versionsnummer und der digitale Fingerabdruck der vom Admin-Tool verwendeten Ausführungsumgebung (*VM Version*),
- die im Betriebssystem aktive Spracheinstellung (*Locale*),
- die im Admin-Tool verwendeten, mitgelieferten Schriftarten (*Fonts*),
- die mit dem Admin-Tool ausgelieferten Wissensnetzkomponenten inklusive Versionsnummer (*Softwarekomponenten*) und
- die im Admin-Tool verwendeten Smalltalk-Pakete inklusive Versionsnummer (*Pakete*).

Die Angaben zur *Netzversion* und zur *Volume-Information* sind hierbei nicht einschlägig.

Die Informationen werden in einem unsichtbaren Textfeld ausgegeben, welches über ein Kontextmenü verfügt, das per Rechtsklick aktiviert werden kann:

- **Select All** markiert den gesamten Text. Alternativ kann mit dem Mauszeiger ein beliebiger Textausschnitt markiert werden.
- **Copy** kopiert den gewählten Textbereich in die Zwischenablage des Betriebssystems.
- **Find Again** sucht nach dem gewählten Textbereich und findet sein nächstes Auftreten gemäß der Leserichtung.
- **Find** erlaubt die Eingabe einer zu suchenden Zeichenkette in einem separaten Fenster und findet ihr nächstes Auftreten gemäß der Leserichtung relativ zur Position der per Mausklick setzbaren Schreibmarke. Bei der Suche wird Groß- und Kleinschreibung un-

terschieden.

Über die Schaltfläche **Kopieren** werden alle Informationen in die Zwischenablage des Betriebssystems kopiert.

Die Schaltfläche **RSA-Key kopieren** kopiert den für jedes kompilierte Admin-Tool einzigartigen Schlüssel in die Zwischenablage des Betriebssystems. Dieser Schlüssel kann in die Initialisierungsdatei eines Mediators (Standarddateiname *mediator.ini*) eingetragen werden und beschränkt dadurch den Zugang zu diesem Mediator über ein Admin-Tool auf Admin-Tools mit genau diesem Schlüssel.

Die Schaltfläche **Ok** ermöglicht eine Rückkehr zum Startfenster.

2.1.4 Verwalten, Neu und Weiter

Neu leitet weiter zur Wissensnetzerzeugung.

Verwalten leitet weiter zur Serververwaltung.

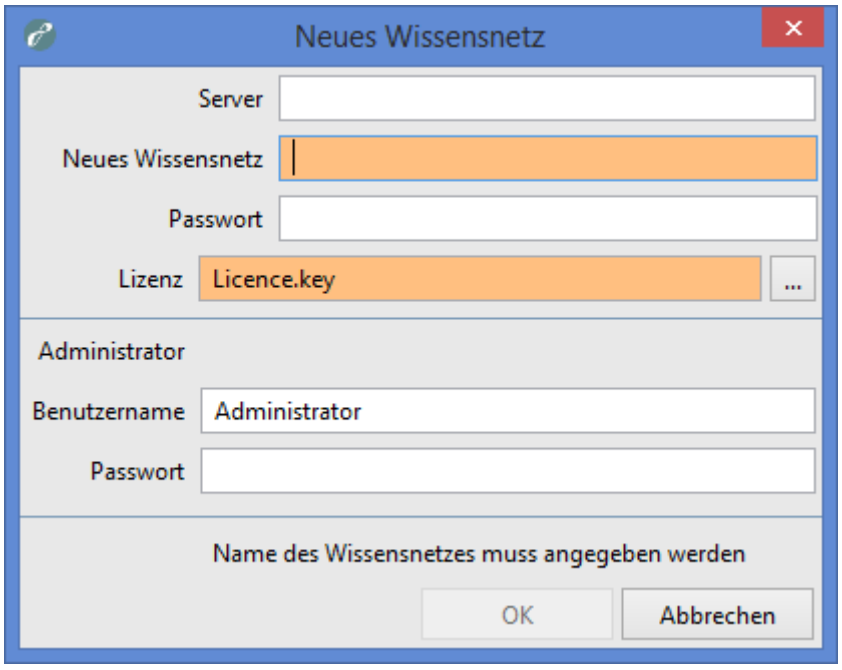
Weiter leitet weiter zur Einzelnetzverwaltung. Hierfür werden die Einträge **Benutzername** und **Passwort** zur Anmeldung mit einem Administratorkonto verwendet.

2.1.5 Ende

Die Schaltfläche **Ende** schließt das Admin-Tool.

2.2 Wissensnetzerzeugung

Das Anlegen eines neuen Wissensnetzes erfolgt über ein eigenes **Netzerzeugungsfenster**. Es kann über die Schaltfläche **Neu** im **Startfenster** erreicht werden. Etwaige Eingaben in den Freitextfeldern **Server** und **Wissensnetz** werden ignoriert.



2.2.1 Server

Im Freitextfeld **Server** wird der Name oder die IP-Adresse des Rechners angegeben, auf dem der Mediator läuft, über den das neue Wissensnetz angelegt werden soll. Sollte dieser nicht über den Standard-Port erreichbar sein, muss außerdem die korrekte Port-Nummer genannt werden. Die Eingabeform hierzu lautet *[Rechnername oder IP-Adresse]*:*[Portnummer]*.

Läuft der Mediator, über den das neue Wissensnetz angelegt werden soll, auf dem gleichen Rechner wie das Admin-Tool, kann er auch über den Rechnernamen *localhost* angesprochen werden.

Bleibt das Feld leer, wird das Wissensnetz im relativ zur Position des Admin-Tools direkten Unterordner *volumes* erzeugt.

2.2.2 Neues Wissensnetz

Im Freitextfeld **Neues Wissensnetz** wird der Name des Wissensnetzes festgelegt. Die dafür erlaubten Zeichen werden über das Dateisystem des Betriebssystems vorgegeben, auf dem das Wissensnetz gespeichert werden soll. Damit die Daten auch auf unterschiedlichen Dateisystemen gespeichert werden können, gilt:

- maximal 64 Zeichen
- keine Leerzeichen am Anfang oder Ende
- erlaubte Zeichen: große und kleine lateinische Buchstaben, Ziffern, Leerzeichen, !@#$%&'()+-.[]∧_‘{}∼Œœ sowie ASCII-Zeichen 160-255
- nicht erlaubte Zeichenfolgen sind: AUX, CON, NUL, PRN sowie COM0-COM9 und LPT0-LPT9

Die Vergabe eines Namens ist zwingend.

Der Name lässt sich später nur bei Kopiervorgängen des Wissensnetzes oder über Umbenennungen der Datei- und Verzeichnisnamen ändern. Bei einer Änderung ist zu beachten, dass der Name des Wissensnetzes eventuell in Initialisierungsdateien verwendet wird und möglicherweise die Lizenz darauf angepasst wurde.

2.2.3 Passwort (Mediator)

Der Mediator unterstützt eine Authentifizierung über ein Passwort. Ist für den Mediator, über den das neue Wissensnetz angelegt werden soll, ein Passwort gesetzt, muss es im Freitextfeld **Passwort**, welches sich zwischen den Feldern **Neues Wissensnetz** und **Lizenz** befindet, eingegeben werden. Ist kein Passwort vergeben, muss das Freitextfeld leer bleiben.

2.2.4 Lizenz

Ein Wissensnetz muss eine gültige Lizenz besitzen, damit der Knowledge-Builder und andere Software-Komponenten (mit Ausnahme des Admin-Tools) damit arbeiten können. Über die Schaltfläche **...** kann auf das Dateisystem des Betriebssystems zugegriffen werden, um einen Lizenzschlüssel (Dateiname: *[Lizenzname].key*) zu laden.

2.2.5 Benutzername

Im Freitextfeld **Benutzername** wird der Name des ersten im Wissensnetz registrierten Nutzers festgelegt. Die Art und Menge der dafür erlaubten Zeichen ist nicht beschränkt.Die Voreinstellung Administrator ist lediglich ein Vorschlag. Dieses Feld darf nicht leer bleiben.

Der Name kann im Admin-Tool oder im Knowledge-Builder nachträglich geändert werden. Der hierüber angelegte Nutzer besitzt automatisch Administratorrechte.

2.2.6 Passwort (Benutzer)

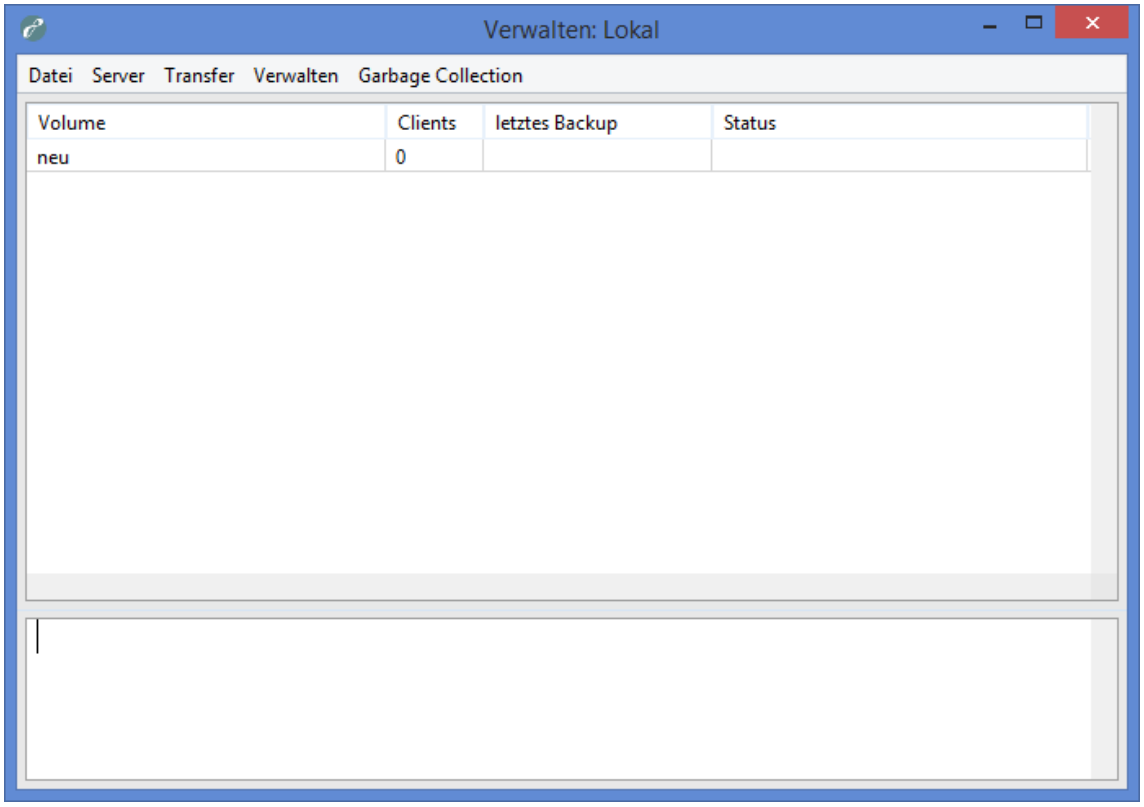
Im Freitextfeld **Passwort** kann ein Passwort für den ersten im Wissensnetz registrierten Nutzer vergeben werden. Dieses Passwort wird später gebraucht, wenn dieser Nutzer sich im Knowledge-Builder oder im Admin-Tool für das Wissensnetz anmelden will.

2.2.7 OK und Abbrechen

Die Schaltfläche **Ok** erzeugt das Wissensnetz unter Einbeziehung der eingegebenen Daten. Die Schaltfläche **Abbrechen** bricht den Vorgang ab. In beiden Fällen erfolgt eine Rückkehr zum **Startfenster**.

2.3 Serververwaltung

Die Gesamtnetzverwaltung erlaubt die Administration aller Wissensnetze eines Mediators beziehungsweise des lokalen Unterordners *volumes*. Sie kann über die Schaltfläche **Verwalten** im **Startfenster** erreicht werden. Erforderlich ist hierzu eine entsprechende Eingabe im Feld **Server** des **Startfensters**. Etwaige Eingaben im Feld **Wissensnetz** des **Startfensters** werden ignoriert. Werden die zu administrierenden Wissensnetze über einen Mediator angesprochen, muss außerdem das korrekte Mediator-Passwort in einem eigenen Fenster angegeben werden.



Das **Gesamtnetzverwaltungsfenster** besteht aus einer tabellarischen **Netzübersicht**, einem **Nachrichtenfeld** und einer **Menüzeile**.

2.3.1 Netzübersicht

Die tabellarische **Netzübersicht** gibt Aufschluss über

- den Namen (*Volume*),
- die Anzahl an gegenwärtig aktiven Nutzern (*Clients*),
- das Datum und die Uhrzeit der letzten Sicherung (*letztes Backup*) sowie
- die letzte Statusmeldung (*Status*) des jeweiligen Netzes.

Die einzelnen Spalten sind über einen Klick auf den Spaltenkopf sortierbar.

Die Daten werden nur beim Auslösen von Operationen aktualisiert und sind deswegen nicht immer aktuell. Eine manuelle Aktualisierung kann jederzeit über den Menüpunkt **Server** –> **Aktualisieren** forciert werden.

2.3.2 Nachrichtenfeld

Das **Nachrichtenfeld** gibt alle Statusmeldungen aller Netze aus. Statusmeldungen werden durch das Auslösen von Aktivitäten im Admin-Tool erzeugt. Sie gehen beim Schließen des Admin-Tools verloren. Um dies zu verhindern, können sie über den Menüpunkt **Datei** –> **Administrations-Log** exportiert werden. Das **Nachrichtenfeld** ist zwar editierbar, Änderungen werden beim Export aber ignoriert.

2.3.3 Menüzeile

Die **Menüzeile** besteht aus den folgenden Menüreitern:

2.3.3.1 Datei

Administrations-Log speichern speichert alle Einträge im Nachrichtenfenster in einer Textdatei (Dateistandardname: *admin.log*) ab. Name und Speicherort können in einem Speicherdialog frei gewählt werden. Diese Operation setzt voraus, dass das Admin-Tool mit einem Mediator verbunden ist.

Abmelden schließt die Serververwaltung und öffnet wieder das Anmeldefenster-

Beenden schließt die Serververwaltung

2.3.3.2 Server

Aktualisieren ermittelt die in der **Netzübersicht** im **Gesamtnetzverwaltungsfenster** erhobenen Daten neu.

Ini-Datei neu einlesen veranlasst den Server, seine ini-Datei neu einzulesen. Dabei können nicht alle Optionen im laufenden Betrieb aktualisiert werden. Der Server leifert eine Mitteilung über aktualisierte Optionen.

Log herunterladen erzeugt eine Kopie der üblicherweise im Ordner des verbundenen Mediators liegenden Mediator-Protokolldatei (Standarddateiname: *mediator.log*). Name und Speicherort der Kopie können in einem Speicherdialog frei gewählt werden. In der Mediator-Protokolldatei wird ein Protokoll über alle Aktivitäten des Mediators seit seiner ersten Inbetriebnahme geführt.

Serververbindungen gibt im **Nachrichtenfeld** die Nummer und die IP-Adresse aller gegenwärtig über den verbundenen Mediator in Wissensnetzen angemeldeten Software-Komponenten (außer dem Blob-Service) aus und gruppiert diese nach Wissensnetzen. Die Nummer wird vom Mediator fortlaufend generiert und bei jeder Neuanmeldung einer Software-Komponente neu vergeben.

2.3.3.3 Verwalten

Volume herunterladen erzeugt eine Kopie des in der **Netzübersicht** ausgewählten Wissensnetzes und speichert sie lokal im relativ zur Position des Admin-Tools liegenden Unterordner *volumes*. In einem separat erscheinenden Freitextfeld kann ein neuer Name für diese Kopie vergeben werden.

Volume kopieren erzeugt eine Kopie des in der **Netzübersicht** ausgewählten Wissensnetzes und speichert sie im gleichen Ordner wie das Originalnetz. In einem separat erscheinenden Freitextfeld muss ein neuer Name für diese Kopie vergeben werden.

Volume hochladen erzeugt eine Kopie eines ausgewählten lokalen Wissensnetzes und speichert sie im relativ zum verbundenen Mediator liegenden Unterordner *volumes*. In einem separat erscheinenden Freitextfeld kann ein neuer Name für diese Kopie vergeben werden. Die Auswahl des lokalen Wissensnetzes, das im relativ zur Position des Admin-Tools liegenden Unterordner *volumes* abgelegt sein muss, erfolgt über ein separates Auswahlfenster.

Volume austauschen erzeugt eine Kopie eines ausgewählten lokalen Wissensnetzes und überschreibt damit das in der **Netzübersicht** ausgewählte Wissensnetz. Die Kopie erhält

dabei den Namen des überschriebenen Wissensnetzes. Die Auswahl des lokalen Wissensnetzes, das im relativ zur Position des Admin-Tools liegenden Unterordner *volumes* abgelegt sein muss, erfolgt über ein separates Auswahlfenster.

In Folge der über Transfer-Operationen ausgelösten Kopierprozesse wird die Blockbelegung der Cluster und Blobs innerhalb der Wissensnetzkopien neu festgelegt und dabei deren Platzverbrauch optimiert. Der hierdurch bewirkte Komprimierungseffekt ist der gleiche wie über die Operation **Verwalten** -> **Volume komprimieren**.

Mit Ausnahme der Operation **Volume kopieren** setzen alle diese Operationen voraus, dass das Admin-Tool mit einem Mediator verbunden ist.

2.3.3.4 Verwalten

Admintool öffnen meldet sich im ausgewählten Volume mit dem Admintool an. Hiefür ist keine Authentifizierung im Volume nötig - die Mediator-Authentifizierung genügt.

Auf diese Art kann bei einem vergessenen Administrator-Passwort auf die Benutzerverwaltung des Volumes zugegriffen werden.

Backup erstellen erzeugt eine Sicherungskopie des in der **Netzübersicht** ausgewählten Wissensnetzes und speichert sie im relativ zur Position dieses Wissensnetzes liegenden Parallelordner *backup*. Für jede Sicherungskopie wird dort ein eigener Unterordner angelegt, in dessen Bezeichnung der Zeitpunkt der Kopieerstellung auf eine Sekunde genau eingearbeitet ist.Jede Sicherungskopie ist eine vollständige Kopie des Originalnetzes.

Vor Erzeugung der Sicherungskopie wird über ein eigenes Fenster erfragt, ob der Nutzer bis zum Abschluss des Kopiervorganges warten will. Gegebenenfalls wird bis zu diesem Zeitpunkt die weitere Nutzung des Admin-Tools blockiert. Andernfalls startet der Kopiervorgang im Hintergrund und eine Nachricht über den Fortgang des Kopiervorganges oder dessen Fertigstellung unterbleibt.

Backup wiederherstellen erzeugt eine Kopie einer ausgewählten Sicherungskopie und speichert sie im gleichen Ordner wie die in der **Netzübersicht** dargestellten Wissensnetze. In einem separat erscheinenden Freitextfeld muss ein neuer Name für diese Kopie vergeben werden. Die Auswahl der Sicherungskopie, die in einem Unterordner des Ordners *backup* abgelegt sein muss, der parallel zur Position der in der **Netzübersicht** dargestellten Wissensnetze liegt, erfolgt über zwei separate Auswahlfenster, in denen zuerst das Wissensnetz und danach die nach Erstellungszeiten sortierte Version ausgewählt werden.

Backup löschen löscht eine ausgewählte Sicherungskopie. Die Auswahl dieser Sicherungskopie, die in einem Unterordner des Ordners *backup* abgelegt sein muss, der parallel zur Position der in der **Netzübersicht** dargestellten Wissensnetze liegt, erfolgt über zwei separate Auswahlfenster, in denen zuerst das Wissensnetz und danach die nach Erstellungszeiten sortierte Version ausgewählt werden.

Die Blockbelegung der Cluster und Blobs innerhalb des Originalwissensnetzes wird bei der Erstellung einer Wissensnetzkopie nicht verändert. Der durch Backup-Operationen ausgelöste Kopiervorgang erzeugt deswegen keinen Komprimierungseffekt.

Volume löschen löscht das in der **Netzübersicht** ausgewählte Wissensnetz.

Volume komprimieren verringert den Platzbedarf des in der **Netzübersicht** ausgewählten Wissensnetzes. Dies geschieht über die Beseitigung ungenutzter Binnenblöcke. Durch Umkopierprozesse der Cluster und Blobs werden zunächst alle ungenutzten Blöcke an das Dateiende verschoben und dann im Dateisystem des Betriebssystems freigegeben.

Volume-Storage aktualisieren aktualisiert die Version des Blockdateisystems des in der

Netzübersicht ausgewählten Wissensnetzes. Wird das Wissensnetz über einen Mediator angesprochen, wird die darin enthaltene Version verwendet, ansonsten wird die im Admin-Tool mitgelieferte Version verwendet. Die Aktualisierung ermöglicht eine schnellere Speicherung von Indexstrukturen. Sie ist möglich für Wissensnetze, deren i-views-Core-Komponente älter als 4.2 ist.

2.3.3.5 Garbage Collection

Die Garbage-Collection ist ein Verfahren, das nicht mehr referenzierte Objekte (nach programmierterminologischer Lesart) in einem Wissensnetz löscht und damit den Speicherverbrauch des Wissensnetzes minimiert. Die Nutzung der Garbage-Collection setzt voraus, dass das zu bereinigende Wissensnetz über einen Mediator angesteuert wird.

Start startet eine neue Speicherbereinigung für das in der **Netzübersicht** ausgewählte Wissensnetz oder setzt eine pausierte Speicherbereinigung fort. Es erfolgt keine Rückmeldung, wann der Vorgang abgeschlossen ist. Der Stand des Fortschritts kann über den Menüpunkt **Status** in Erfahrung gebracht werden.

Pause unterbricht die Durchführung der aktiven Speicherbereinigung für das in der **Netzübersicht** ausgewählte Wissensnetz.

Anhalten bricht die Durchführung der aktiven Speicherbereinigung für das in der **Netzübersicht** ausgewählte Wissensnetz ab.

Status schreibt den aktuellen Zustand der Speicherbereinigung für das in der **Netzübersicht** ausgewählte Wissensnetz in die Statusspalte der **Netzübersicht** und in das **Nachrichtenfeld**. Ist eine Speicherbereinigung aktiv, erfolgt zusätzlich eine Rückmeldung über den Stand des Fortschritts in Form einer Prozentangabe.

2.4 Einzelnetzverwaltung

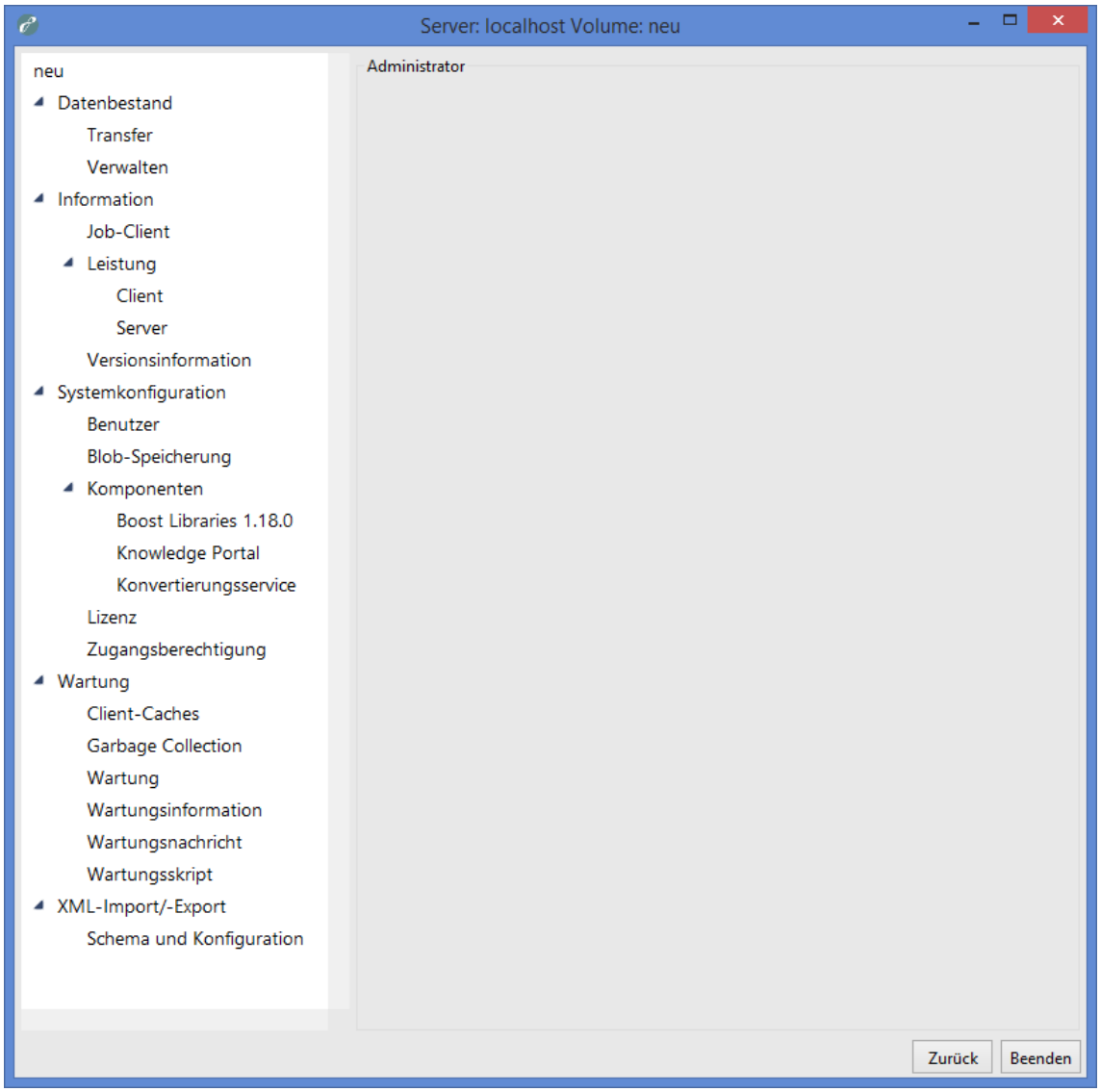
Die Einzelnetzverwaltung erlaubt die Administration eines einzelnen Wissensnetzes. Sie kann über die Schaltfläche **Start** im Startfenster erreicht werden. Erforderlich sind hierzu entsprechende Eingaben in den Feldern **Server**, **Wissensnetz**, **Benutzer** und **Passwort** des Startfensters.

2.4.1 Nutzerauthentifizierung

Für den Zutritt zum **Einzelnetzverwaltungsfenster** ist die Anmeldung eines Nutzers mit Administratorrechten notwendig.

Falls man keinen Zugang mehr zum Wissensnetz mehr hat, kann man mit Hilfe der Anmeldung in der **Serververwaltung** über die Authentifizierung am Server Zugang zum Wissensnetz bekommen.

2.4.2 Einzelnetzverwaltungsfenster



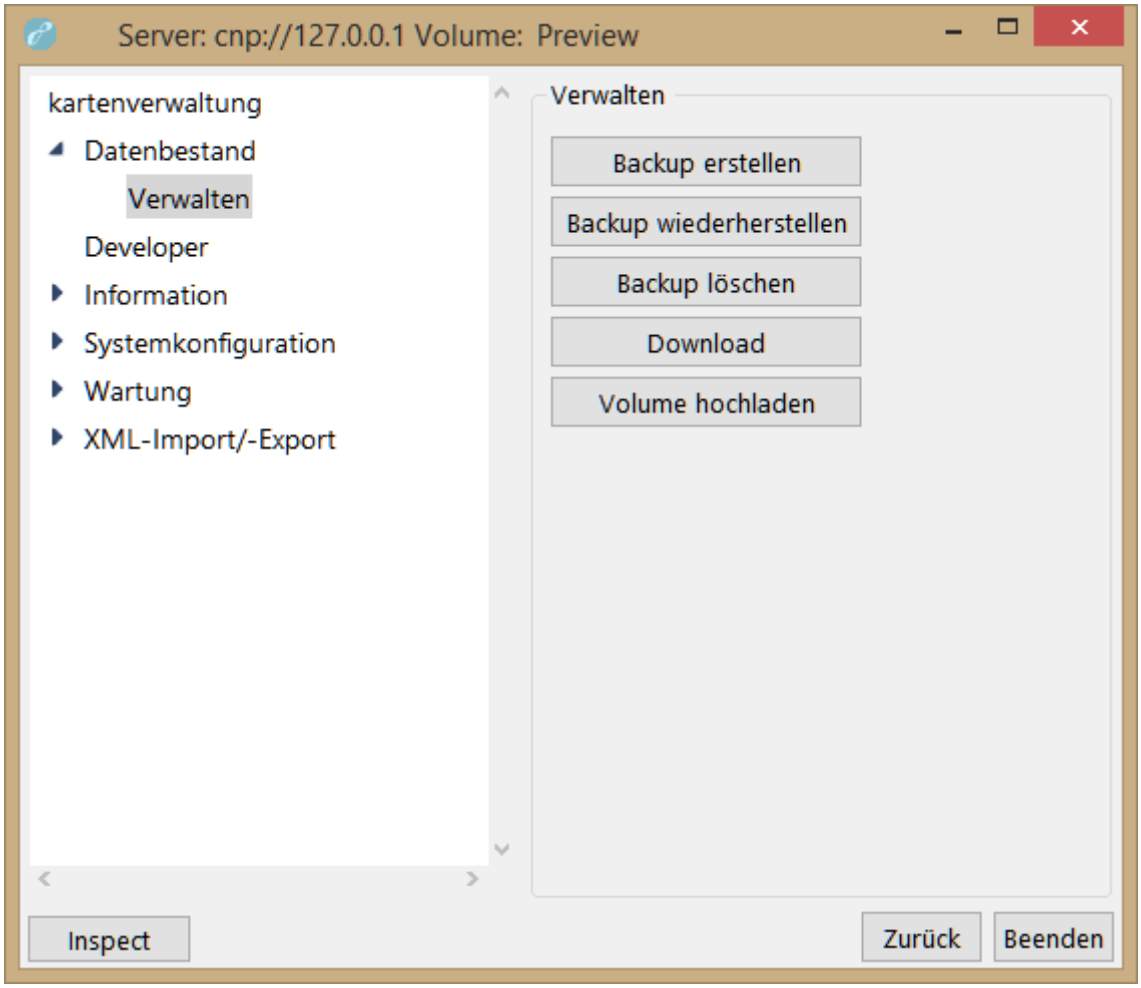
Das **Einzelnetzverwaltungsfenster** verfügt über eine mehrstufig gegliederte Menüliste links und ein Operationsfenster rechts. Der Inhalt des Operationsfensters richtet sich nach dem gewählten Menüpunkt in der Menüliste.

Die Schaltfläche **Zurück** wechselt zum Startfenster zurück.

Die Schaltfläche **Beenden** schließt das Admin-Tool.

Wird das zu administrierende Wissensnetz ohne Mediator angesprochen, ist ein Zutritt anderer Nutzer zum Wissensnetz über den Knowledge-Builder oder eine weitere Instanz des Admin-Tools nicht möglich, solange das **Einzelnetzverwaltungsfenster** geöffnet ist.

2.4.2.1 Datenbestand verwalten



Backup erstellen erzeugt eine Sicherungskopie des Wissensnetzes und speichert sie (auf dem Server) im relativ zur Position dieses Wissensnetzes liegenden Parallelordner *back-up*. Für jede Sicherungskopie wird dort ein eigener Unterordner angelegt, in dessen Bezeichnung der Zeitpunkt der Kopierstellung auf eine Sekunde genau eingearbeitet ist.Jede Sicherungskopie ist eine vollständige Kopie des Originalnetzes.

Vor Erzeugung der Sicherungskopie wird über ein eigenes Fenster erfragt, ob der Nutzer bis zum Abschluss des Kopiervorganges warten will. Gegebenenfalls wird bis zu diesem Zeitpunkt die weitere Nutzung des Admin-Tools blockiert. Andernfalls startet der Kopiervorgang im Hintergrund und eine Nachricht über den Fortgang des Kopiervorganges oder dessen Fertigstellung unterbleibt.

Backup wiederherstellen ersetzt das aktuelle Wissensnetz durch eine Sicherungskopie (danach wird man automatisch abgemeldet). Die Auswahl der Sicherungskopie erfolgt über die Zeitpunkte der jeweiligen Backups.

Backup löschen löscht eine einzelne Sicherungskopie dieses Wissensnetzes.

Die Blockbelegung der Cluster und Blobs innerhalb des Originalwissensnetzes wird bei der Erstellung einer Wissensnetzkopie nicht verändert. Der durch Backup-Operationen ausgelöste Kopiervorgang erzeugt deswegen keinen Komprimierungseffekt.

Download erzeugt eine Kopie des Wissensnetzes und speichert sie lokal im relativ zur Posi-

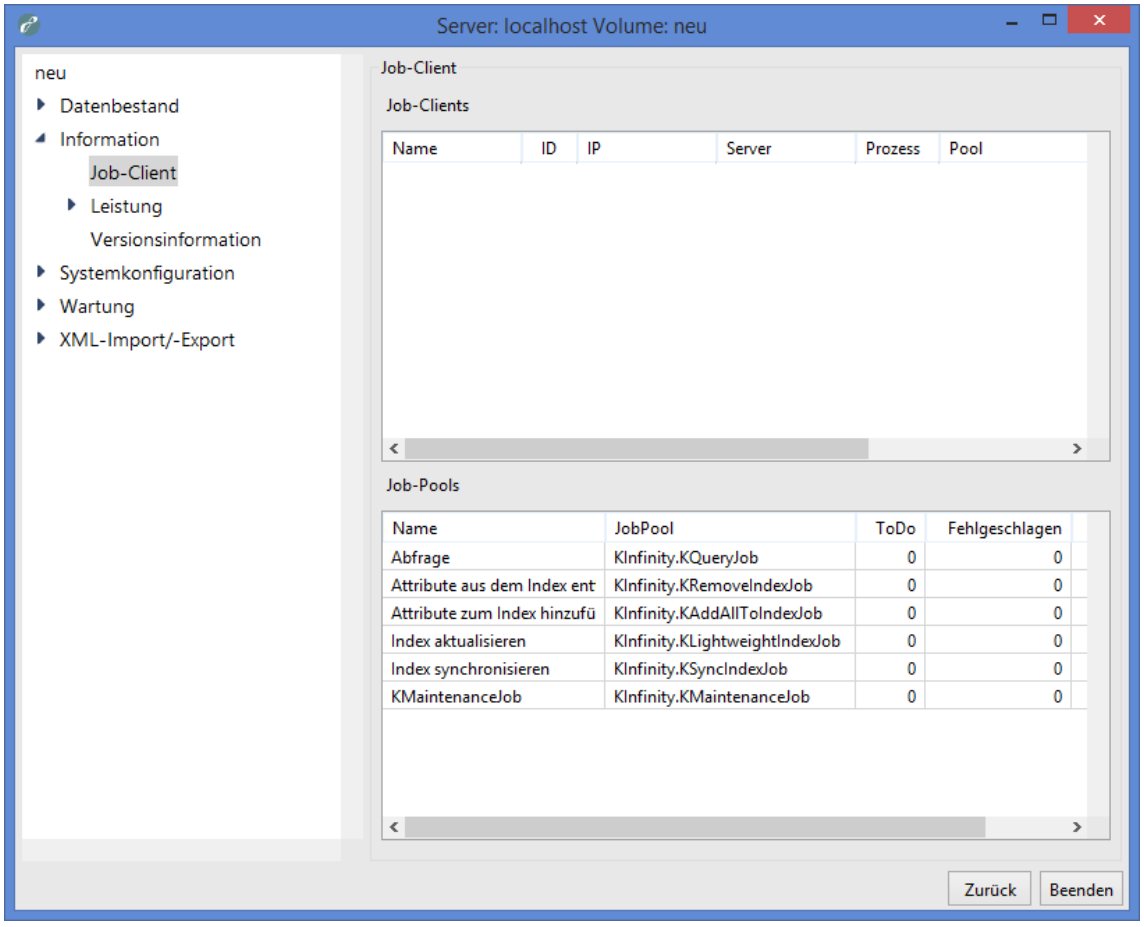
tion des Admin-Tools liegenden Unterordner *volumes*. In einem separat erscheinenden Freitextfeld kann ein neuer Name für diese Kopie vergeben werden.

Volume hochladen überträgt ein lokal gespeichertes Netz und ersetzt das aktuelle Wissensnetz durch dieses (danach wird man automatisch abgemeldet)

2.4.2.2 Information

2.4.2.2.1 Job-Client

Um den Knowledge-Builder von bestimmten rechenintensiven Prozessen wie der Indizierung und der Abfrage von Wissensnetzen sowie der Ausführung von Skripten zu entlasten, können diese Prozesse teils wahlweise, teils exklusiv als Aufgaben (Jobs) von Job-Clients (einem Software-Dienst) übernommen werden. Dazu muss über die Benutzeroberfläche des Knowledge-Builders oder per Skript eine Aufgabe ausgelöst oder es müssen die Bedingungen für ihre Auslösung festgelegt werden. Außerdem ist mindestens ein Job-Client zu konfigurieren und zu starten, der Aufgaben dieses Aufgabentyps (Job Pool) übernehmen kann. Das Admin-Tool übernimmt hierbei überwiegend eine Beobachtungsfunktion. Unerledigte Aufgaben erscheinen im Knowledge-Builder unter dem Eintrag *Aufträge* in der Rubrik *Technik*. Die Verwaltung von Job-Clients über das Admin-Tool setzt voraus, dass das Admin-Tool mit einem Mediator verbunden ist.



Die tabellarische **Job-Clients-Übersicht** zeigt für jeden aktuell laufenden Job-Client

- seinen Namen im Format [Job-Client-Name]@[Mediator-Name] (*Name*),
- seine Job-Client-Nummer (*ID*),

- seine IP-Adresse (*IP*),
- den Namen des mit ihm verbundenen Mediators (*Server*),
- seine vom Betriebssystem vergebene Prozessnummer (*Prozess*),
- die ihm zugeordneten Aufgabentypen (*Pool*),
- seinen Arbeitsstatus (*Status*) und
- die Anzahl von ihm erledigter Aufgaben (*Erledigt*).

Die Job-Client-Nummer wird vom Mediator fortlaufend generiert und bei jeder Neuanmeldung neu vergeben. Der Job-Client-Name und die dem Job-Client zugeordneten Aufgabentypen werden in der Initialisierungsdatei des jeweiligen Job-Clients (Standarddateiname: *jobclient.ini*) unter dem Schlüssel *name* respektive dem Schlüssel *jobPools* festgelegt. In der Job-Client-Übersicht wird jeder Aufgabentyp eines Job-Clients in einer eigenen Zeile dargestellt, so dass ein Job-Client regelmäßig mehrere Zeilen belegt.

Die einzelnen Spalten der **Job-Clients-Übersicht** sind über einen Klick auf den Spaltenkopf sortierbar. Per Rechtsklick auf eine Zeile kann außerdem ein Kontextmenü geöffnet werden:

- **Informationen anzeigen** stellt alle in der ausgewählten Zeile gelisteten Daten mit Ausnahme des Aufgabentyps und der erledigten Aufgabenanzahl in einem neuen Fenster dar. Ergänzt werden
  - Datum und Uhrzeit des letzten Startzeitpunkts des Job-Clients (*startUpTime*),
  - die ihm zur Verfügung stehende maximal nutzbare Menge an Systemarbeitsspeicher in Byte (*max Memory*),
  - der Name seiner Protokolldatei (*logFileName*) und
  - sein spezieller Name, unter dem er zum Beenden gezwungen werden kann (eine Verkettung der Zeichenkette „jobclient“ und der Job-Client-Nummer) (*shutDownString*).
- Die Daten können dort in die Zwischenablage des Betriebssystems kopiert (Schaltfläche **In Zwischenablage kopieren**) oder über einen Speicherdialog als frei benennbare Textdatei an eine beliebige Stelle exportiert werden (Schaltfläche **Speichern**).
- Die über den Menüpunkt **Informationen anzeigen** ausgelöste Operation kann alternativ über einen Doppelklick auf eine Zeile in der Job-Clients-Übersicht erwirkt werden.
- **Job-Client entfernen** beendet den in der **Job-Clients-Übersicht** ausgewählten Job-Client.
- **Alle Job-Clients** entfernen beendet alle in der **Job-Clients-Übersicht** gelisteten Job-Clients.

Die tabellarische **Job-Pools-Übersicht** listet alle Aufgabentypen, die in der **Job-Clients-Übersicht** mindestens einem Job-Client zugeordnet sind. Für jeden Aufgabentyp werden

- seine Bezeichnung (*Name*),
- seine in der Job-Clients-Initialisierungsdatei verwendete technische Bezeichnung (*JobPool*),
- die Anzahl unerledigter Aufgaben dieses Aufgabentyps (*ToDo*),
- die Anzahl fehlgeschlagener Aufgaben dieses Aufgabentyps (*Fehlgeschlagen*) und
- die Anzahl der ihm zur Verfügung stehender Job-Clients (*Job-Clients*)

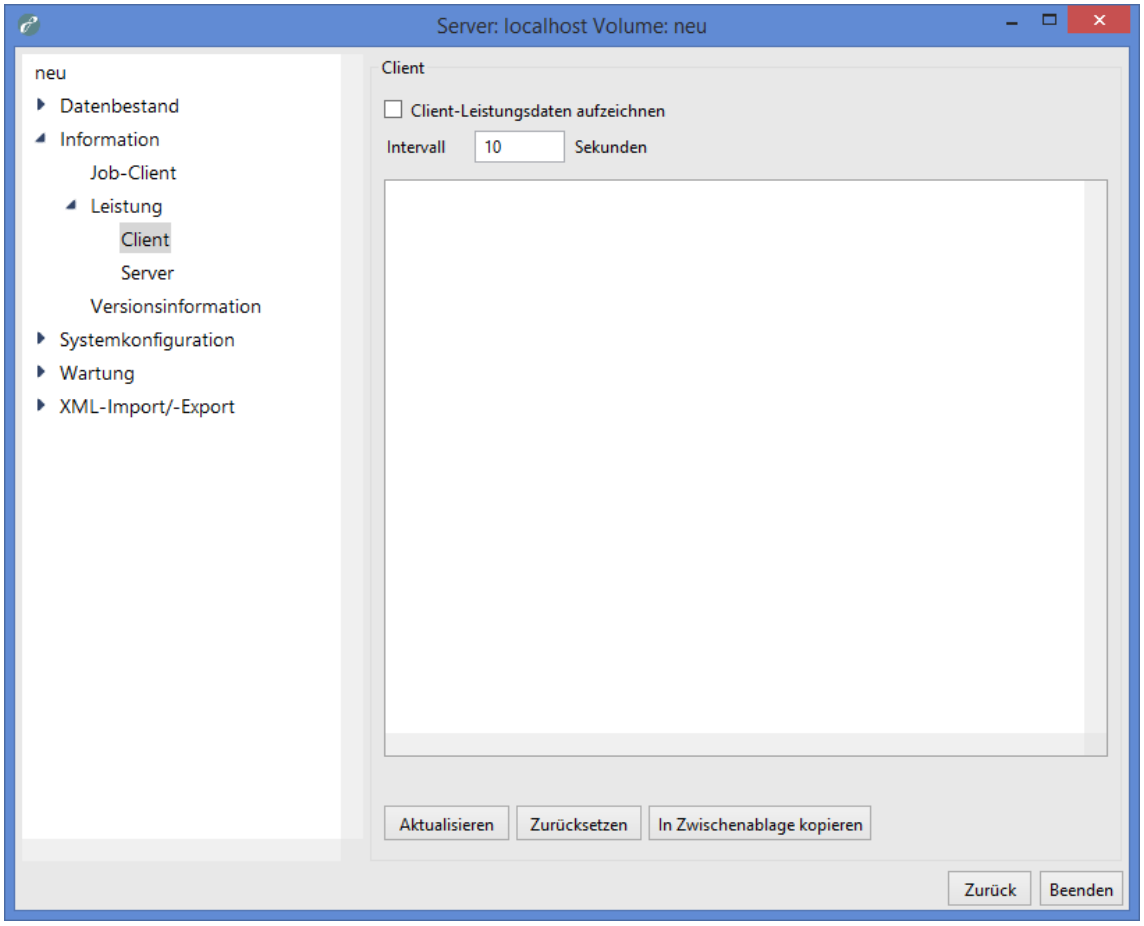
genannt.

Die einzelnen Spalten der **Job-Pools-Übersicht** sind über einen Klick auf den Spaltenkopf sortierbar. Per Rechtsklick auf eine Job-Client-Zeile kann außerdem ein Kontextmenü geöffnet werden:

- **Job-Pool leeren** löscht alle unerledigten und fehlgeschlagenen Aufgaben des in der **Job-Pools-Übersicht** ausgewählten Aufgabentyps. Diese Operation ist nur möglich, wenn kein Job-Client läuft.
- **Zu ignorierende Fehlermeldungen konfigurieren** ermöglicht die Unterdrückung bestimmter Fehlermeldungen bei der Ausführung von Aufgaben des in der **Job-Pools-Übersicht** ausgewählten Aufgabentyps. Wird eine Fehlermeldung auf diese Weise unterdrückt, bleibt die dem Fehler zugehörige Aufgabe bei der Ermittlung der Anzahl fehlgeschlagener Aufgaben in der **Job-Pools-Übersicht** unberücksichtigt. Diese Operation ist nur möglich, wenn bereits Aufgaben des in der **Job-Pools-Übersicht** ausgewählten Aufgabentyps auf ihre Bearbeitung warten oder bereits bearbeitet wurden.
- Die Verwaltung der zu unterdrückenden Fehlermeldungen erfolgt in einem separaten Fenster:
  - In der alphabetisch sortierten **Fehlermeldungsliste** werden alle zu unterdrückenden Fehlermeldungen gelistet. Eine Fehlermeldung wird unterdrückt, wenn ihr Ausgabetext mit einem Text in der **Fehlermeldungsliste** übereinstimmt.
  - **+** erlaubt die Eingabe einer zu unterdrückenden Fehlermeldung über ein eigenes Fenster. Die eingegebene Fehlermeldung erscheint in der **Fehlermeldungsliste**.
  - **...** erlaubt die Änderung der in der **Fehlermeldungsliste** ausgewählten Fehlermeldung.
  - **-** löscht die in der **Fehlermeldungsliste** ausgewählte Fehlermeldung.

2.4.2.2.2 Leistung

Client



Client-Leistungsdaten aufzeichnen startet und beendet die Erhebung diverser Leistungskennzahlen, die an Aktivitäten der mit dem Wissensnetz verbundenen Software-Komponenten gekoppelt sind. Diese Leistungskennzahlen können zur Performanzanalyse verwendet werden.

Intervall setzt die notwendige Zeitspanne in Sekunden, bis eine Software-Komponente erneut ein Datenpaket mit Leistungskennzahlen an das Admin-Tool sendet. Es kann nach dem Start der Aufzeichnung nicht mehr verändert werden. Die Vorerinstellung liegt bei 10 Sekunden.

In der **Leistungskennzahlenübersicht** werden die Leistungskennzahlen in geschachtelten Listenpunkten ausgegeben. Per Klick auf die links neben den Rubriken befindlichen Dreiecksymbole können Listenunterpunkte ein- und ausgeklappt werden. Alternativ lässt sich dies über ein Kontextmenü realisieren, das über einen Klick mit der rechten Maustaste auf einen Listenpunkt aufgerufen werden kann:

- **Expand** klappt alle direkten Listenunterpunkte des gewählten Listenpunkts aus.
- **Expand fully** klappt alle direkten und alle indirekten Listenunterpunkte des gewählten Listenpunkts aus.
- **Contract fully** klappt alle Listenunterpunkte des gewählten Listenpunkts wieder ein.

Mit einem Doppelklick auf einen Listenpunkt lassen sich alle darunter abgelegten Leistungskennzahlen in einem separaten Fenster auf einen Blick darstellen. Dort können sie in die Zwischenablage des Betriebssystems kopiert (Schaltfläche **In Zwischenablage kopieren**) oder über einen Speicherdialog als frei benennbare Textdatei an eine beliebige Stelle ex-

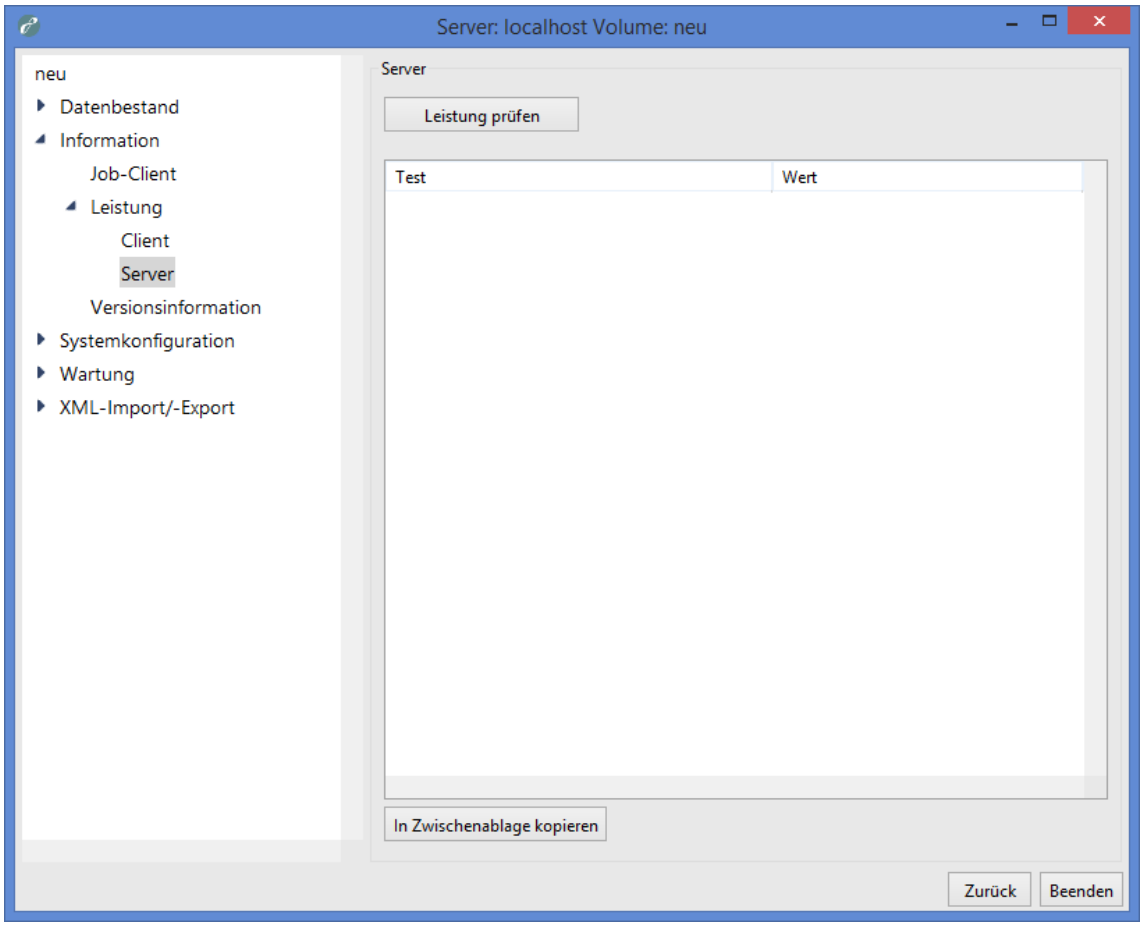
portiert werden (Schaltfläche **Speichern**).

Aktualisieren aktualisiert die in der **Leistungskennzahlenübersicht** dargestellten Leistungskennzahlen.

Zurücksetzen löscht die in der **Leistungskennzahlenübersicht** dargestellten Leistungskennzahlen.

In Zwischenablage kopieren kopiert diein der **Leistungskennzahlenübersicht** dargestellten Leistungskennzahlen in die Zwischenablage des Betriebssystems.

Server



Leistung prüfen startet einen Testvorgang, der die Performanz des angeschlossenen Mediators auswertet. Dabei werden vier Anfragen an den Mediator geschickt und die an das Admin-Tool gesendeten Antworten ausgewertet. Gemessen werden

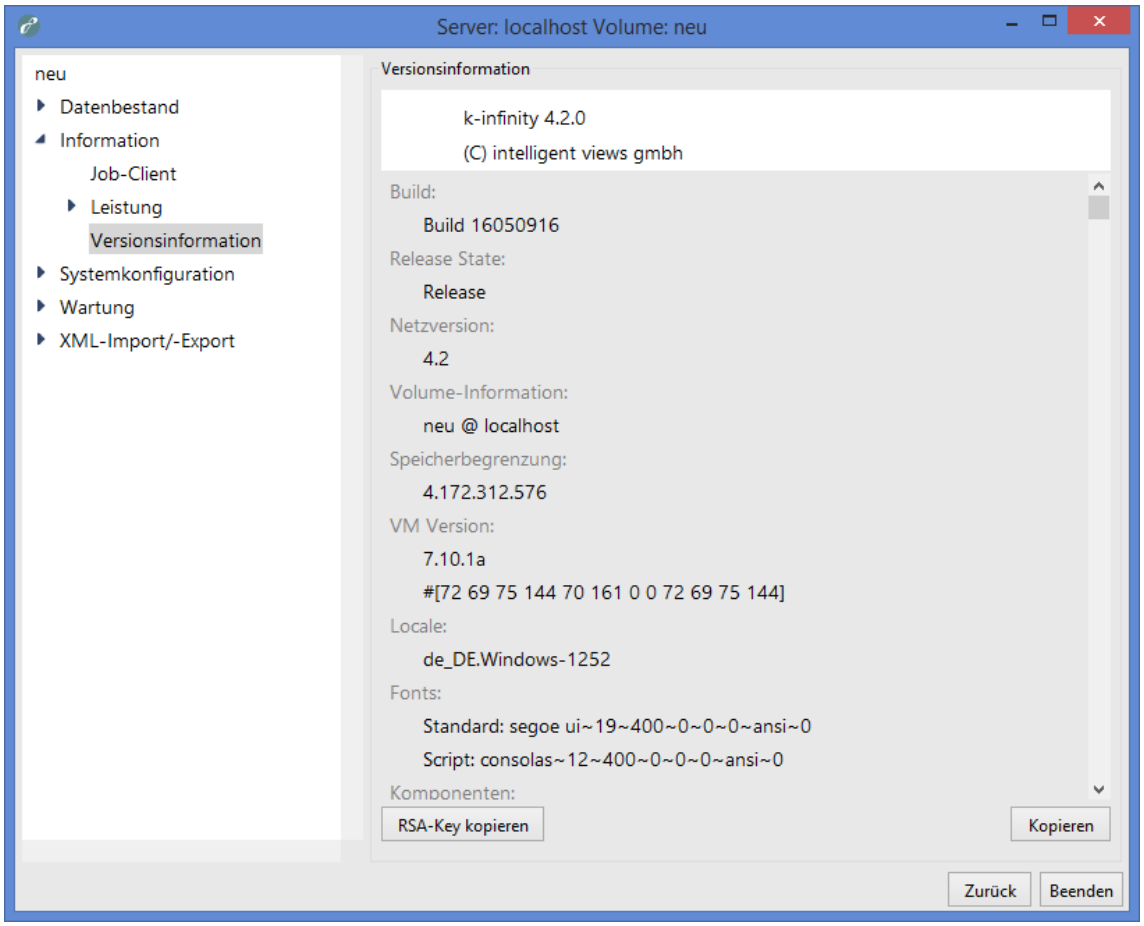
- die Zeiten bis zur Rücksendung einer kleinen Datei (*Roundtrip: Blob*) und
- des Ergebnisses einer Indexsuchanfrage (*Roundtrip: RPC*) sowie
- die durchschnittliche Übertragungsgeschwindigkeite beim Versand mehrerer Dateien der Größe 1 MB (*Throughput: Blob (1.0 MB)*) und
- die durchschnittliche Übertragungsgeschwindigkeite beim Versand mehrerer Dateien der Größe100KB (*Throughput: Blob (100.0 KB)*).

Die Testergebnisse werden in die bereitgestellte **Ergebnisliste** geschrieben. Die einzelnen Spalten der Tabelle sind über einen Klick auf den Spaltenkopf sortierbar.

In Zwischenablage kopieren kopiert die Testergebnisse der **Ergebnisliste** als Reintext in die Zwischenablage des Betriebssystems.

2.4.2.2.3 Versionsinformation

Über diesen Menüpunkt lassen sich diverse versionsspezifische Informationen über das Wissensnetz und das Admin-Tool abrufen.



Konkret handelt es sich dabei um

- die Versionsnummer des Admin-Tools (*Build*),
- den Veröffentlichungsstatus des Admin-Tools (*Release*),
- die Versionsnummer des Wissensnetzes (Netzversion), die Namen des Wissensnetzes und des verwendeten Mediators (*Volume-Information*),
- die vom Admin-Tool maximal nutzbare Menge an Systemarbeitsspeicher in Byte (*Speicherbegrenzung*),
- die Versionsnummer und der digitale Fingerabdruck der vom Admin-Tool verwendeten Ausführungsumgebung (*VM Version*),
- die im Betriebssystem aktive Spracheinstellung (*Locale*),
- die im Admin-Tool verwendeten, mitgelieferten Schriftarten (*Fonts*),
- die im Wissensnetz installierten Wissensnetzkomponenten inklusive Versionsnummer (*Softwarekomponenten*) und

- die im Admin-Tool verwendeten Smalltalk-Pakete inklusive Versionsnummer (*Pakete*).

Die Informationen werden in einem unsichtbaren Textfeld ausgegeben, welches über ein Kontextmenü verfügt, das per Rechtsklick aktiviert werden kann:

- **Select All** markiert den gesamten Text. Alternativ kann mit dem Mauszeiger ein beliebiger Textausschnitt markiert werden.
- **Copy** kopiert den gewählten Textbereich in die Zwischenablage des Betriebssystems.
- **Find Again** sucht nach dem gewählten Textbereich und findet sein nächstes Auftreten gemäß der Leserichtung.
- **Find** erlaubt die Eingabe einer zu suchenden Zeichenkette in einem separaten Fenster und findet ihr nächstes Auftreten gemäß der Leserichtung relativ zur Position der per Mausklick setzbaren Schreibmarke. Bei der Suche wird Groß- und Kleinschreibung unterschieden.

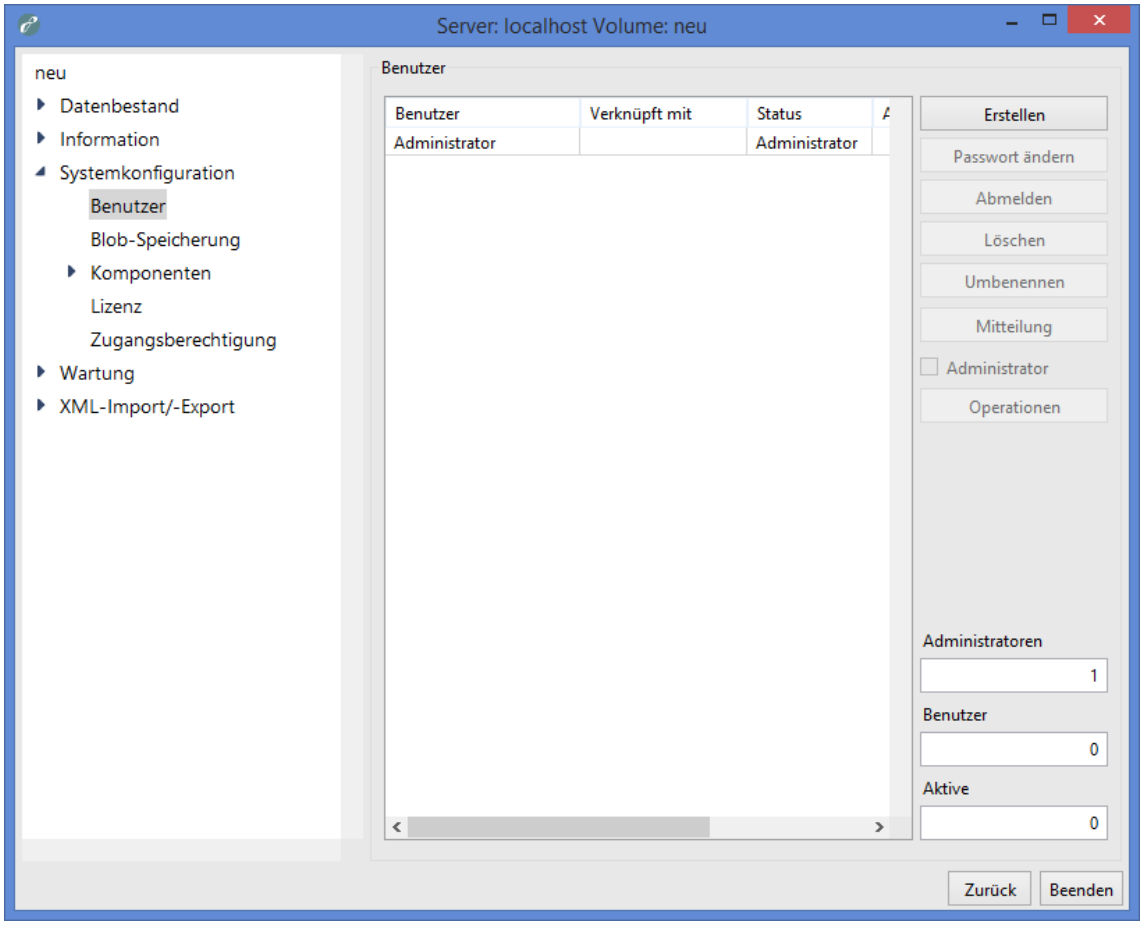
Über die Schaltfläche **Kopieren** werden alle Informationen in die Zwischenablage des Betriebssystems kopiert.

Die Schaltfläche **RSA-Key kopieren** kopiert den für jedes kompilierte Admin-Tool einzigartigen Schlüssel in die Zwischenablage des Betriebssystems. Dieser Schlüssel kann in die Initialisierungsdatei eines Mediators (Standarddateiname: *mediator.ini*) eingetragen werden und beschränkt dadurch den Zugang zu diesem Mediator über ein Admin-Tool auf Admin-Tools mit genau diesem Schlüssel.

2.4.2.3 Systemkonfiguration

2.4.2.3.1 Benutzer

Die Benutzerverwaltung gleicht derjenigen im Knowledge-Builder mit der Ausnahme, dass keine Verknüpfungen zwischen Nutzern und Objekten des nutzergenerierten Teilnetzes bearbeitet werden können.



Die tabellarische **Nutzerübersicht** zeigt für jeden im Wissensnetz registrierten Nutzer

- seinen Benutzernamen (*Benutzer*),
- mit welchem Objekt des nutzergenerierten Teilnetzes er verknüpft ist (*Verknüpft mit*),
- welchen Status er momentan besitzt (*Status*),
- an welchem Datum und zu welcher Uhrzeit er sich über den Knowledge-Builder im Wissensnetz angemeldet hat(*Anmeldedatum*), sofern er noch angemeldet ist, und
- mit welchem Verfahren das Passwort verschlüsselt ist (*Passworttyp*).

Die einzelnen Spalten der Tabelle sind über einen Klick auf den Spaltenkopf sortierbar.

Der *Status* gibt Auskunft darüber, ob ein Nutzer Administratorrechte besitzt, ob ein Nutzer mit Administratorrechten kein Passwort besitzt und ob ein Nutzer über den Knowledge-Builder im Wissensnetz angemeldet ist. Namen von Nutzern mit Administratorrechten ohne Passwort sind rot markiert.

Erstellen legt einen neuen Nutzer an. Benutzername (verpflichtend) und Passwort (optional) werden in einem eigenen Fenster festgelegt.Die Art und Menge der dafür erlaubten Zeichen ist nicht beschränkt.

Passwort ändern ändert das Passwort des in der **Nutzerübersicht** ausgewählten Nutzers. In zwei aufeinanderfolgenden Fenstern wird zweimal das neue Passwort eingegeben.

Abmelden meldet den in der **Nutzerübersicht** ausgewählten Nutzer nach einer Sicherheitsbestätigung aus dem Wissensnetz ab. Damit diese Operation eine Wirkung entfaltet, muss

dieser Nutzer aktuell über den Knowledge-Builder im Wissensnetz angemeldet sein.

Löschen löscht den in der **Nutzerübersicht** ausgewählten Nutzer nach einer Sicherheitsbestätigung. Mindestens ein Nutzer mit Administratorrechten muss verbleiben.

Umbenennen erlaubt über ein Freitextfeld in einem separaten Fenster die Vergabe eines neuen Benutzernamens für den in der **Nutzerübersicht** ausgewählten Nutzer. Bleibt das Freitextfeld leer, erfolgt keine Umbenennung.

Mitteilung sendet über ein Freitextfeld in einem separaten Fenster eine Nachricht an den in der **Nutzerübersicht** ausgewählten Nutzer. Die Nachricht wird im Wissensnetz zwischengespeichert und erscheint dem adressierten Nutzer in einem separaten Fenster im Knowledge-Builder, sobald er sich damit am Wissensnetz anmeldet. Der Nutzer kann auf diese Nachricht nicht antworten.

Administrator verleiht oder nimmt dem in der **Nutzerübersicht** ausgewählten Nutzer Administratorrechte. Damit ein Nutzer Administratorrechte erhalten kann, muss er ein Passwort besitzen. Nachdem er Administratorrechte besitzt, ist eine Löschung des Passworts indes möglich. Mindestens ein Nutzer muss Administratorrechte besitzen.

Operationen öffnet ein neues Fenster, in dem für den in der **Nutzerübersicht** ausgewählten Nutzer aus einer Liste von Operationen, namentlich

- Backup erstellen,
- Backup löschen,
- Backup wiederherstellen,
- Garage Collection,
- Kopieren,
- Log herunterladen,
- Volume herunterladen,
- Volume hochladen,
- Volume löschen,

diejenigen gewählt werden können, die dieser Nutzer im Rahmen der Einzelnetzverwaltung in Zukunft ohne Eingabe des Mediator-Passworts ausführen darf. Zur Bestätigung der Auswahl muss in das Freitextfeld **Server-Passwort für Operationen** das korrekte Mediator-Passwort eingegeben werden.

Die Operation **Operationen** ist nur wählbar für einen Nutzer mit Administratorrechten. Ihre Verwendung setzt außerdem voraus, dass ein Mediator-Passwort gesetzt wurde.

Das Feld **Administratoren** gibt die Anzahl aller im Wissensnetz registrierten Nutzer mit Administratorrechten an.

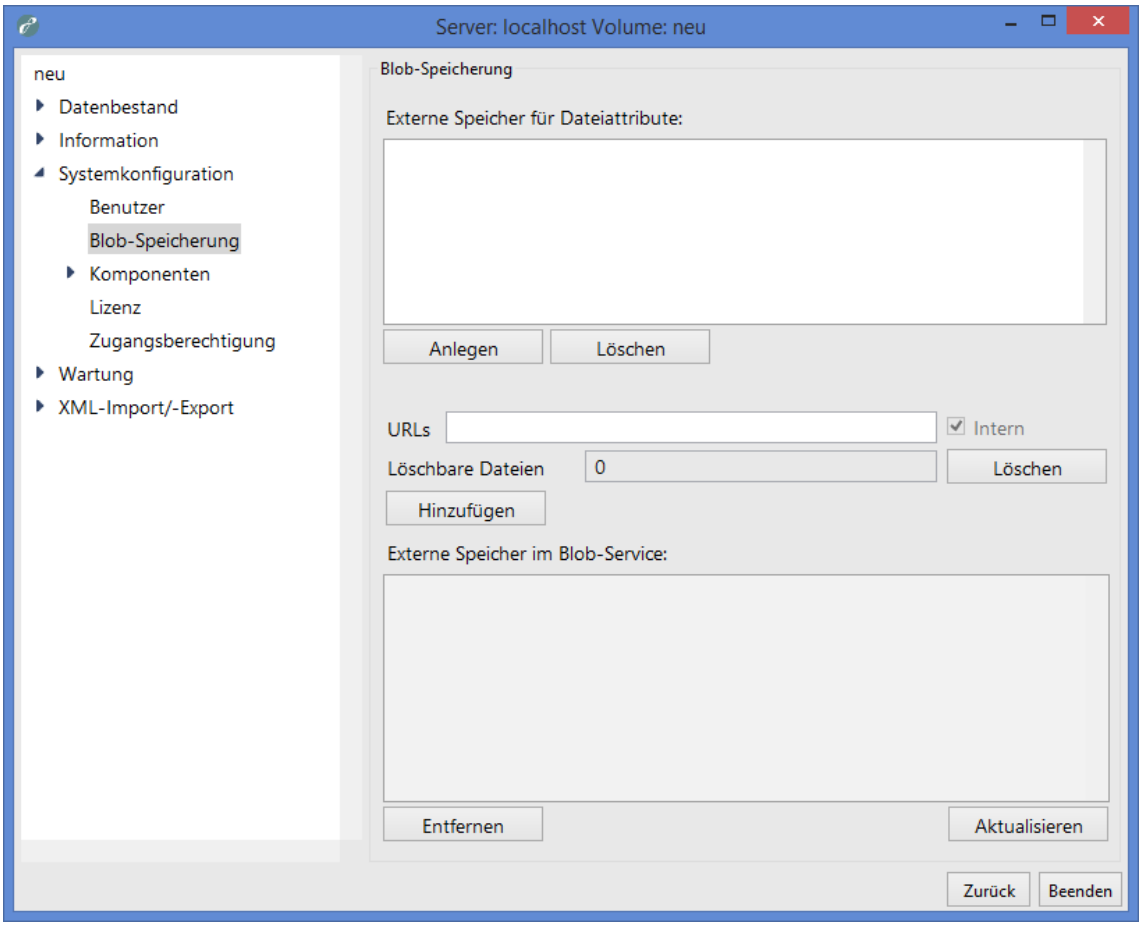
Das Feld **Benutzer** gibt die Anzahl aller im Wissensnetz registrierten Nutzer ohne Administratorrechte an.

Das Feld **Aktive** gibt die Anzahl aller gegenwärtig über den Knowledge-Builder im Wissensnetz angemeldeten Nutzer an.

2.4.2.3.2 Blob-Speicherung

Attributwerte von Attributen mit dem Attributdatentyp *Datei* (sogenannte Blobs) können auch wissensnetzextern in einem Blob-Speicher gespeichert werden. Dies hat den Vorteil, dass sie unabhängig vom Wissensnetz und damit bei Bedarf in einer anderen Systemumge-

bung verwaltet werden können. Um Blobs in einem Blob-Speicher zu sichern, muss der Blob-Speicher eingerichtet und mit einem konfigurierten Blob-Service (einem Software-Dienst) verbunden werden.



Anlegen erzeugt einen neuen Blob-Speicher. Er erscheint unter Verwendung des Namensformats *[Wissensnetz-ID]+[Blob-Speicher-ID]* in dem darüber liegenden Textfeld, der **Blob-Speicher-Gesamtübersicht**.

Löschen löscht den in der **Blob-Speicher-Gesamtübersicht** ausgewählten Blob-Speicher.

Das Zahlenfeld **Löschbare Dateien** zeigt die Anzahl der nicht mehr gebrauchten Blobs in dem in der **Blob-Speicher-Gesamtübersicht** ausgewählten Blob-Speicher. Blobs werden nicht mehr gebraucht, wenn ihre jeweiligen Attribute im Wissensnetz gelöscht wurden oder wenn die Verbindung zwischen Blob-Service und Blob-Speicher per Admin-Tool aufgehoben wurde.

Löschen löscht alle nicht mehr gebrauchten Blobs in dem in der **Blob-Speicher-Gesamtübersicht** ausgewählten Blob-Speicher.

Im Freitextfeld **URLs** kann ein Blob-Service identifiziert werden. Dies geschieht über die Eingabe der in der Initialisierungsdatei des zugehörigen Blob-Services (Standarddateiname: *blobservice.ini*) unter dem Schlüssel *interfaces* hinterlegten Netzadresse inklusive des Präfixes *http*. Soll der Blob-Service über mehrere Netzadressen angesprochen werden, können diese hintereinander mit Komma getrennt eingegeben werden.

Alternativ kann auch der im Mediator integrierte Blob-Service angesprochen werden. Dazu müssen in der Initialisierungsdatei des Mediators (Standarddateiname: *mediator.ini*) unter dem Schlüssel *startBlobService* der Wert *true* gesetzt und das Freitextfeld **URLs** leer gelassen

werden. Das rechts neben dem Freitextfeld **URLs** positionierte Kontrollkästchen **Intern** indiziert, ob der integrierte Blob-Service oder ein externer Blob-Service angesprochen wird. Die Konfiguration des im Mediator integrierten Blob-Services erfolgt nicht über die Mediator-Initialisierungsdatei, sondern über eine separate Initialisierungsdatei (Standarddateiname: *blobservice.ini*).

Hinzufügen verbindet den in der **Blob-Speicher-Gesamtübersicht** ausgewählten Blob-Speicher mit dem über das Freitextfeld **URLs** identifizierten Blob-Service. Dafür muss der Blob-Service aktiviert sein. Gelingt die Verknüpfung, erscheint der Blob-Speicher unter Verwendung des Namensformats *[Wissensnetz-ID]+[Blob-Speicher-ID]* in dem darunter liegenden Textfeld, der **Übersicht angemeldeter Blob-Speicher**.

Aktualisieren aktualisiert die **Übersicht angemeldeter Blob-Speicher**. Dazu muss ein Blob-Speicher in der **Blob-Speicher-Gesamtübersicht** ausgewählt sein.

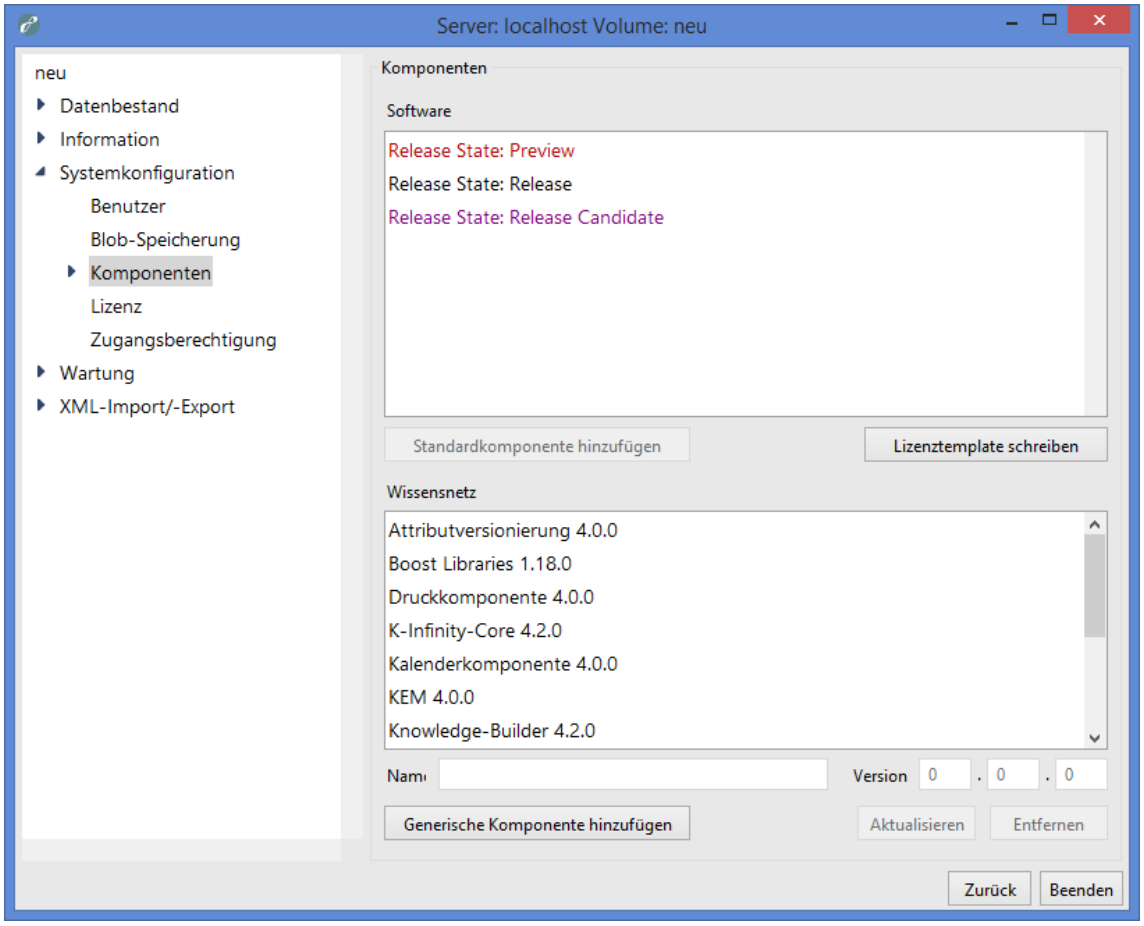
Entfernen unterbricht die Verbindung des in der **Übersicht angemeldeter Blob-Speicher** ausgewählten Blob-Speichers mit dem Blob-Service und entfernt den Blob-Speicher aus der Übersicht. Dabei verlieren alle im Blob-Speicher hinterlegten Blobs unwiderruflich ihre internen Verweise zu den jeweiligen Attributen im Wissensnetz und können im Wissensnetz nicht mehr abgerufen werden. Damit die Entfernung gelingt, muss der in der **Übersicht angemeldeter Blob-Speicher** ausgewählte Blob-Speicher auch in der **Blob-Speicher-Gesamtübersicht** ausgewählt sein.

Alle über einen Blob-Service gesicherten Blobs werden in einem relativ zur Position des Blob-Services liegenden Unterordner *blobs* abgelegt. Die interne Zuordnung jedes Blobs zu seinem Blob-Speicher und seinem Wissensnetz erfolgt über eine SQLite-Datenbank.

2.4.2.3.3 Komponenten

Wissensnetze bestehen aus Wissensnetzkomponenten. Neben den Basisfunktionalitäten gewähren sie den Wissensnetzen im Wesentlichen zusätzliche Schnittstellen und im Browser darstellbare Bedienoberflächen für die Nutzdaten (Webfrontends).

Eine besondere Untergruppe von Wissensnetzkomponenten sind die Veröffentlichungsstatuskomponenten (*Release States*), die in drei Varianten (*Preview*, *Release Candidate*, *Release*) existieren. Wird eine derartige Komponente im Wissensnetz installiert, können ausschließlich Software-Komponenten mit dem passenden Veröffentlichungsstatus auf das Wissensnetz zugreifen.



In der **Software-Liste** sind alle mit dem Admin-Tool mitgelieferten Wissensnetzkomponenten mit ihren jeweiligen Versionsnummern alphabetisch gelistet. Bedürfen sie einer eigenen Lizenz, ist außerdem vermerkt, ob die aktuelle Lizenz des Wissensnetzes sie umfasst oder nicht. Veröffentlichungsstatuskomponenten verfügen über keine Versionsnummer.

Wird eine Wissensnetzkomponente mit der rechten Maustaste angeklickt, erscheint ein Kontextmenü. Der dort verfügbare Menüpunkt **Standardkomponente hinzufügen** verfügt über die gleiche Funktionalität wie die gleichnamige Schaltfläche.

Standardkomponente hinzufügen installiert die in der **Software-Liste** ausgewählte Wissensnetzkomponente im Wissensnetz. Ein separates Fenster informiert über den Installationsstatus. Manche Wissensnetzkomponenten setzen für ihre Installation die Installation anderer Wissensnetzkomponenten im Wissensnetz voraus. Die meisten installierten Wissensnetzkomponenten (außer Veröffentlichungsstatuskomponenten) tauchen im Knowledge-Builder als eigene Einträge in der Rubrik *Technik* auf. Es kann immer nur eine Veröffentlichungsstatuskomponente zur gleichen Zeit installiert sein.

Lizenztemplate schreiben erzeugt eine inhaltlich zu vervollständigende Vorlage der für die Lizenzschlüsselgenerierung verwendeten Komponentenlizenzkonfigurationsdatei und speichert sie über einen Speicherdialog an einer frei wählbaren Stelle ab (Standarddateiname: *[Wissensnetz].componentLicenseTemplate.ini*). Unabhängig von der Konfiguration des gerade administrierten Wissensnetzes werden Konfigurationsplatzhalter für die Komponenten *KEM*, *i-views-Core* und *Knowledge-Builder* vorgegeben. In jedem Konfigurationsplatzhalter wird die im Admin-Tool mitgelieferte Versionsnummer der jeweiligen Wissensnetzkomponente voreingetragen.

In der **Wissensnetzliste** sind alle im Wissensnetz installierten Wissensnetzkomponenten mit

ihren jeweiligen Versionsnummern alphabetisch gelistet. Eine installierte Wissensnetzkomponente, für die im Admin-Tool eine neuere Version mitgeliefert ist, ist rot markiert. Die optionale Komponente *Knowledge-Builder* ist bei einem neuen Wissensnetz standardmäßig vorinstalliert.

Die Textfelder **Name** und **Version** zeigen den Namen respektive die dreistellige Versionsnummer der in der **Wissensnetzliste** ausgewählten installierten Wissensnetzkomponente.

Generische Komponente hinzufügen fügt der **Wissensnetzliste** eine generische Modellkomponente oder eine generische Softwarekomponente hinzu. Die Auswahl des Komponententyps erfolgt in einem separaten Fenster. Generische Komponenten erlauben die Bündelung projektspezifisch angefertigter Wissensnetzerweiterungen und vereinfachen deren (De-)Installation und Versionskontrolle über das Admin-Tool. Name und Versionsnummer einer im Wissensnetz installierten generischen Wissensnetzkomponente können in den entsprechend benannten Textfeldern frei vergeben werden.

Aktualisieren (die Bezeichnung wechselt zu **Erneuern**, wenn sie aktiviert werden kann) aktualisiert die in der **Wissensnetzliste** ausgewählte installierte Wissensnetzkomponente auf die im Admin-Tool mitgelieferte Version. Weicht die Sprache des aktuell laufenden Admin-Tools von der Sprache des Admin-Tools ab, mit dem die Wissensnetzkomponente ursprünglich im Wissensnetz installiert wurde, werden außerdem die Bezeichner aller Elemente und Elementtypen dieser Wissensnetzkomponente aktualisiert. Je nach Wissensnetzkomponente fügt die Aktualisierung den alten Bezeichnern neue Bezeichner in der Sprache des aktuell laufenden Admin-Tools hinzu (in Abhängigkeit von der Spracheinstellung des Knowledge-Builders wird dann die jeweils zutreffende Sprachversion dargestellt) oder ersetzt die alten Bezeichner mit den neuen Bezeichnern.

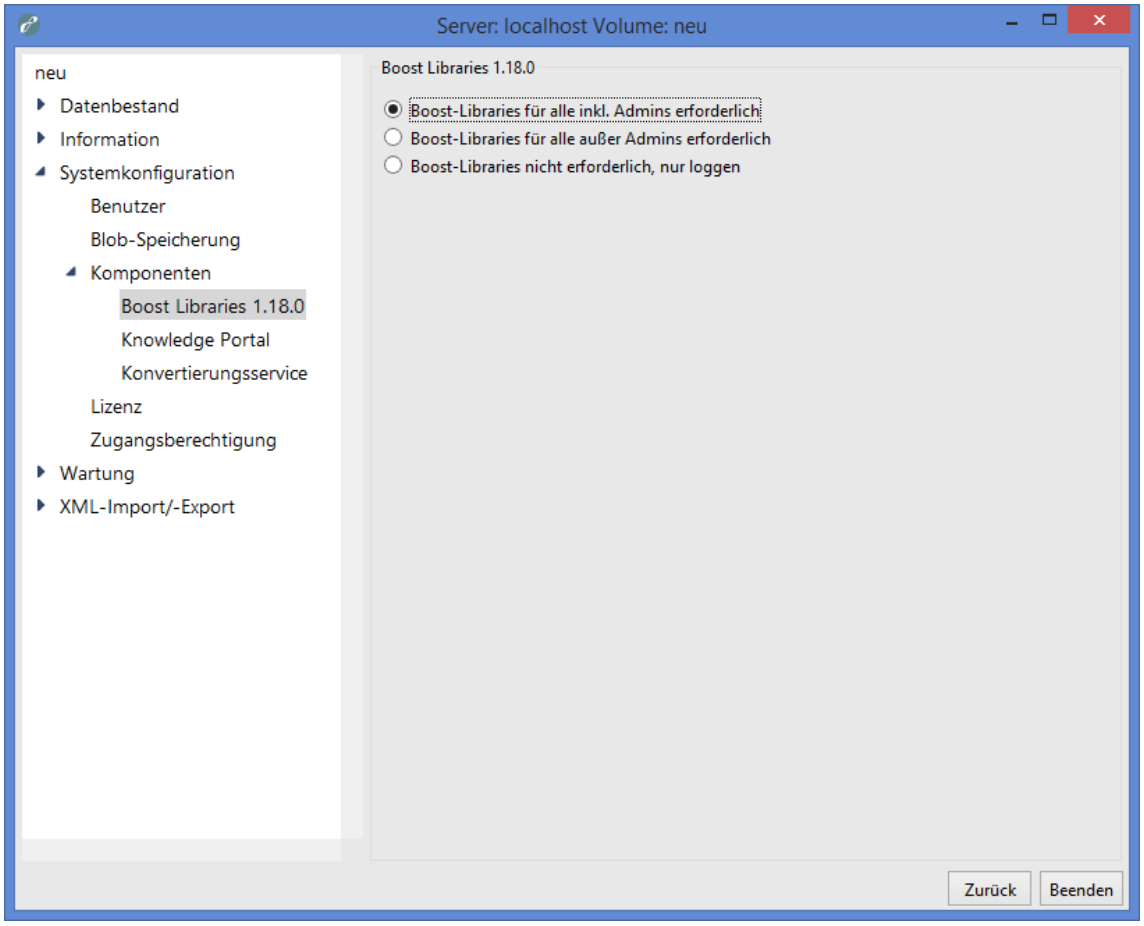
Entfernen deinstalliert die in der **Wissensnetzliste** ausgewählte installierte Wissensnetzkomponente. Sofern Wissensnetzkomponenten im installierten Zustand im Knowledge-Builder über einen eigenen Eintrag in der Rubrik *Technik* verfügen, hinterlassen sie dort nach ihrer Deinstallation ein eigenes Teilnetz, das manuell entfernt werden muss. Wissensnetzkomponenten lassen sich nur entfernen, wenn keine weiteren Wissensnetzkomponenten installiert sind, die von den zu deinstallierenden Wissensnetzkomponenten abhängig sind. Die beiden Wissensnetzkomponenten *i-views-Core* und *View-Konfiguration* bieten Basisfunktionalitäten und lassen sich nicht entfernen.

Boost Libraries 1.18.0

Dieses Konfigurationsmenü erscheint nur, wenn die Wissensnetzkomponente *Boost Libraries* installiert ist.

Mit Ausnahme des Blob-Services und des Mediators können alle Software-Komponenten JavaScript interpretieren. Um den Interpretationsumfang und die Interpretationsgeschwindigkeit von in JavaScript eingebetteten regulären Ausdrücken zu verbessern, kann deren Interpretation an die Bibliothek Boost.Regex übergeben werden. Unter Windows und Linux muss sich dazu die Bibliothek (Dateiname in Windows: *boost_regex.dll*, Dateiname in Linux: *libboost_regex.so*) im gleichen Verzeichnis befinden wie die übergebende Software-Komponente. In Mac OS ist die Bibliothek in die Datei der übergebenden Software-Komponente integriert.

Die Wissensnetzkomponente *Boost Libraries* ermöglicht die Sicherstellung, dass auf die Boost.Regex-Bibliothek zugegriffen werden kann.



Ist die Option **Boost-Libraries für alle inkl. Admins erforderlich** ausgewählt, können alle Software-Komponenten außer dem Admin-Tool auf das Wissensnetz nur zugreifen, wenn sie auf die Bibliothek Boost.Regex zugreifen können.

Ist die Option **Boost-Libraries für alle außer Admins erforderlich** ausgewählt, können alle Software- Komponenten außer dem Admin-Tool auf das Wissensnetz nur zugreifen, wenn sie auf die Bibliothek Boost.Regex zugreifen können. Ausgenommen von dieser Zugriffsperre sind Nutzer mit Administratorrechten, die das Wissensnetz über den Knowledge-Builder betreten.

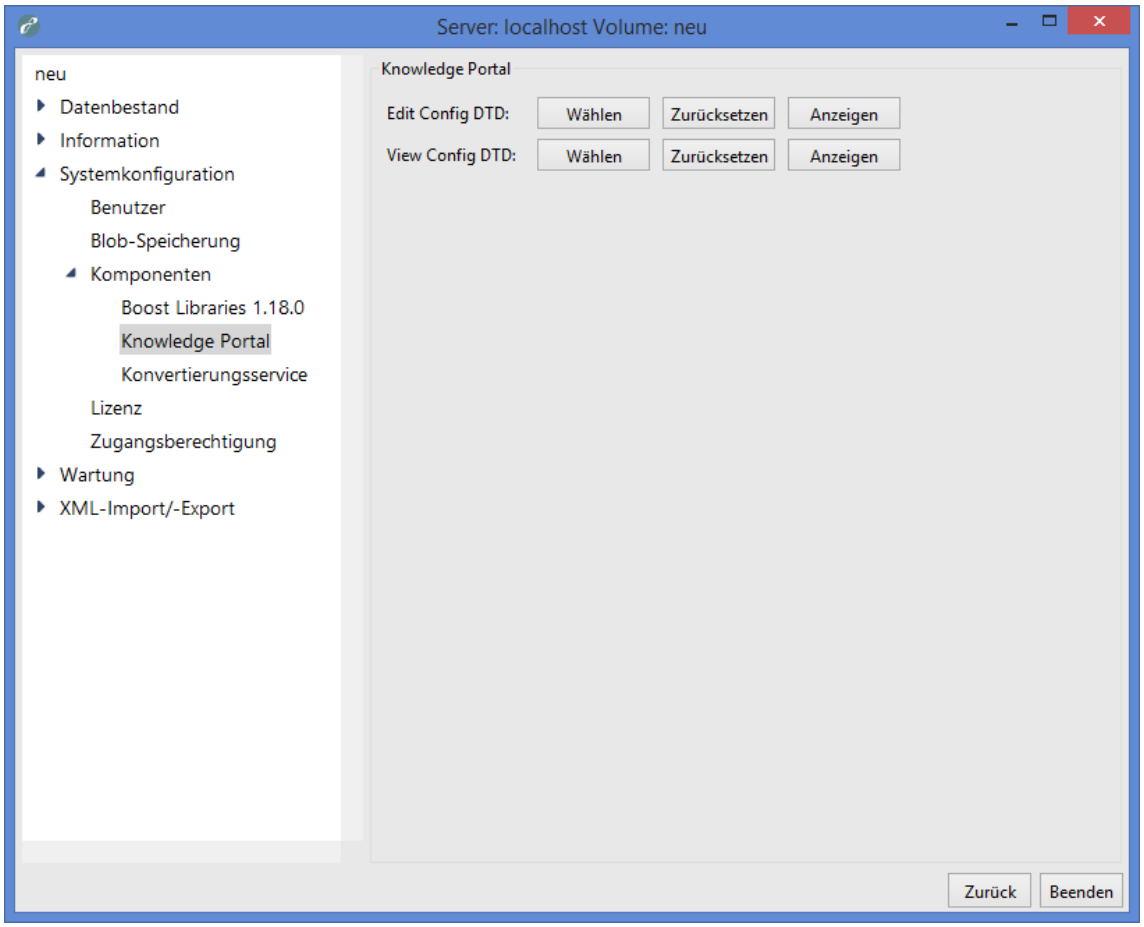
Ist die Option **Boost-Libraries nicht erforderlich, nur loggen** ausgewählt, hinterlegt jede Software- Komponente in seiner jeweiligen Protokolldatei, sofern vorhanden, eine entsprechende Warnung, wenn es beim Start nicht auf die Bibliothek Boost.Regex zugreifen kann. Ein Zugriff auf das Wissensnetz bleibt unabhängig davon möglich.

Knowledge Portal

Dieses Konfigurationsmenü erscheint nur, wenn die Wissensnetzkomponente *Knowledge-Portal* installiert ist.

Die Wissensnetzkomponente *Knowledge-Portal* ermöglicht einem Wissensnetz den Betrieb eines Knowledge-Portals (eines über einen Browser darstellbaren Frontends). Die Konfiguration der Darstellungs- und Bedienelemente dieses Frontends erfolgt im Knowledge-Builder an den entsprechenden Elementtypen über einen von der Wissensnetzkomponente speziell dafür bereitgestellten Editor mit Hilfe der Auszeichnungssprache XML. Zur einfacheren Wartung und logischen Reglementierung der XML-Dokumente lassen sich Schemata im Format DTD installieren, anhand derer die XML-Dokumente validiert werden können.

Im Frontend wird eine Bearbeitungssicht und eine Präsentationssicht mit jeweils exklusiven Darstellungs- und Bedienelementen unterschieden. Für beide Sichten werden separate DTD-Schemata geführt. Die nachfolgend erläuterten Bedienelemente existieren jeweils für jede Sicht.



Über die Schaltfläche **Wählen** kann auf das Dateisystem des Betriebssystems zugegriffen werden, um eine DTD-Schema-Datei für die jeweilige Sicht zu laden und im Wissensnetz zu installieren. Der Standarddateiname für Bearbeitungssicht-DTDs lautet *editConfig.dtd*, der Standarddateiname für Präsentationssicht-DTDs lautet *viewConfig.dtd*.

Zurücksetzen löscht das für die jeweilige Sicht installierte DTD-Schema aus dem Wissensnetz.

Anzeigen stellt das für die jeweilige Sicht installierte DTD-Schema in einem eigenen Fenster dar. Dort kann es in die Zwischenablage des Betriebssystems kopiert (Schaltfläche **In Zwischenablage kopieren**) oder über einen Speicherdialog als frei benennbare Textdatei an eine beliebige Stelle exportiert werden (Schaltfläche **Speichern**). Das Fenster verfügt außerdem über ein eigenes Kontextmenü, welches mit einem rechten Mausklick geöffnet werden kann:

- **Suche** erlaubt die Eingabe einer zu suchenden Zeichenkette in einem separaten Fenster und findet ihr nächstes Auftreten gemäß der Leserichtung relativ zur Position der per Mausklick setzbaren Schreibmarke. Bei der Suche wird Groß- und Kleinschreibung unterschieden.
- **Alles markieren** markiert den gesamten Text. Alternativ kann mit dem Mauszeiger ein beliebiger Textausschnitt markiert werden.
- **Kopieren** kopiert den gewählten Textbereich in die Zwischenablage des Betriebssys-

tems.

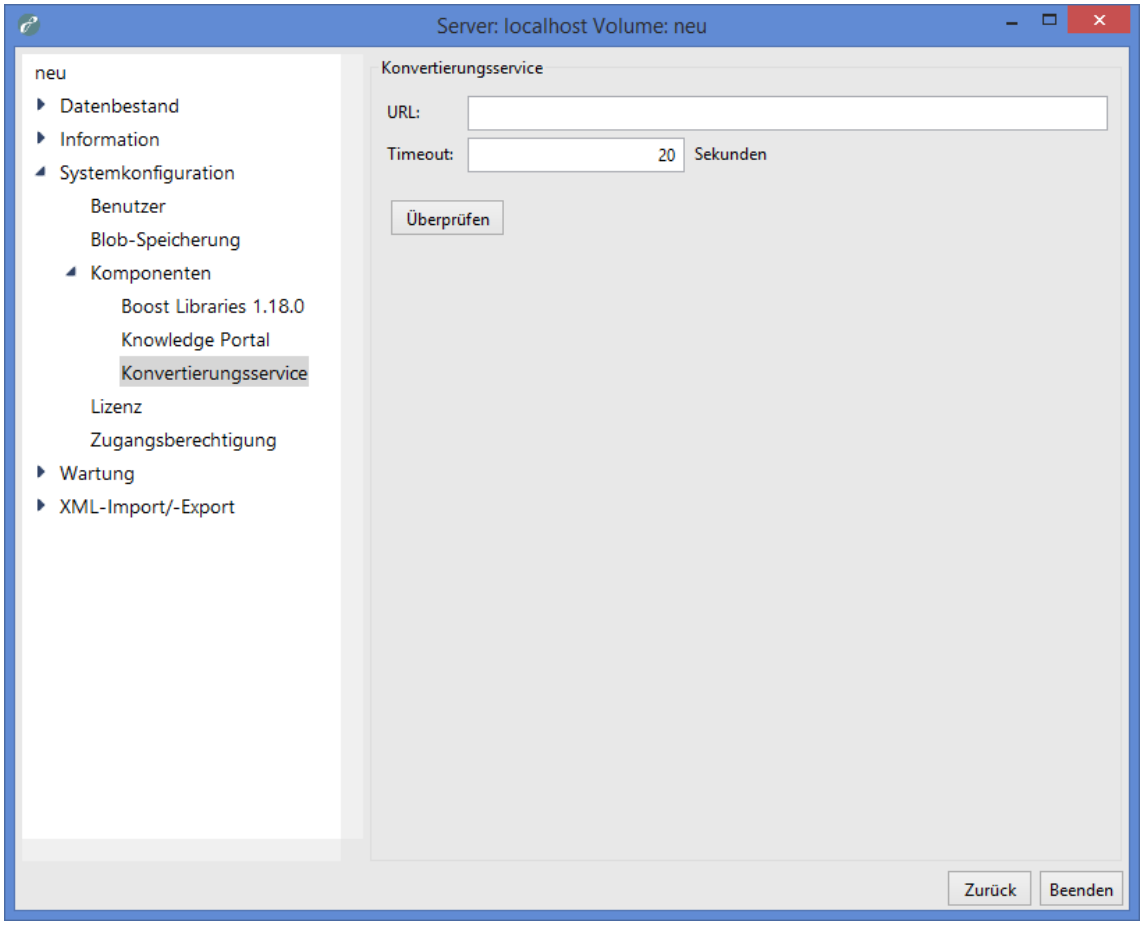
Konvertierungsservice

Dieses Konfigurationsmenü erscheint nur, wenn die Wissensnetzkomponente *Druckkomponente* installiert ist.

Die *Druckkomponente* ermöglicht die Integration ausgewählter Wissensnetzelemente in ein speicherbares elektronisches Dokument. Dazu muss eine Dokumentvorlage in den Formaten ODT, DOCX oder RTF über den Knowledge-Builder in das Wissensnetz importiert und mit den in ein Dokument zu integrierenden Wissensnetzelementen verknüpft werden. Die Gestaltung dieser Dokumentvorlage erfolgt in einem externen Office-Programm, von Elementen des Wissensnetzes auszufüllende Platzhalter lassen sich mit Hilfe von KScript und KPath definieren.

Zum Funktionsumfang der Druckkomponente gehört der Konvertierungsservice. Soll im Knowledge-Builder über den Kontextmenüpunt **Drucken** ein Dokument erzeugt werden, lassen sich neben dem Ursprungsformat der importierten Dokumentvorlage diverse andere Ausgabeformate wählen, in die die Dokumentvorlage konvertiert werden kann. Damit diese Konvertierung funktioniert, müssen eine passend konfigurierte Bridge (ein Software-Dienst) gestartet und mit der Druckkomponente verknüpft sowie eine Version von LibreOffice oder OpenOffice installiert sein.

Passend konfiguriert wird die Bridge über ihre Initialisierungsdatei (Standarddateiname: *bridge.ini*). Dort muss in der Sektion *[KHTTPRestBridge]* unter dem Schlüssel *services* der Wert *jodService* ergänzt werden. Außerdem ist eine neue Sektion *[file-format-conversion]* anzulegen und dort über das Schlüsselwertpaar *sofficePath="[Dateipfad]/soffice.exe"* mit einer korrekten Pfadangabe der Ort der Startdatei von LibreOffice beziehungsweise OpenOffice zu hinterlegen.



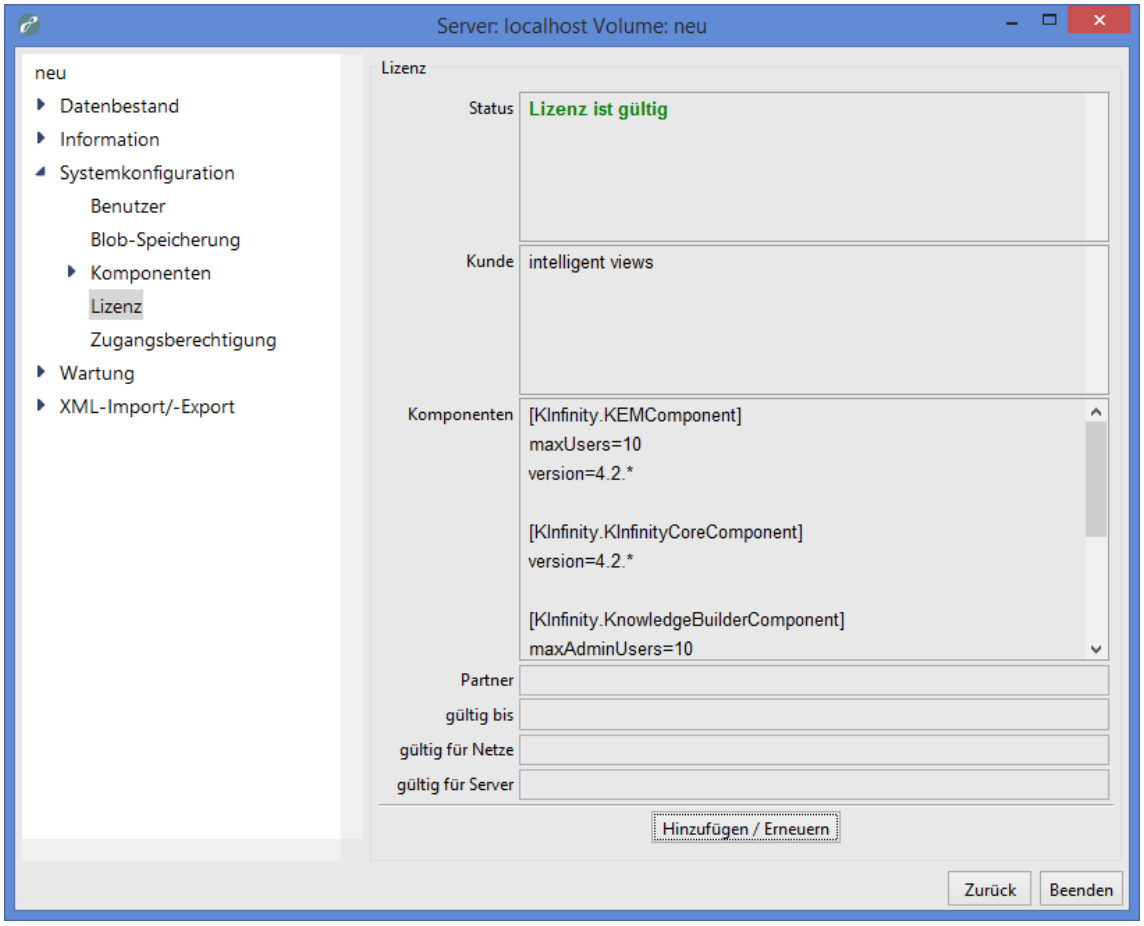
Die Verknüpfung der Bridge mit der Druckkomponente erfolgt über das Freitextfeld **URL**. Dort wird die Netzadresse der Bridge im Format *http:///[Bridge-IP-Nummer]:[Bridge-Port]/jodService/jodconverter/se* eingetragen. Der Pfadabschnitt */jodService/jodconverter/service* ist historisch bedingt und aktiviert den vordefinierten jodService.

Überprüfen startet einen Testprozess. Der Testprozess schickt über REST ein Testdokument an die über die Netzadresse festgelegte Bridge und erwartet, dass ein ordnungsgemäß konvertiertes Testdokument zurückgeschickt wird. Das Testergebnis wird in einem separaten Fenster ausgegeben.

Im Freitextfeld **Timeout** wird festgelegt, wie viele Sekunden lang auf die Rücksendung des konvertierten Testdokuments gewartet wird, bevor eine Fehlermeldung generiert wird. Die Voreinstellung liegt bei 20 Sekunden.

2.4.2.3.4 Lizenz

Ein Wissensnetz muss eine gültige Lizenz besitzen, damit der Knowledge-Builder und andere Software- Komponenten (mit Ausnahme des Admin-Tools) damit arbeiten können.



Das Feld **Status** gibt an, ob die Lizenz gegenwärtig gültig oder ungültig ist. Falls sie ungültig ist, wird außerdem ein Grund genannt. Gründe für eine ungültige Lizenz können die Überschreitung des Gültigkeitsdatums oder der maximalen Anzahl erlaubter registrierter Nutzer sein.

Das Feld **Kunde** beschreibt den Kunden, für den die Lizenz ausgestellt wurde. Neben dem Namen können auch die Adresse und die Abteilung genannt sein.

Das Feld **Komponenten** stellt den Inhalt der für die Lizenzschlüsselgenerierung verwendeten Komponentenlizenzkonfigurationsdatei *[Wissensnetz].componentLicenseTemplate.ini* dar. Dort werden Festlegungen über

- die lizensierten Versionen einzelner Komponenten (*version*),
- die maximale Anzahl registrierter Nutzer mit Administratorrechten (*maxAdminUsers*) und
- die maximale Anzahl registrierter Nutzer ohne Administratorrechte (*maxUsers*) getroffen.

Das Feld **Partner** enthält den Namen des Partners, über den die Lizenz weitergegeben wird.

Das Feld **gültig bis** enthält das Datum, nach dessen Ablauf die Lizenz erlischt.

Das Feld **gültig für Netze** enthält eine Liste der Namen aller Netze, auf die die Lizenz beschränkt ist. Möglich ist eine Erfassung über einen regulären Ausdruck.

Das Feld **gültig für Server** enthält eine Liste aller IP-Adressen und Port-Nummern, über die

ein an das Wissensnetz angeschlossener Mediator erreicht werden darf.

Die Felder **Partner**, **gültig bis**, **gültig für Netze** und **gültig für Server** können leer sein.

Alle Felder verfügen über ein Kontextmenü, das per Rechtsklick aktiviert werden kann:

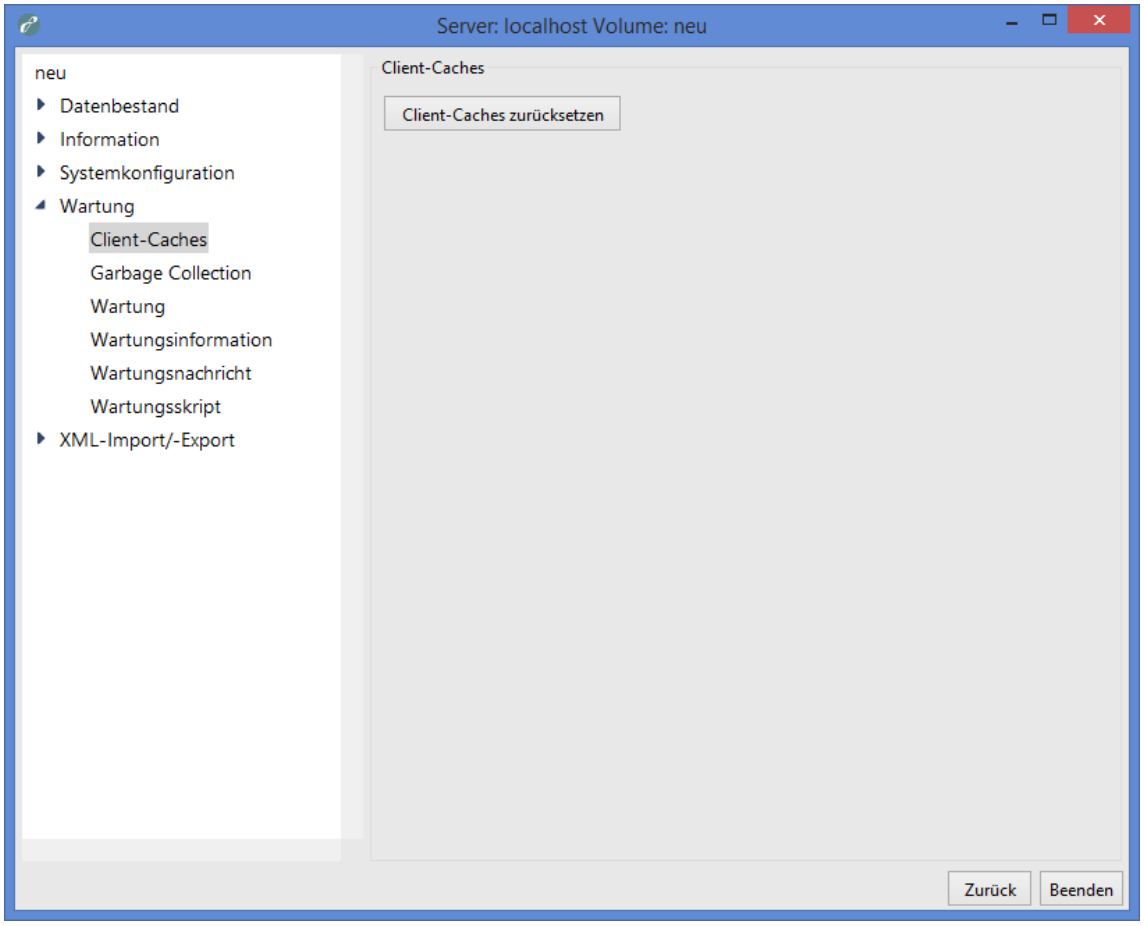
- **Select All** markiert den gesamten Text. Alternativ kann mit dem Mauszeiger ein beliebiger Textausschnitt markiert werden.
- **Copy** kopiert den gewählten Textbereich in die Zwischenablage des Betriebssystems.
- **Find Again** sucht nach dem gewählten Textbereich und findet sein nächstes Auftreten gemäß der Leserichtung.
- **Find** erlaubt die Eingabe einer zu suchenden Zeichenkette in einem separaten Fenster und findet ihr nächstes Auftreten gemäß der Leserichtung relativ zur Position der per Mausklick setzbaren Schreibmarke. Bei der Suche wird Groß- und Kleinschreibung unterschieden.

Hinzufügen / Erneuern erlaubt das Laden eines neuen Lizenzschlüssels (Dateiname: *[Lizenzname].key*) über das Dateisystem des Betriebssystems.

2.4.2.4 Wartung

2.4.2.4.1 Client-Caches

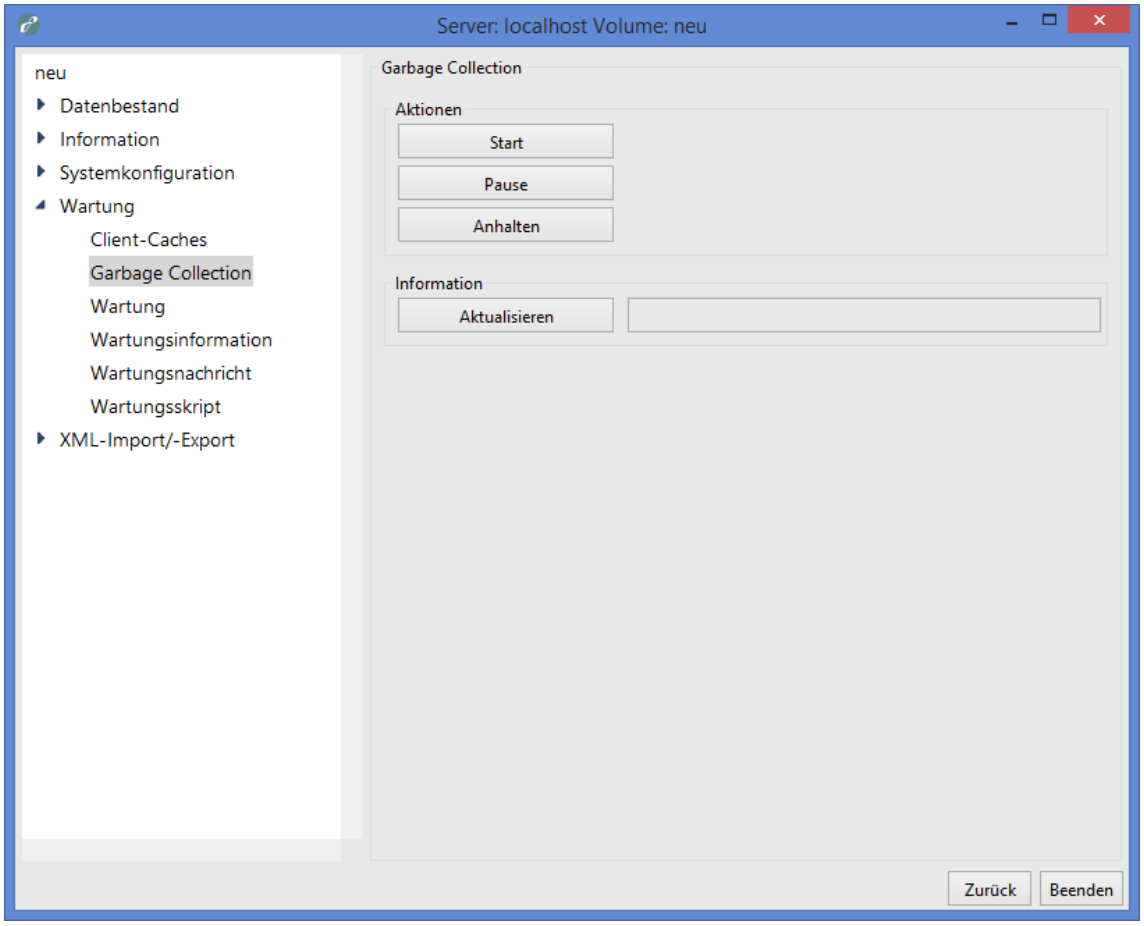
Zur Performanzverbesserung verwenden auf das Wissensnetz zugreifende Software-Komponenten häufig einen eigenen Pufferspeicher (Cache). Darin werden Schema- und Konfigurationsdaten des Wissensnetzes zwischengespeichert, um im Falle einer späteren Verwendung schneller auf sie zugreifen zu können.



Client-Caches zurücksetzen löscht diese zwischengespeicherten Daten. Dies ist sinnvoll, wenn sie aufgrund von Änderungen am Schema oder an der Konfiguration veraltet sind. Diese Operation setzt voraus, dass das Wissensnetz über einen Mediator angesteuert wird.

2.4.2.4.2 Garbage Collection

Die Garbage-Collection ist ein Verfahren, das nicht mehr referenzierte Objekte (nach programmierterminologischer Lesart) in einem Wissensnetz löscht und damit den Speicherverbrauch des Wissensnetzes minimiert. Die Nutzung der Garbage-Collection setzt voraus, dass das zu bereinigende Wissensnetz über einen Mediator angesteuert wird.



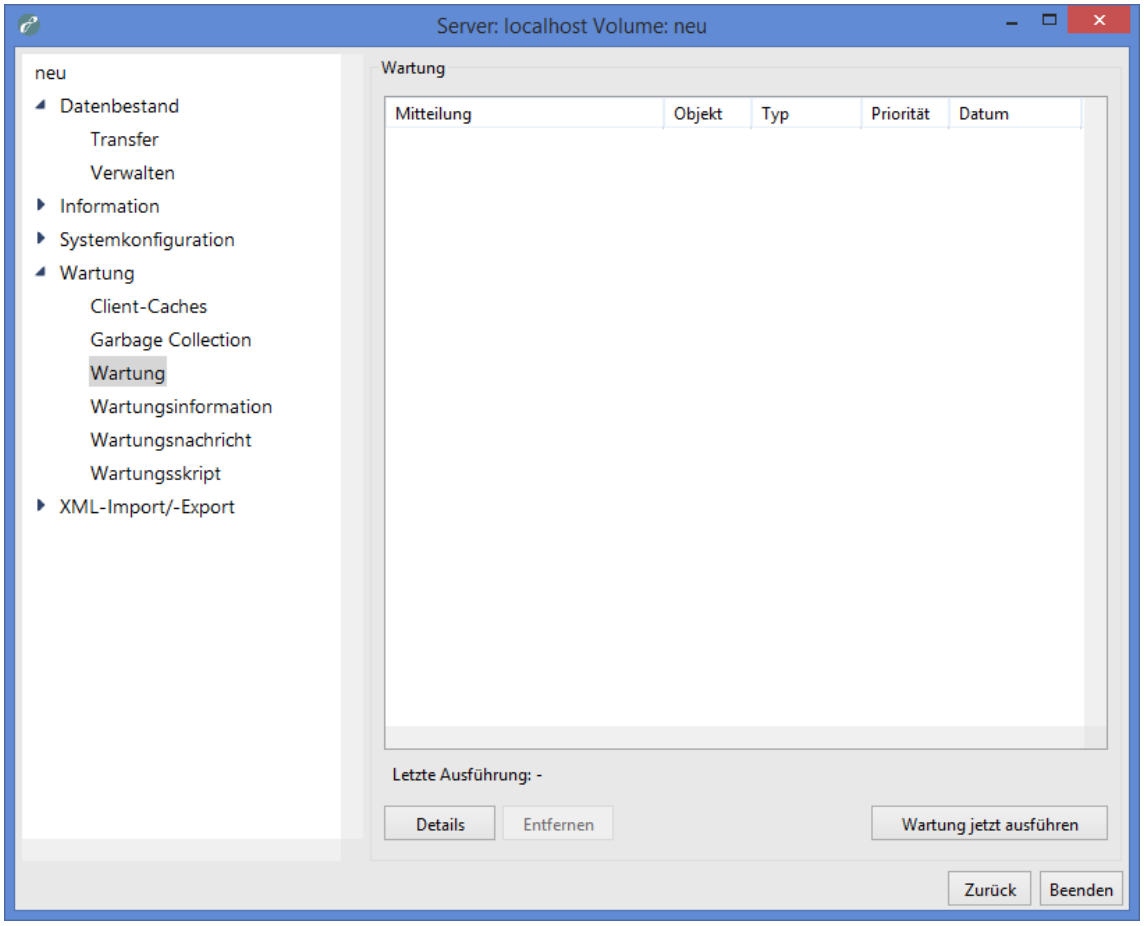
Start startet eine neue Speicherbereinigung für das Wissensnetz oder setzt eine pausierte Speicherbereinigung fort. Es erfolgt keine Rückmeldung, wann der Vorgang abgeschlossen ist. Der Stand des Fortschritts kann über den Menüpunkt **Aktualisieren** in Erfahrung gebracht werden.

Pause unterbricht die Durchführung der aktiven Speicherbereinigung für das Wissensnetz.

Anhalten bricht die Durchführung der aktiven Speicherbereinigung für das Wissensnetz ab.

Aktualisieren schreibt den aktuellen Zustand der Speicherbereinigung für das Wissensnetz in das nebenstehende Textfeld. Ist eine Speicherbereinigung aktiv, erfolgt zusätzlich eine Rückmeldung über den Stand des Fortschritts in Form einer Prozentangabe.

2.4.2.4.3 Wartung



Wartung jetzt ausführen überprüft

- die Lizenz (*Lizenz*),
- Indizes (*Indizes*),
- registrierte Objekte (*die Registratur*),
- Rechte (*Zugriffsrechte*),
- Trigger (*Trigger*) und
- installierte Wissensnetzkomponenten (*aktive Komponenten*)

auf Mängel. Im Zuge der Prüfung wird auch die über den Knowledge-Builder einsehbare Statistik von Eigenschaftshäufungen pro Objekt (Metriken) aktualisiert.

Gefundene Mängel werden in einer tabellarischen **Mangelübersicht** gesammelt. Für jeden Mangel wird dort

- eine kurze Beschreibung, falls einschlägig inklusive der Cluster-ID und der Frame-ID (Format *Cluster-ID/Frame-ID*) des mangelhaften Objekts (in programmierterminologischer Lesart) (*Mitteilung*),
- das vom Mangel betroffene übergeordnete Wissensnetzelement (*Objekt*),
- dessen Typ (*Typ*),

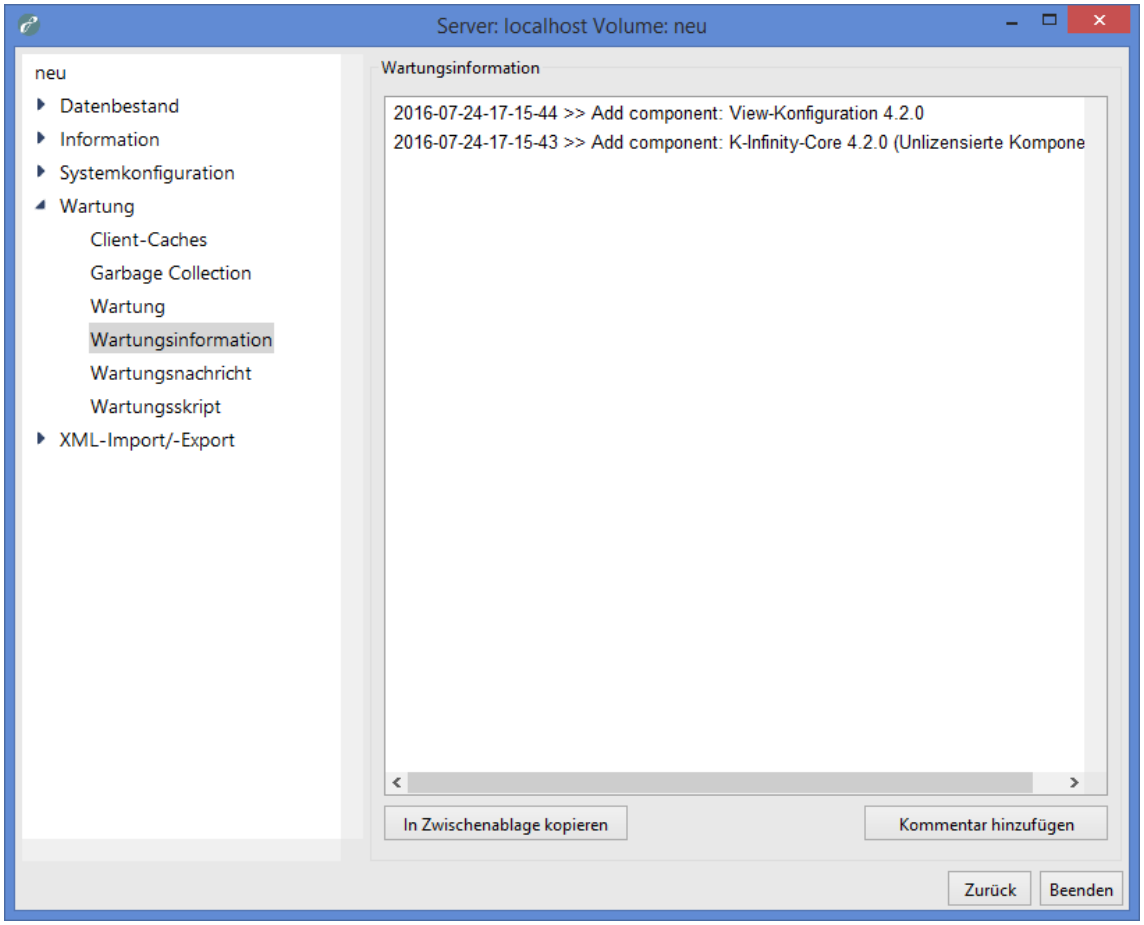
- die Schwere des Mangels (*Priorität*) und
- dessen ersten Feststellungszeitpunkt in Form eines Datums (*Datum*)

ausgegeben. Die einzelnen Spalten der Tabelle sind über einen Klick auf den Spaltenkopf sortierbar.

Details stellt alle in der **Mangelübersicht** des ausgewählten Mangels gelisteten Daten in einem neuen Fenster dar. Ergänzt werden die Uhrzeit des ersten Feststellungszeitpunkts sowie Datum und Uhrzeit des letzten Feststellungszeitpunkts. Die Daten können dort in die Zwischenablage des Betriebssystems kopiert (Schaltfläche **In Zwischenablage kopieren**) oder über einen Speicherdialog als frei benennbare Textdatei an eine beliebige Stelle exportiert werden (Schaltfläche **Speichern**). Die über die Schaltfläche **Details** ausgelöste Operation kann alternativ über einen Doppelklick auf einen Mangel in der Mangelübersicht erwirkt werden.

Entfernen löscht den in der **Mangelübersicht** ausgewählten Mangel. Dies hat keine Auswirkungen auf den ersten Feststellungszeitpunkt des Mangels.

2.4.2.4.4 Wartungsinformation



Über diesen Menüpunkt lässt sich eine chronologisch sortierte **Wartungshistorie** aller wesentlichen Administrationsvorgänge im Wissensnetz seit seiner Entstehung abrufen. Erfasst werden Backup- und Transfervorgänge, Komponenteninstallationen und -aktualisierungen sowie Ausführungen von Wartungsskripten und der Garbage-Collection, jeweils mit Datum und Uhrzeit.

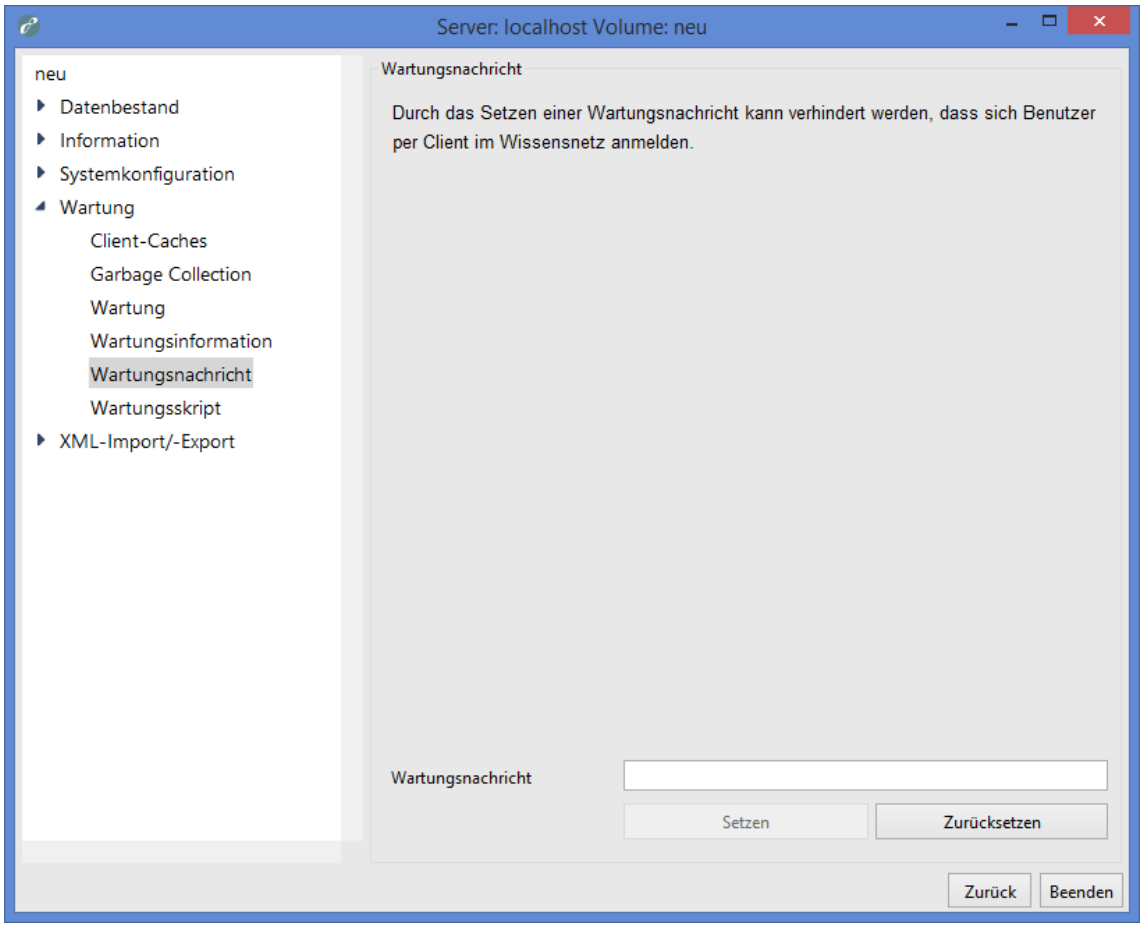
Die **Wartungshistorie** verfügt über ein Kontextmenü, das per Rechtsklick aktiviert werden kann:

- **Select All** markiert den gesamten Text. Alternativ kann mit dem Mauszeiger ein beliebiger Textausschnitt markiert werden.
- **Copy** kopiert den gewählten Textbereich in die Zwischenablage des Betriebssystems.
- **Find Again** sucht nach dem gewählten Textbereich und findet sein nächstes Auftreten gemäß der Leserichtung.
- **Find** erlaubt die Eingabe einer zu suchenden Zeichenkette in einem separaten Fenster und findet ihr nächstes Auftreten gemäß der Leserichtung relativ zur Position der per Mausklick setzbaren Schreibmarke. Bei der Suche wird Groß- und Kleinschreibung unterschieden.

In Zwischenablage kopieren kopiert die gesamte **Wartungshistorie** in die Zwischenablage des Betriebssystems.

Kommentar hinzufügen erlaubt die Eingabe einer Anmerkung über ein Freitextfeld in einem separaten Fenster. Sie wird mit einem Zeitstempel versehen und in die **Wartungshistorie** aufgenommen. In die **Wartungshistorie** aufgenommene Anmerkungen lassen sich nicht löschen.

2.4.2.4.5 Wartungsnachricht



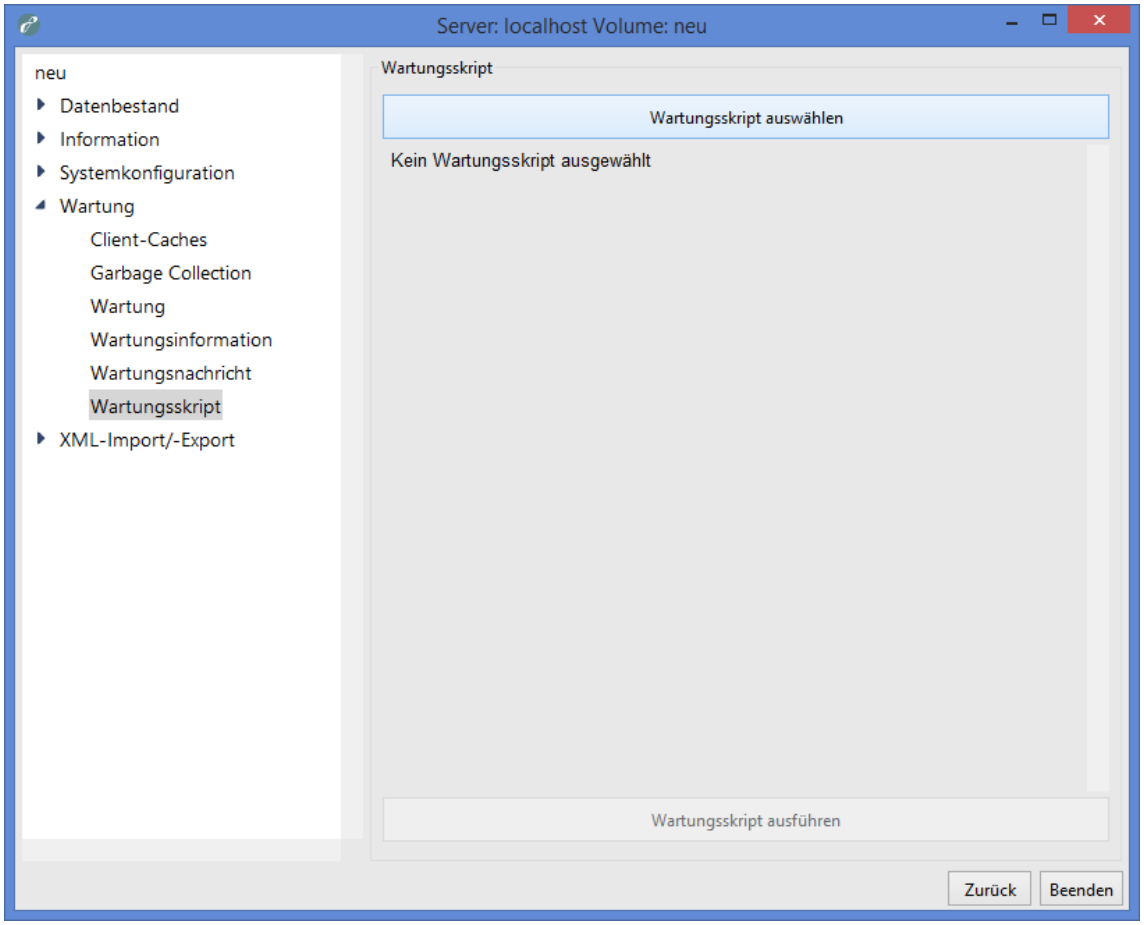
Die Schaltfläche **Setzen** aktiviert eine Wartungssperre, die jedem Nutzer den Zutritt zum

Wissensnetz über den Knowledge-Builder verwehrt. Dafür muss eine Wartungsnachricht formuliert werden.

Die Wartungsnachricht wird im Freitextfeld **Wartungsnachricht** formuliert. Sie wird jedem Nutzer, der das Wissensnetz bei aktivierter Wartungssperre über den Knowledge-Builder betreten will, in Form einer Fehlermeldung angezeigt.

Die Schaltfläche **Zurücksetzen** hebt die zuvor gesetzte Wartungssperre auf und löscht die Wartungsnachricht.

2.4.2.4.6 Wartungsskript



Über **Wartungsskript auswählen** kann auf das Dateisystem des Betriebssystems zugegriffen werden, um ein Wartungsskript (Dateiname: *[Wartungsskript].kss*) zu laden. Wartungsskripte werden fallspezifisch in der Programmiersprache Smalltalk angefertigt und erlauben Operationen, die sich nicht über die vordefinierten Funktionen des Admin-Tools oder über die KEM- oder JS-Schnittstellen realisieren lassen.

Verfügt das Wartungsskript über eine Beschreibung, wird diese nach dem Laden des Wartungsskripts in einem unsichtbaren Textfeld unterhalb der Schaltfläche **Wartungsskript auswählen** ausgegeben. Dieses Textfeld verfügt über ein Kontextmenü, das per Rechtsklick aktiviert werden kann:

- **Select All** markiert den gesamten Text. Alternativ kann mit dem Mauszeiger ein beliebiger Textausschnitt markiert werden.

- **Copy** kopiert den gewählten Textbereich in die Zwischenablage des Betriebssystems.
- **Find Again** sucht nach dem gewählten Textbereich und findet sein nächstes Auftreten gemäß der Leserichtung.
- **Find** erlaubt die Eingabe einer zu suchenden Zeichenkette in einem separaten Fenster und findet ihr nächstes Auftreten gemäß der Leserichtung relativ zur Position der per Mausklick setzbaren Schreibmarke. Bei der Suche wird Groß- und Kleinschreibung unterschieden.

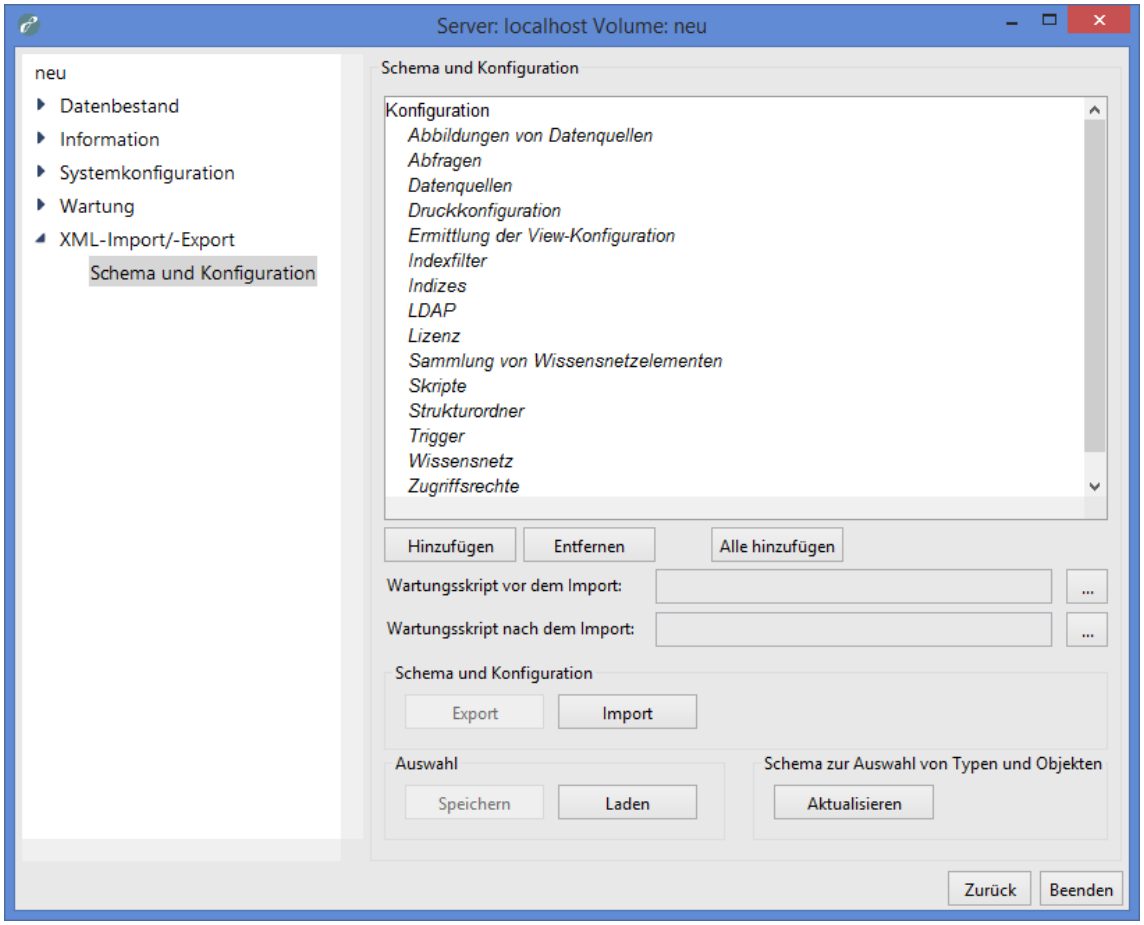
Wartungsskript ausführen startet das Wartungsskript. Ein separat erscheinendes Fenster gibt Auskunft, wenn das Wartungsskript vollzogen wurde, und bietet je nach Skript zusätzliche Ausführungsinformationen oder erlaubt skriptspezifische Ausführungsoptionen.

2.4.2.5 XML-Import/-Export

2.4.2.5.1 Schema und Konfiguration

Ein Wissensnetz im weiteren Sinne besteht neben den nutzergenerierten und über Komponenten eingebrachten Teilnetzen (Schemata mit Nutzdaten) noch aus diversen weiteren Bausteinen (Konfigurationen), die dieses Teilnetz funktional erweitern, konfigurieren oder damit arbeiten. Im Rahmen dieses Menüpunkts werden Schemata und Konfigurationen zusammenfassend als Konfigurationen bezeichnet.

Zahlreiche Konfigurationen eines Wissensnetzes lassen sich gezielt exportieren und importieren.



Die **Konfigurationsübersicht** bietet einen listenartigen Überblick über alle mittels der im Folgenden beschriebenen Operationen prinzipiell transferierbaren Konfigurationstypen eines Wissensnetzes. Prinzipiell transferierbar sind

- einzelne registrierte Abbildungen von Datenquellen (*Abbildungen von Datenquellen*),
- einzelne von Administratoren konfigurierte und benutzerdefinierte Suchfelder (*Abfragen*),
- einzelne Datenquellenzugriffseinstellungen zur Nutzung für Abbildungen von Datenquellen (*Datenquellen*),
- die Druckkonfiguration (*Druckkonfiguration*),
- die Menge aller innerhalb der Rubrik *Ermittlung* der View-Konfiguration definierten Bausteine (*Ermittlung der View-Konfiguration*),
- einzelne Indexfilter (*Indexfilter*),
- einzelne Indexerkonfigurationen (*Indizes*),
- die LDAP-Authentifizierung (*LDAP*),
- die Lizenz des Wissensnetzes (*Lizenz*),
- einzelne registrierte Sammlungen semantischer Objekte (*Sammlung von Wissensnetzelementen*),
- einzelne registrierte Skripte (*Skripte*),
- den Arbeitsordner (*Strukturordner*),
- die Menge aller innerhalb der Rubrik Trigger definierten Bausteine (*Trigger*),
- einzelne Teilnetze (*Wissensnetz*) und
- die Menge aller innerhalb der Rubrik Rechte definierten Bausteine (*Zugriffsrechte*).

Die **Konfigurationsübersicht** verwaltet überdies alle konkret zum Export bestimmten Konfigurationen. Zum Export bestimmte Konfigurationen erscheinen als ausklappbare Listenunterpunkte ihrer jeweiligen Konfigurationstypen. Benötigen diese Konfigurationen für ihren erfolgreichen Export andere Konfigurationen, sind diese anderen Konfigurationen wiederum in Form ausklappbarer Listenunterpunkte der jeweiligen Konfigurationen aufgeführt. Konfigurationstypen ohne eigene Konfigurationen sind kursiv gekennzeichnet, Konfigurationstypen mit eigenen Konfigurationen sind fett gekennzeichnet und stellen die Anzahl ihrer zugeordneten Konfigurationen in Klammern dar. Konfigurationstypen und Konfigurationen jedes Konfigurationstyps sind jeweils in alphabetischer Reihenfolge sortiert.

Das Ein- und Ausklappen von Listenunterpunkten in der **Konfigurationsübersicht** geschieht per Klick auf die sich links neben den Listenpunkten befindlichen Dreiecksymbole. Alternativ lässt sich dies über ein Kontextmenü realisieren, das über einen Klick mit der rechten Maustaste auf einen Listenpunkt aufgerufen werden kann:

- **Expand** klappt alle direkten Listenunterpunkte des gewählten Listenpunkts aus.
- **Expand fully** klappt alle direkten und alle indirekten Listenunterpunkte des gewählten Listenpunkts aus.
- **Contract fully** klappt alle Listenunterpunkte des gewählten Listenpunkts wieder ein.

Hinzufügen fügt der **Konfigurationsübersicht** eine Konfiguration des dort ausgewählten Konfigurationstyps hinzu. Existiert mehr als eine Konfiguration für den ausgewählten Konfigurationstyp im Wissensnetz, schließt sich eine Auswahlmöglichkeit in einem separaten Fenster an. Die Auswahl erfolgt dort entweder einzeln per Klick auf die jeweiligen Konfiguratio-

nen in einer Liste oder pauschal über die Schaltfläche **Alles aus-/abwählen**.

Entfernen löscht entweder alle Konfigurationen des in der **Konfigurationsübersicht** ausgewählten Konfigurationstyps oder die in der **Konfigurationsübersicht** ausgewählte Konfiguration.

Alle hinzufügen fügt der **Konfigurationsübersicht** alle im Wissensnetz existierenden Konfigurationen hinzu und verteilt diese auf die jeweils passenden Konfigurationstypen.

Über die Schaltflächen **...** kann auf das Dateisystem des Betriebssystems zugegriffen werden, um ein Wartungsskript (Dateiname: *[Wartungsskript].kss*) zu laden. Wartungsskripte werden fallspezifisch in der Programmiersprache Smalltalk angefertigt und erlauben Operationen, die sich nicht über die vordefinierten Funktionen des Admin-Tools oder über die KEM- oder JS-Schnittstellen realisieren lassen.

Wird ein Wartungsskript geladen, erscheint der Dateiname des gewählten Wartungsskripts im links von der jeweiligen Schaltfläche positionierten Textfeld. Werden im Anschluss Konfigurationen importiert, wird das Wartungsskript ausgeführt. Werden im Anschluss Konfigurationen exportiert, wird das Wartungsskript ebenfalls exportiert und erst beim Import dieser Konfigurationen ausgeführt. Der genaue Ausführungszeitpunkt des Wartungsskripts in Relation zum Importprozess hängt davon ab, über welche der beiden **...**-Schaltflächen es geladen wurde. Er befindet sich entweder vor dem Start des Importprozesses oder nach dem Ende des Importprozesses.

Export exportiert die in der **Konfigurationsübersicht** ausgewählten Konfigurationen. Zur Auswahl stehen der Export in eine einzige Archivdatei im Archivformat *tar* oder in einzelne Dateien in einem Ordner. Die Auswahl der Exportmethode vollzieht sich in einem separaten Fenster:

- In den Freitextfeldern **Datei** oder **Verzeichnis** kann der Name der Archivdatei (Dateiname: *[Wissensnetz].tar*) respektive des Ordners (kein Standardname) angegeben werden. Die Archivdatei respektive der Ordner wird im gleichen Ordner wie das Admin-Tool angelegt. Alternativ kann über **Wählen** ein Speicherdialog aufgerufen werden, um Name und Speicherort der Archivdatei respektive des Ordners frei festzulegen.

Import importiert nach einer Bestätigungsfrage Konfigurationen in das Wissensnetz. Zur Auswahl stehen der Import aus einer einzigen Archivdatei im Archivformat *tar* oder aus einzelnen Dateien in einem Ordner. Die Auswahl der Importmethode vollzieht sich in einem separaten Fenster:

- In den Freitextfeldern **Datei** oder **Verzeichnis** kann der Name der Archivdatei (Dateiname: *[Wissensnetz].tar*) respektive des Ordners (kein Standardname) angegeben werden. Die Archivdatei respektive der Ordner wird im gleichen Ordner wie das Admin-Tool gesucht. Alternativ kann über **Wählen** auf das Dateisystem des Betriebssystems zugegriffen werden, um eine Archivdatei respektive einen Ordner an einer beliebigen Stelle auszuwählen.
- Ist die zu importierende Archivdatei respektive der zu importierende Ordner gewählt, erscheint in einem weiteren Fenster eine Übersicht über alle darin enthaltenen Konfigurationen. Diese Übersicht kann dort in die Zwischenablage des Betriebssystems kopiert (Schaltfläche **In Zwischenablage kopieren**) oder über einen Speicherdialog als frei benennbare Textdatei an eine beliebige Stelle exportiert werden (Schaltfläche **Speichern**). Die Schaltfläche **Import** startet den Importprozess. Das Fenster verfügt außerdem über ein eigenes Kontextmenü, welches mit einem rechten Mausklick geöffnet werden kann:
  - **Suche** erlaubt die Eingabe einer zu suchenden Zeichenkette in einem separaten Fenster und findet ihr nächstes Auftreten gemäß der Leserichtung relativ zur Position der per Mausklick setzbaren Schreibmarke. Bei der Suche wird Groß- und Kleinschreibung unterschieden.

- **Alles markieren** markiert den gesamten Text. Alternativ kann mit dem Mauszeiger ein beliebiger Textausschnitt markiert werden.
- **Kopieren** kopiert den gewählten Textbereich in die Zwischenablage des Betriebssystems.

Speichern speichert die in der Konfigurationsübersicht für dieses Wissensnetz aktuell getroffene Auswahl an Konfigurationen als XML-Datei. Über einen Speicherdialog werden Name und Ort der XML-Datei (Standarddateiname: *instruction.xml*) festgelegt.

Laden greift auf das Dateisystem des Betriebssystems zu, um eine zuvor gespeicherte Auswahl an Konfigurationen für dieses Wissensnetz aus einer XML-Datei (Standarddateiname: *instruction.xml*) zu laden.

Aktualisieren fügt dem Wissensnetz die mit dem Attributdatentyp Boolesch ausgestatteten Attributtypen

- *XML: Alle Objekte exportieren*,
- *XML: Direkte Objekte exportieren*,
- *XML: Typ und alle Untertypen nicht exportieren* und
- *XML: Untertypen nicht exportieren*

hinzu, falls sie darin noch nicht existieren. Diese Attributtypen werden benötigt, um bei einem Export einer Konfiguration des Konfigurationstyps Wissensnetz auszuwählen, welche in dieser Konfiguration befindlichen Elemente und Elementtypen jeweils exportiert und nicht exportiert werden sollen. Dazu werden diese Attributtypen über den Knowledge-Builder an passende Objekttypen gehängt und mit passenden Attributwerten versehen.

Soweit nicht anders über diese Attributwerte konfiguriert, gilt für jeden Objekttyp, dass er selbst exportiert wird, nicht aber seine Objekte. Wird ein Objekt oder Objekttyp exportiert, werden alle direkt mit ihm verbundenen Attribute und Relationen sowie deren Attribut- respektive Relationstypen ebenfalls exportiert.

3 i-views-Dienste

3.1 Allgemeines

3.1.1 Kommandozeilen-Parameter

Falls es zu einem Aufrufparameter auch einen Eintrag der Ini-Datei gibt, hat der der Aufrufparameter höhere Priorität.

`-inifile <Dateiname>, -ini < Dateiname >`

Name der Ini-Datei, die statt dem Standard-Ini-Datei verwendet wird.

3.1.2 Konfigurationsdatei

Einige Einstellungen können über eine Konfigurationsdatei (*.ini) festgelegt werden. Der Aufbau der Datei sieht folgendermaßen aus:

```
[Default]
parameterName1=parameterWert1
parameterName2=parameterWert2
...
```

Im folgenden sind Konfigurationen aufgeführt, die für jeden Dienst verwendet werden können. Für dienstspezifische Einstellungen siehe den Abschitt "Konfigurationsdatei" des entsprechenden Dienstes.

Logging-Einstellungen

`loglevel = <LogLevel>`

Konfiguriert, welche Meldungen im Log erscheinen sollen:

- FATAL ERROR: Nur kritische Fehlermeldungen
- ERROR: Nur Fehlermeldungen
- WARNING: Nur Warnungen und Fehlermeldungen
- NORMAL (Standardwert): Alle Meldungen außer Debug-Ausgaben
- NOTIFY: Alle Meldungen inklusive einiger Debug-Ausgaben
- DEBUG: Alle Meldungen inklusive aller Debug-Ausgaben

`debug = true/false`

Veraltet. Setzt den Log-Level bei true auf DEBUG, bei false auf NORMAL. Wird nur noch ausgewertet, wenn logLevel nicht gesetzt wird

`nolog = true/false`

Veraltet. Entspricht bei true einem logtargets=null. Wird nur noch ausgewertet, wenn logtargets nicht gesetzt wird

`channels = <Channel1> [,<Channel2>,...]`

Namen von Channelfiltern. Mit Hilfe der Channelfilter werden nur Log-Meldungen ausgegeben, die zu den angegebenen Channelfiltern gehören. Der Name eines Channelfilters deutet darauf hin, zu welchem Themengebiet die Log-Ausgaben gehören. Welche Channelfilter möglich sind, erfährt man in der Kommandozeile mit Hilfe des Parameters -availableChannels.

`channelLevels = <Channel1>:<Level1> [,<Channel2>:<Level2>,...]`

Gezielte Konfiguration des Loglevels für den jeweiligen Channel.

`logtargets = <Name1> [,<Name2>,...]`

Namen von Log-Targets. Für die Konfiguration siehe Abschnitt „Log-Targets“.

`logprefix = <Prefix1> [, <Prefix2>,... ]`

Zusätzliche Daten, die bei jeder Log-Ausgabe hinzugefügt werden:

- pid : Prozess-ID der Anwendung
- $proc$: ID des aktuellen Smalltalk-Threads
- $alloc$: belegter Speicher der VM (in Megabyte)
- $free$: Freier Speicher der VM (in Megabyte)
- $incGC$: Status inkrementelle GCs
- os : Information über OS
- cmd : Kommandozeile
- $build$: Build-Version
- $coast$: COAST-Version

Bei einem Präfix, der nicht in dieser Liste enthalten ist, wird der Präfix unverändert ausgegeben.

`logTimestampFormat = <FormatString>`

Formatierungsangabe für den Timestamp des Log-Eintrags, z.B. "hh:mm:ss".

`exceptionLogSize = <Integer>`

Setzt die maximale Größe des bei einer Fehlermeldung mitgelieferten StackTrace.

Log-Targets

Über Log-Targets lassen sich verschiedene Ziele für das Logging festlegen, für die sich jeweils Log-Level, Channels, Formatierung und mehr konfigurieren lassen. Für jeden angegebenen Namen aus der logtargets-Liste muss eine Konfiguration im Abschnitt [<Konfigurationsname>] angegeben werden:

```
logtargets=default

[default]
type=stderr
format=json
loglevel= ERROR
```

konfiguriert zum Beispiel eine Ausgabe aller Fehlermeldungen im JSON-Format auf dem Standard-Error-Stream.

Eine Ausnahme stellt das Log-Target null da: bei eine konfiguration von logtargets=null muss kein Konfigurationsabschnitt erstellt werden. Fehlt dieser, so ist dies gleichbedeutend mit folgender Konfiguration

```
logtargets=null

[null]
type=null
```

Es ist jedoch möglich, null als bezeichner für eine beliebige Log-Target Konfiguration zu verwenden.

Grundsätzlich lassen sich genau wie in der allgemeinen Konfiguration loglevel, debug, channels, channelLevels, logprefix und logTimestampFormat festlegen (siehe oben). Die Konfiguration am Log-Target hat immer Vorrang, wenn keine angegeben ist wird jedoch auf die

allgemeine Konfiguration zurückgegriffen.

Zusätzlich gibt es noch einige weitere Konfigurationsmöglichkeiten:

`format = <Format>`

legt das Ausgabeformat fest. Mögliche Werte sind:

- **plain**: Die Standardformatierung in möglichst menschenlesbarer Form
- **json**: einzeilige Ausgabe als JSON-String, vorallem für Maschinenverarbeitung

`type = <Zieltyp>`

legt den Typ der Ausgabe fest. Diese Konfiguration MUSS angegeben werden, sonst wird das Log-Target ignoriert. Im folgenden sind Beschreibung und weitere Konfigurationsmöglichkeit der verschiedenen Typen angegeben:

file

Ausgabe in eine Log-Datei.

`file = <Dateiname>`

legt den Dateinamen der Ziel-Datei fest.

`maxLogSize = <size>`

Die maximale Größe des Logfiles, ab der die alte Logdatei archiviert wird und eine neue angebrochen wird. Bei Werten kleiner als 1024 wird die Angabe als in MB verstanden.

stdout

Ausgabe auf den Standard-Out-Stream.

stderr

Ausgabe auf den Standard-Error-Stream.

mail

Versendet die Log-Ausgabe per Mail.

```
[errorMail]
type = mail
loglevel =  ERROR    ;Nur Fehler ausgeben.
attachLog = true    ;Bei true wird zusätzlich eine Log-Datei als Anhang beigefügt.
sender = mail@example.org    ;Absender-Adresse
recipient = rec@example.org    ;Empfänger-Adresse
smtpHost = stmp.example.org    ;Mail-Server
smptPort = 465    ;Port des Mailservers. Standardmäßig 25 oder 465.
tls = true    ;Aktiviert bei true die gesicherte Verbindung (TLS/SSL).
              ;Bei true muss username und password gesetzt sein.
username = mail@example.org    ;Nutzername zur Authentifizierung (TLS)
password = 12345abc    ;Passwort zur Authentifizierung (TLS)
retries = 3    ;Anzahl der Versuche, die Mail bei einem Fehlschlag erneut zu senden.
retryDelay = 5    ;Wartezeit zwischen zwei Versuchen in Sekunden
```

mailfile

Wie mail, allerdings werden Ausgaben mit niedrigem Log-Level zunächst angesammelt und erst per Mail versendet, wenn ein Eintrag mit hohem Level geloggt wird.

`mailSendLevel = <LogLevel>`

setzt das Log-Level, ab dem die Mail gesendet wird.

syslog

Ausgabe als UDP-Datagramm an einen Syslog-Client.

`format = <Format>`

Anders als bei anderen Log-Targets werden json und plain als Formatierung nicht unterstützt, stattdessen kann hier die Syslog-Version angegeben werden:

- **rfc5424**: Formatiert die Nachricht nach RFC 5424. Die meisten Daten werden strukturiert im Structured-Data-Feld hinterlegt. Nur die eigentliche Log-Message wird im Message-Feld übertragen.
- **rfc3164**: Formatiert die Nachricht nach RFC 3164. Da dieser Standard kein Structured-Data-Feld hat, werden die entsprechenden Daten in der gleichen Formatierung an den Anfang des Message-Felds gestellt. Achtung: Der Zeitstempel ist standardkonform in Lokalzeit des sendenden Rechners angegeben.

`facility = <Integer>`

Die Facility als Ganzzahl. Für detailierte Informationen siehe https://tools.ietf.org/html/rfc5424#section-6.2.1

`targetHostname = <Hostname>`

Der Hostname des Zielsystems. Falls nicht angegeben wird localhost verwendet.

`targetPort = <Integer>`

Der Port, an den gesendet werden soll. Falls nicht angegeben, wird der Syslog-Standard-Port 514 verwendet.

`hostname = <Hostname>`

Der Hostname des Senders. Falls nicht angegeben, wird des Hostname des Systems ausgelesen.

`appname = <Name>`

Name der sendenden Anwendung. Falls nicht angegeben, wird der Name der EXE verwendet.

`maxMessageSize = <Integer>`

Die maximale Nachrichtengröße in Bytes. Falls nicht angegeben, wird die maximale Größe für UDP verwendet. Zum Kürzen der Nachricht wird zunächst stückweise Structured Data entfernt und im Notfall die Message abgeschnitten. Die Nachricht ist auch nach der Kürzung in validem Syslog-Format.

null

Zum Unterdrücken der Logausgaben. Es werden keinerlei Optionen ausgelesen.

3.2 Mediator

3.2.1 Allgemeines

Der i-views-Server sorgt für konsistente und persistente Datenhaltung und für die Aktualität der Daten auf den angeschlossenen i-views-Clients.

Die Datenhaltung erfolgt in einer objektorientierten Datenbank, die durch ein optimistisches Transaktionssystem kooperatives Arbeiten auf dem Wissensnetz ermöglicht.

In seiner Funktion als Kommunikationszentrale sorgt der i-views-Server für die Synchronisation von Clients und Services. Als Basismechanismus stellt er hierfür einen geteilten Objektraum und aktive Updates zur Verfügung.

Technische Daten:

- Multi-Platform Executable auf Basis der VisualWorks Smalltalk Virtual Machine (mediator.exe bzw. mediator.im).
- Konfigurierbarer TCP/IP Server-Port für die Kommunikation mit den Clients, Standard bei i-views 5.0 ist 30062.

3.2.2 Systemvoraussetzungen

Der i-views-Server ist Plattform-unabhängig und läuft auf allen gängigen Betriebssystemen, z. B. Windows, Solaris und Linux (ab Kernel 2.2).

OS	Version	Prozessor	Unterstützt	64 Bit VM
Windows	10, 8, 7,Vista, 2000, XP, Server 2003, 2008, 2012	x86	ja	ja
	2003	IA64	x86 Emulation	nein
AIX	5.3		ja	nein
HP-UX	11.x	PA-RISC	ja	nein
	11.x	IA64	nein	nein
Solaris	8+	Sparc	ja	ja
	10	x86	nein	nein
Linux	RedHat, SLES9+, u.a.	x86	ja	ja

	RedHat, SLES9+, u.a.	IA64	x86 Emulation	nein
	RedHat, SLES9+, u.a.	PPC	ja	nein
Mac	OSX 10.4+	x86	ja	nein
	OSX 10.4+	PPC	ja	nein

3.2.3 Installation

Der i-views-Server benötigt prinzipiell keine spezielle Installation, d.h. er ist ad hoc aus einem beliebigen Verzeichnis startbar.

Es ist dabei darauf zu achten, dass die notwendigen Zugriffsrechte (lesen/schreiben/erzeugen) für das Arbeitsverzeichnis des Servers und alle Unterverzeichnisse gesetzt sind.

3.2.3.1 Startparameter

Dem Mediator Prozess können beim Start diverse Parameter mit übergeben werden. Die meisten Parameter können aber auch in der mediator.ini angegeben werden, sodass man den Mediator mit einer einfachen Kommandozeile starten kann. Dabei gilt, dass auf der Kommandozeile angegebene Parameter Präzedenz vor evtl. doppelt in der .ini-Datei angegebenen Parametern haben.

Die komplette Liste der möglichen Startparameter gibt der Mediator beim Aufruf mit dem Parameter "-?" aus.

`-interface <interface-1>`

Dieser Parameter bestimmt, unter welchen Adressen und Protokollen der Server erreichbar ist. Mögliche Protokolle sind: http, https, cnp, cnps. Dabei steht "cnp" für "Coast Native Protocol" bzw. "Coast Native Protocol Secure". Der syntaktische Aufbau einer Interface-Definition entspricht der einer URL mit Schema, Host und Port. Über den Host-Teil steuert man, über welche Netzwerkadresse(n) der Server erreicht werden kann. z.B: "0.0.0.0"=IPv4 alle Interfaces, "[::1]"=IPv6 nur Loopback.

Die Protokolle "http" und "https" lassen sich über Proxies umleiten, so dass man den Server bspw. über einen auf Port 443 laufenden IIS erreichen kann.

`-clientTimeout <sec>`

Setzt die Zeit, innerhalb der sich ein Client automatisch melden muss, auf <sec> Sekunden. Der Wert sollte mindestens auf 600 gesetzt werden (was auch der Standardwert ist).

`-baseDirectory <directory>`

Setzt das Verzeichnis, in dem sich das Verzeichnis "volumes" befindet. Neben dem Unterverzeichnis "volumes" werden hier auch standardmäßig die Verzeichnisse für Backups und Downloads angelegt. Dieser Parameter hieß früher "-volumes".

Die folgenden Parameter stellen Kommandos an das Mediator Executable, um bestimmte Aufgaben auszuführen, ohne danach als Server für Wissensnetze zu fungieren.

`-quickRecover <volume> -recover <volume>`

Falls der Mediator unordnungsgemäß beendet wird (z.B. Absturz des Rechners), bleiben in Volumes, die in Benutzung waren, Lock-Dateien stehen. Das Volume kann dann nicht mehr betreten werden. Um das Lock aufzuheben, kann man mit dem Aufruf von -quickRecover <volume> das Lock entfernen. Der Aufruf schlägt fehl, wenn (mögliche) Inkonsistenzen gefunden wurden. In diesem Fall muss der Startparameter -recover verwendet werden.

Achtung:

Das Arbeitsverzeichnis beim Aufruf muss hier das Verzeichnis sein, welches das „volumes"-Verzeichnis enthält. Der „-volumes"-Parameter wirkt hier also nicht.

`-bfscommand <volume> <command>`

Führt Kommandos aus, die vom BlockFileSystem erkannt werden.

Kommandozeilen-Parameter für das Logging:

`-nolog`

Schaltet Logging ab

`-loglevel <Integer>`

Konfiguriert, welche Meldungen im Log erscheinen sollen:

- 0: Alle Meldungen inklusive Debug-Ausgaben
- 10 (Standardwert): Alle Meldungen außer Debug-Ausgaben
- 20: Nur Warnungen und Fehlermeldungen
- 30: Nur Fehlermeldungen

`-logfile <Dateiname>, -log <Dateiname>`

Name der Log-Datei, die statt der Standard-Log-Datei verwendet wird. Dieser Parameter muss auf jeden Fall verändert werden, wenn mehrere Clients im selben Arbeitsverzeichnis gestartet werden sind.

`-debug`

Schaltet das Logging auf debug-mode

`-log <logname>`

Setzt die Logdatei auf <logname>.

3.2.3.2 Konfigurationsdatei "mediator.ini"

Einige Mediator-Einstellungen können auch in der Konfigurationsdatei mediator.ini festgelegt werden. Der Aufbau der Datei sieht folgendermaßen aus:

```
[Default]
parameterName1=parameterWert1
parameterName2=parameterWert2
```

...

Folgende Parameter sind an dieser Stelle einsetzbar:

`port=<portnummer>`

Startet den Server mit der Portnummer <num>. Ohne diese Angabe wird Port 30061 verwendet.

Dieser Parameter ist veraltet. Er wird durch den Parameter "interfaces" ersetzt. Dabei entspricht ein Eintrag von "port=1234" dem Eintrag "interfaces=cnp://0.0.0.0:1234". Im Unterschied zum Startparameter sind hier mehrere Werte zulässig, die, durch Komma getrennt, hintereinander aufgezählt werden.

`interfaces=<interface-1>,<interface-2>,...<interface-n>`

Dieser Parameter bestimmt, unter welchen Adressen und Protokollen der Server erreichbar ist. Mehrere Werte sind zulässig, und werden durch Komma getrennt. Mögliche Protokolle sind: http, https, cnp, cnps. Dabei steht "cnp" für "Coast Native Protocol" bzw. "Coast Native Protocol Secure". Der syntaktische Aufbau einer Interface-Definition entspricht der einer URL mit Schema, Host und Port. Über den Host-Teil steuert man, über welche Netzwerkadresse(n) der Server erreicht werden kann. z.B: "0.0.0.0"=IPv4 alle Interfaces, "[::1]"=IPv6 nur Loopback.

Die Protokolle "http" und "https" lassen sich über Proxies umleiten, so dass man den Server bspw. über einen auf Port 443 laufenden IIS erreichen kann.

`baseDirectory=<Verzeichnis>`

Setzt das Verzeichnis, in dem sich das Verzeichnis volumes befindet. Sollte dieser Wert auf volumes enden, so wird dieses Verzeichnis direkt verwendet, ohne noch zusätzlich darunter ein Verzeichnis volumes anzulegen.

`volumesDirectory=<Verzeichnis>`

In diesem Verzeichnis liegen die Wissensnetze. Als Standardwert ist an dieser Stelle ’volumes’ eingetragen.

`backupDirectory=<Verzeichnis>`

Gibt das Verzeichnis an, in welches Wissensnetz-Backups geschrieben und zum Wiederherstellen auch gelesen werden. Nur vollständige Verzeichnisnamen erlaubt, keine Relativ-Angabe.

`networkBufferSize=<Größe in Bytes>`

Gibt die Größe des Puffers an, der für das Senden/Empfangen von Daten verwendet wird. Der Standardwert ist 20480. In manchen Infrastrukturen kann man durch Angabe von

networkBufferSize=4096

einen höheren Durchsatz erreichen.

`flushJournalThreshold=<Anzahl der Cluster>`

gibt den Maximalwert an, den "veränderte Cluster" + "Indexcluster" in einem Speichervorgang erreichen dürfen. Wenn der Wert für "veränderte Cluster" bereits überschritten ist, werden keine "Indexcluster" gespeichert sondern diese werden mit Journal geführt.

Ein niedriger Wert (z.B. 50) garantiert schnelle Speicherzeiten, baut aber potentiell ein großes Journal auf.

Ein Wert von "0" deaktiviert Journaling. Standardwert ist "2000".

Anmerkung: Ein "flush" des Journals wird immer spätestens bei einer vollständigen Speicherung ausgeführt. Diese wiederum wird ausgelöst, wenn:

- der Mediator beendet wird
- der letzte Client des entsprechenden Volumes abgemeldet wird
- eine Speicherung durch einen full-save-job (siehe jobs.ini) ausgelöst wird

`autoSaveTimeInterval=<Warteintervall in Sekunden>`

gibt an in Sekunden, wie lang nach dem letzten Cluster-Speichern maximal gewartet wird, bis wieder automatisch gespeichert wird. Standardwert ist 15.

`clientTimeout=<Timeout in Sekunden>`

gibt die Zeit in Sekunden an, die ein verbundener Client maximal keine Alive-Nachricht geschickt haben darf, bevor der Mediator ihn als inaktiv erachtet und ihn ausschließt.

```
password.flavour=19013329307152292800186471980559137636 1
password.hash=11199545182458660705495599802052624171734965791427080638694954247035513239844
```

Das Mediator-Passwort wird zusammen mit einem zufälligem flavour zu einen (SHA256) hashwert berechnet. Diese beiden Informationen genügen dann, um dem Mediator eine Authentifizierungsanfrage zu überprüfen. Beim Authentifizieren am Server muss der Benutzername mit "Server:admin" angegeben werden. Um diese Werte zu ermitteln kann man mit

`password.update=neues_passwort`

den Server veranlassen, beim Start einen neuen Flavour und einen passenden Hashwert zu berechnen und in die ini-Datei zurückzuschreiben. Der Eintrag password.update wird dabei entfernt.

`password=<String>`

Die veraltete, noch unterstützte Art, das Mediator-Passwort zu setzen. Diese Variante darf nicht gleichzeitig mit der SHA256-Hash Variante verwendet werden.

Geändert

`skipVolumesCheck=<true|false>`

gibt an, ob die normalerweise nach dem Start des Mediators durchgeführte Überprüfung der vorhandenen Volumes ausgelassen wird

Geändert

Logging-Einstellungen:

Zu den Konfigurationsmöglichkeiten des Loggings siehe Logging-Einstellungen im Kapitel 11.1.2 Konfigurationsdatei.

Speicher-Einstellungen:

Die folgenden drei Parameter dienen zur Konfiguration der Speicherzuteilung und -nutzung. Erlaubt ist die Angabe von Werten entweder in Megabyte oder in tatsächlichen Byte, wobei die Annahme gilt, dass sich Werte kleiner als 1048576 auf Megabyte-Angaben beziehen.

`maxMemory=<Integer, in MB>`

Maximal erlaubte Hauptspeicherbelegung. Minimal 50 MB, standardmäßig gesamter physikalisch vorhandener Hauptspeicher (unter Windows) bzw. 512 MB.

`growthRegimeUpperBound=<Integer, in MB>`

Hauptspeicherbelegung ab der verstärkt versucht wird, Speicher freizugeben. Standardmäßig 0.6 * maxMemory.

`freeMemoryBound=<Integer, in MB> [10]`

Falls belegter, aber nicht mehr benötigter Speicher diese Grenze überschreitet, wird er wieder freigegeben.

BLOB-Service-Konfiguration

Soll der Mediator mit einem integrierten BLOB-Service gestartet werden, damit die BLOBs getrennt von der Datenbank auf der Festplatte gespeichert werden, so muss die folgende Einstellung in der "mediator.ini" eingetragen werden:

```
startBlobService=true
```

Nähere Informationen dazu finden Sie in der Dokumentation des BLOB-Service (siehe Link unten).

3.2.3.3 Sicherheitskonzept des Mediators

Der i-views-Server ist eine generische Komponente, die nicht nur für i-views verwendet werden kann. Neben der Einschränkungen über die Authentifizierungen am Server oder in der Datenbank kann man auch kontrollieren, welche Anwendungen sich verbinden dürfen.

Jede Anwendung (Client und Server) enthält ein RSA-Schlüsselpaar, das je ausgelieferter Anwendung eindeutig ist. Den öffentlichen Schlüssel kann man über die Information erhalten (KB: Menü „Werkzeuge“, „Info“, dann die Schaltfläche „RSA-Key kopieren“) bzw. für Konsolen-Anwendungen per Aufruf mit dem Parameter -showBuildID. Die hierdurch exportierte Build-Information enthält den öffentlichen RSA-Exponenten (rsa.e_1) und RSA-Modul (aufgeteilt auf mehrere rsa.n_x) sowie eine MD5 Prüfsumme dieser Informationen (buildID).

Beispiel einer Build-Information:

```
[buildID.90A1203EFB957A58C2268AD8FE3CC5A3] rsa.n_1=93D516DF61395258AA21A91B33E8EE67 rsa.n_2=B07C6
```

Möchte man nun, dass sich nur eine bestimmte Menge Client-Anwendungen mit dem Server verbinden kann, so muss man im Server die jeweiligen Abschnitte in die mediator.ini übertragen. Beim Verbindungsaufbau überträgt der Client seine buildID. Wenn der Mediator einen passenden Eintrag enthält, so wird er die Client-Authentizität prüfen. Andernfalls wird er eine Verbindung nur aufbauen, wenn es gar keine Einträge zu Build-Informationen in seiner Ini-Datei gibt. Somit kann beispielsweise verhindert werden, dass sich veraltete Client-Anwendungen oder modifizierte Client-Anwendungen mit dem Mediator verbinden.

Umgekehrt können auch in der Client-Anwendung entsprechende buildIDs für die Mediatoren in die jeweilige ini-Datei eingetragen werden, um eine Verbindung zu einem kompromittierten oder veralteten Server zu verhindern.

So kann man eine Umgebung einrichten, in der nur mit der aktuellsten Software auf die Produktivdaten zugegriffen werden kann, aber auf die Server mit den Testdaten auch von einer Entwicklungsumgebung aus. Die Anwendersoftware wiederum kann nur auf den Produktivserver oder auf den Testserver zugreifen.

Konfiguriert man weder Server noch Client, so verhält sich die Installation wie in den Vorgängerver-

sionen: Jede Anwendung kann sich mit jedem Server verbinden (sofern die Protokollversion übereinstimmt).

Seit der Version 5.4 des Servers benötigt man zum Durchführen administrativer Befehle das Server-Passwort als Parameter (über die Rest-Schnittstelle oder über die Verwaltung per Administrationswerkzeug). Für Aktionen, die sich auf eine existierende Datenbank beziehen (backup, download, garbage collection usw) genügt hierfür seit Version 6.2 eine Authentifizierung als Administrator im Volume.

Umgekehrt ist es mit dem Serverpasswort möglich, sich in einem Volume anzumelden. Details hierzu finden sich im Admin-Tool.

Ist am Server kein Passwort konfiguriert, so kann man sich mit einem beliebigen Passwort am Server anmelden. Die Anmeldung im Volume ist dann jedoch nicht möglich.

3.2.3.4 Audit-Log konfigurieren

In einigen Anwendungsszenarien kann es gefordert sein, alle Zugriffe auf ein Wissensnetz in einem Zugriffs- oder Audit-Log zu protokollieren. Dieses Audit-Log enthält Einträge für alle An- und Abmeldevorgänge, schreibende und lesende Zugriffe auf Wissensnetz-Inhalte, gestellte Suchanfragen, Ausdrucke, Exporte etc.

Damit das Zugriffslog aktiviert werden kann, muss eine Konfigurationsdatei 'log.ini' im Verzeichnis des zu überwachenden Wissensnetzes angelegt werden:

```
[Default]
applicationLog=CoastJSONApplicationLogger

[CoastJSONApplicationLogger]
backupInterval=14
maxLogSize=5
```

Zusätzlich muss das Log im Admin-Tool in der Rubrik 'Systemkonfiguration / Audit-Log' aktiviert werden. Die Aktivierung bzw. Deaktivierung des Logs resultiert wiederum in einem Eintrag im Audit-Log.

Für die Auswertung der Zugriffslogs steht Administratoren eine Funktion im Administrations-Menue des Knowledge-Builders zur Verfügung.

3.2.4 Betrieb

3.2.4.1 Herunterfahren des Servers

Der i-views-Server lässt sich lokal durch das Strg-C Abbruchsignal herunterfahren.

Bei der Installation als Windows-Dienst muss der Server mit der Diensteverwaltung gestoppt werden.

Unter UNIX sowie beim Betrieb als Windows-Dienst, wird der Server beim Herunterfahren des Betriebssystems ordnungsgemäß beendet.

3.2.4.2 Speicherung und Backup von Wissensnetzen

Verzeichnisstruktur

Das Basisverzeichnis des i-views-Servers weist folgende Struktur auf:

```
volumes/
   wissensnetzName/
     wissensnetzName.cbf
     wissensnetzName.cdr
     wissensnetzName.cfl
     wissensnetzName.lock (wenn das Wissensnetz geöffnet ist)

backup/
  wissensnetzName/
    <zehnstellige Nummer>/
      wissensnetzName.cbf
      wissensnetzName.cdr
      wissensnetzName.cfl
```

Speicherung von Wissensnetzen

Wissensnetze werden im Dateisystem im Unterverzeichnis "volumes" des Basisverzeichnisses des i-views-Servers abgelegt. In diesem Verzeichnis wird für jedes Wissensnetz ein Unterverzeichnis mit entsprechendem Namen angelegt. Eine Datei mit der Dateiendung '.lock' zeigt an, dass ein Wissensnetz gerade in Verwendung ist.

Backup von Wissensnetzen

Die Wissensnetzverzeichnisse dürfen auf keinen Fall direkt kopiert werden, so lange der i-views-Server läuft. Zu diesem Zweck besitzt der Server einen Backup-Service, der einen konsistenten Stand des Wissensnetzes in einen Backup-Bereich kopiert. Dieser Backup-Bereich muss in regelmäßigen Abständen gesichert werden (z.B. im Rahmen einer allgemeinen Backup-Strategie).

Der Ort, an dem die Backups angelegt werden kann über den Eintrag

```
backupDirectory=<Verzeichnis>
```

in der Datei "mediator.ini" festgelegt werden. Ohne diese Angabe wird das Unterverzeichnis "backup" des Basisverzeichnisses verwendet.

Der Backup-Service des i-views-Servers kann auf zwei Arten angestoßen werden:

1. Durch einen direkten Request an den Serverprozess (z.B. vom i-views-Administrationstool aus)
2. Durch Einträge in der Datei 'jobs.ini' im Arbeitsverzeichnis des i-views-Servers. Diese Datei kann pro Wissensnetz eine Rubrik mit folgenden Einträgen enthalten:

```
;Uhrzeit, zu dem das Backup gestartet wird backupTime=00:45 ;Turnus in Tagen - hier täglich backup
```

Der Eintrag 'backupInterval' ist optional.

3.2.4.3 Garbage Collection

Ohne Garbage Collection wächst das Wissensnetz kontinuierlich mit der der Verwendung. Folglich ist es sinnvoll, von Zeit zu Zeit eine Bereinigung (Garbage Collection) durchzuführen. Wie die Datensicherung kann die Garbage Collection jederzeit manuell (z.B. mit einem

speziellen Administrationswerkzeug) oder automatisch gestartet werden.

Die Garbage Collection kann - je nach Netzgröße - viel Zeit und Arbeitsspeicher in Anspruch nehmen. Bei der Durchführung auf großen Netzen empfiehlt es die Garbage Collection ohne verbundene Clients (z.B. KBuilder und JobClients) und ohne weitere aktive Prozesse (z.B. Backup) zu starten.

Automatische Garbage Collection: Aufbau der Datei jobs.ini

Die automatische Garbage Collection wird durch einen Eintrag in der Datei 'jobs.ini' konfiguriert, z.B.

```
[volume1] garbageCollectTime=00:55 garbageCollectInterval=7
```

Dieser Eintrag in der jobs.ini sorgt dafür, dass das Netz mit Namen "volume1" im Abstand von "7" Tagen jeweils um "00:55" Uhr garbage-collected wird. Für das Intervall ist der Standardwert "1" (also täglich), der Zeitpunkt muss angegeben werden.

Bei der Angabe der Netznamen in eckigen Klammern ist die Verwendung der Platzhalter "*" und "?" erlaubt, Groß- und Kleinschrift wird ignoriert.

Manueller Start der Garbage Collection

Alternativ kann die Garbage Collection auch durch spezielle Aufrufparameter des i-views-Servers gesteuert werden:

-startGC <volume> -host <hostname>	Startet die Garbage Collection auf dem Netz mit Namen <volume> auf einen ggfs. entfernten Mediator auf Rechner <hostname> (optional inkl. Portangabe).
-stopGC <volume> -host <hostname>	beendet eine ggfs. auf dem Mediator <hostname> laufende Garbage-Collection des Netzes mit dem Namen <volume>.
-infoGC <volume> -host <hostname>	Informiert über den aktuellen Stand der Garbage Collection.

Diese Kommandos werden mit Hilfe eines Mediator-Executables an einen anderen bereits laufenden Mediator übermittelt.

Als weitere Möglichkeit bietet sich das Starten der Garbage Collection über das Admin-Tool an.

Zum Ausführen dieser Befehle muss mittels Parameter -password das richtige Serverpasswort übermittelt werden.

3.2.4.4 Betrieb unter Unix

Unter UNIX reagiert der Server auf folgende Signale:

`SIGTERM/SIGHUP`

Beendet den Server

`SIGUSR2`

Der Server startet einen sofortigen Backup aller Wissensnetze, die in der jobs.ini-Datei für Backup spezifiziert sind (siehe auch Abschnitt über Backup).

3.2.4.5 Betrieb im Cluster

Der Mediator kann in einem Cluster betrieben werden. In einer Cluster-Umgebung wird i.d.R. eine laufende Spiegelung der Verzeichnisse und damit des Wissensnetzes vorgenommen. Fällt der Teil des Clusters, auf dem der Mediator läuft aus, wird automatisch ein neuer Mediator gestartet, der dann den Zugriff auf das Wissensnetz verwaltet.

Bei Ausfall des ersten Mediators kann es passieren, dass der Mediator keine Zeit mehr hat, das Wissensnetz in einen konsistenten Zustand zu bringen, das Netz damit eine Inkonsistenz aufweist und die "lock"-Datei des alten Mediators noch im entsprechenden Verzeichnis bestehen bleibt. Damit der neue Mediator in der Lage ist, die "lock"-Datei zu löschen, muss folgender Parameter in die mediator.ini hinzugefügt werden.

```
host=NameDesClusters
```

In diesem Fall können alle Mediatoren mit diesem ini-Eintrag auch gesperrte Volumes anderer Mediatoren, die beim Start den selben Wert in der mediator.ini ausgelesen hatten, entsperren. "NameDesClusters" ist frei wählbar, muss aber den Regeln entsprechen, die für Hostnamen gelten (keine Leerzeichen, Doppelpunkte, o.ä.)

Eine Konsistenzprüfung des Volumes läuft beim Starten des Mediators automatisch ab. Soweit möglich, wird das Wissensnetz in einen konsistenten Zustand versetzt und der Betrieb läuft normal weiter.

3.2.4.6 Problembehebung

Falls der i-views-Server während des Betriebs nicht ordnungsgemäß heruntergefahren wurde (z.B. Absturz des Rechners), bleiben bei geöffneten Wissensnetzen die Sperren bestehen. Beim Öffnen eines gesperrten Wissensnetzes wird diese Sperre erkannt und - falls möglich - entfernt.

Falls der Mediator eine Inkonsistenz erkennt, kann in der Kommandozeile durch den Aufruf des Mediators mit den Parametern -quickRecover / -recover das Wissensnetz geprüft und Inkonsistenzen soweit möglich repariert werden.

Sollte eine Auflösung der Inkonsistenzen wider Erwarten nicht möglich sein, muss auf eine Sicherungskopie zurückgegriffen werden.

3.2.4.7 Kommandos des BlockFileSystems

Die Befehle hinter -bfscommand ermögliche Oerationen auf dem BlockFileSystem und sind für Supportfälle vorgesehen. Ein solcher Befehl könnte zBsp so aussehen:

`-bfscommand quickCheck {target volume}`

Die mit {raget volume} addressierte Datenbank wird einer schnellen Strukturanalyse unterzogen. Analog kann mit deepCheck eine komplettanalyse ausgeführt werden.

3.3 Bridge

3.3.1 Allgemeines

Die Bridge ermöglicht den externen Zugang zu Wissensnetzen auf drei Arten/Betriebsmodi:

- Über eine RESTful Services-Architektur (REST-Bridge). Die Schnittstelle steht als HTTP- oder HTTPS-Version zur Verfügung.
- Über http/XML (KMultiBridge): Alte Zugriffsart. Wird noch vom Net-Navigator benötigt.
- Über KEM-RPC (KEMBridge): Zugang über KEM. Falls Binärdaten im Wissensnetz gespeichert werden ist zusätzlich eine Streaming-Bridge (KEMStreamingBridge) erforderlich.

Zusätzlich gibt es noch den Betriebsmodus "Lastverteiler für andere Bridges" (KLoadBalancer).

ACHTUNG: KMultiBridge und KEMBridge dürfen nicht gleichzeitig in einer Bridge aktiviert werden, weil diese sich gegenseitig behindern. Gleiches gilt für den Betrieb als KLoadBalancer, auch dieser sollte allein betrieben werden (siehe Kapitel KLoadBalancer).

Die Bridge und alle in ihr zu aktivierenden Zugänge lassen sich über eine ini-Datei konfigurieren. Einstellungen für die Zugänge sind dabei in Abschnitten gebündelt. Die wichtigsten dieser Parameter lassen sich aber auch über die Kommandozeile spezifizieren. Ist dies der Fall, so haben die Werte des Kommandozeilenaufrufs Vorrang vor denen in der ini-Datei. Die einzelnen Parameter werden nun erläutert.

3.3.2 Gemeinsame Kommandozeilen-Parameter

Wird die Bridge ohne jegliche Parameter gestartet, so werden die erforderlichen Parameter aus der Ini-Datei bridge.ini gelesen und die Fehlermeldungen in die Datei bridge.log geschrieben.

Falls es zu einem Aufrufparameter auch einen Eintrag der Ini-Datei gibt, hat der der Aufrufparameter höhere Priorität.

`-inifile <Dateiname>, -ini < Dateiname >`

Name der Ini-Datei, die statt dem Standard-Ini-Datei verwendet wird. Standard ist bridge.ini

`-host <hostname:port>, -hostname <hostname:port>`

Name des Mediators, der als Datenserver fungiert. Dieser gilt für alle aktivierten Bridgeclients

`-port |<ClientName> <portnumber>`

Der Parameter -port ist eigentlich für jeden Klienten in der ini-Datei zu setzen. Will man dieses aber bereits in der Kommandozeile tun, so lassen sich die unterschiedlichen Klienten durch

Voranstellen des Klientennamens vor die Portnummer spezifizieren. Die obige Zeile gilt für einen Klienten, entsprechend muss der Parameter -port wiederholt werden, sollen mehrere Klienten konfiguriert werden.

Beispiele für den Aufruf der Bridge:

```
bridge -host server01:30000 -port KEMBridge 4713 -port KEMStreamingBridge 4714

bridge -ini bridge2.ini -port KMultiBridge 3030
```

Kommandozeilen-Parameter für das Logging:

`-nolog`

Schaltet Logging ab

`-loglevel <Integer>`

Konfiguriert, welche Meldungen im Log erscheinen sollen:

- 0: Alle Meldungen inklusive Debug-Ausgaben
- 10 (Standardwert): Alle Meldungen außer Debug-Ausgaben
- 20: Nur Warnungen und Fehlermeldungen
- 30: Nur Fehlermeldungen

`-logfile <Dateiname>, -log <Dateiname>`

Name der Log-Datei, die statt der Standard-Log-Datei verwendet wird. Dieser Parameter muss auf jeden Fall verändert werden, wenn mehrere Clients im selben Arbeitsverzeichnis gestartet werden sind.

`-debug`

Schaltet das Logging auf debug-mode

`-log <logname>`

Setzt die Logdatei auf <logname>.

`-stop <hostname>`

Ruft man die Bridge mit dem obigen Parameter auf, so wird die auf dem angegebenen Host laufende Bridge zum Beenden aufgefordert. Alle in ihr gestarteten Klienten werden heruntergefahren und die Bridge beendet.

3.3.3 Konfigurationsdatei "bridge.ini"

Alle der folgenden Einträge befinden sich unterhalb des ini-Datei-Abschnitts [Default]. Die Einträge für die einzelnen Klienten schließen daran an. Durch das Einfügen klientenspezifischer Konfigurationsabschnitte wird zusätzlich definiert, welche Klienten in der zu konfigurierenden und zu startenden Bridge aktiviert sind. Im Moment mögliche Klienten sind dabei:

- KEMBridge
- KEMStreamingBridge

- KMultiBridge

Zusätzlich kann noch der KLoadBalancer als Klient der Bridge gestartet werden, dann enthält die ini-Datei nur den Abschnitt

- KLoadBalancer

host = <hostname:portnumber>

siehe Kommandozeilenparameter -host

Speicher-Einstellungen:

Die folgenden drei Parameter dienen zur Konfiguration der Speicherzuteilung und -nutzung. Erlaubt ist die Angabe von Werten entweder in Megabyte oder in tatsächlichen Byte, wobei die Annahme gilt, dass sich Werte kleiner als 1048576 auf Megabyte-Angaben beziehen.

`maxMemory=<Integer, in MB>`

Maximal erlaubte Hauptspeicherbelegung. Minimal 50 MB, standardmäßig gesamter physikalisch vorhandener Hauptspeicher (unter Windows) bzw. 512 MB.

`growthRegimeUpperBound=<Integer, in MB>`

Hauptspeicherbelegung ab der verstärkt versucht wird, Speicher freizugeben. Standardmäßig 0.6 * maxMemory.

`freeMemoryBound=<Integer, in MB> [10]`

Falls belegter, aber nicht mehr benötigter Speicher diese Grenze überschreitet, wird er wieder freigegeben.

`minAge=<Integer> [30]`

Mindestdauer (in Sekunden), die ein Cluster im Speicher bleibt. Ein Cluster ist eine Menge von Objekten, die immer zusammen am Stück geladen werden (z.B. ein Individuum mit all seinen (Meta)eigenschaften. Cluster, die längere Zeit nicht mehr verwendet werden, werden bei Bedarf ausgelagert.

`unloadInterval=<Integer> [10]`

Mindestdauer (in Sekunden) zwischen zwei Cluster-Auslagerungen

`unloadSize=<Integer> [4000]`

Mindestanzahl an geladenen Cluster, ab der ausgelagert wird

`keepSize=<Integer> [3500]`

Zahl der Cluster, die beim Auslagern behalten werden

`useProxyValueHolder=true/false`

Um den Mediator bei Suchen zu entlasten, kann die Option useProxyValueHolder=false verwendet werden. Der Client lädt dann Indizes in den Hauptspeicher, statt per RPCs den Mediator abzufragen.Der Nachteil dieser Option ist, dass dann nur noch lesender Zugriff möglich ist.

`loadIndexes=true/false`

Über diese Option werden Indizes ebenfalls in den Speicher geladen. Es ist aber auch weiterhin schreibender Zugriff möglich. Die Option kann bei allen Clients inkl. Knowledge-Builder

aktiviert werden.

Logging-Einstellungen:

Zu den Konfigurationsmöglichkeiten des Loggings siehe Logging-Einstellungen im Kapitel 11.1.2 Konfigurationsdatei.

3.3.4 REST-Bridge

3.3.4.1 Einführung

Die REST-Bridge Application ermöglicht den lesenden und schreibenden Zugriff auf i-views über eine RESTful Services-Architektur. Die Schnittstelle steht als HTTP- oder HTTPS-Version zur Verfügung.

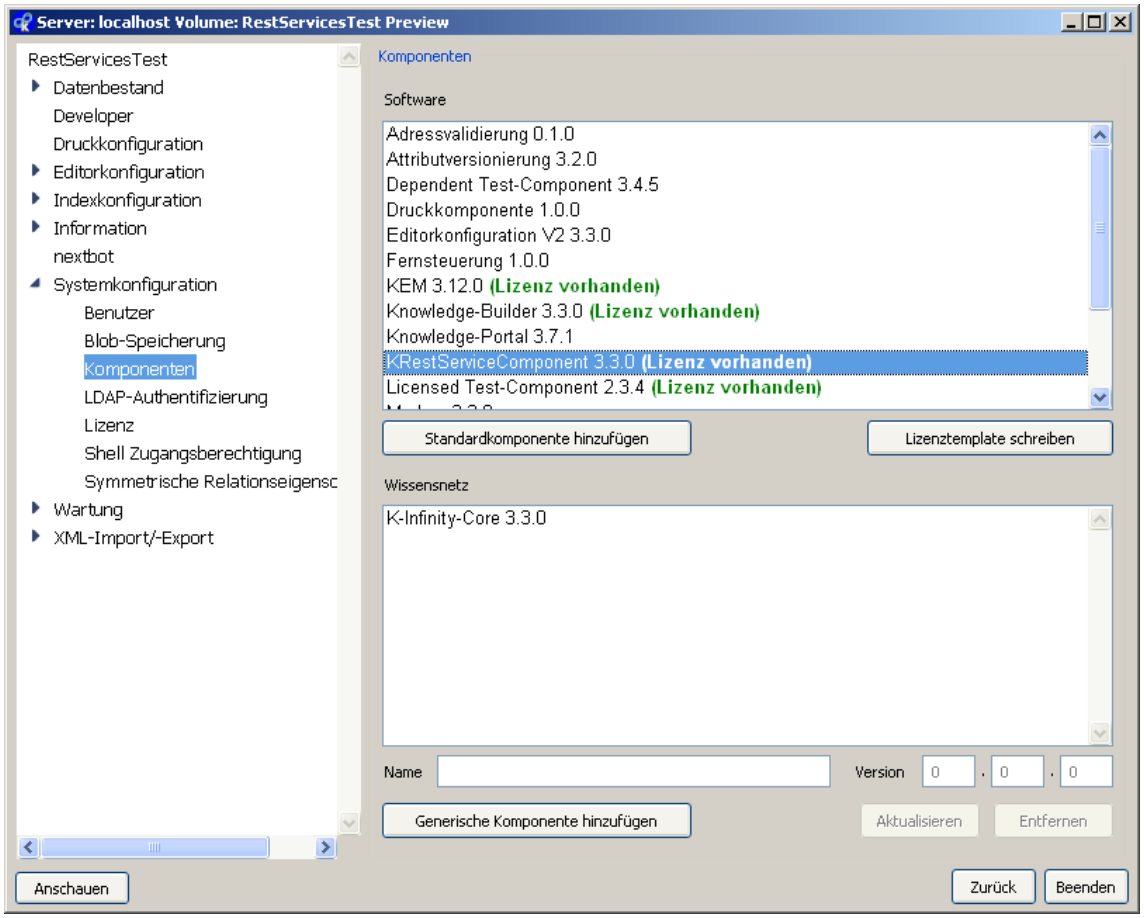
Die REST-Bridge läuft innerhalb der Standard-Bridge von i-views (bridge.exe).

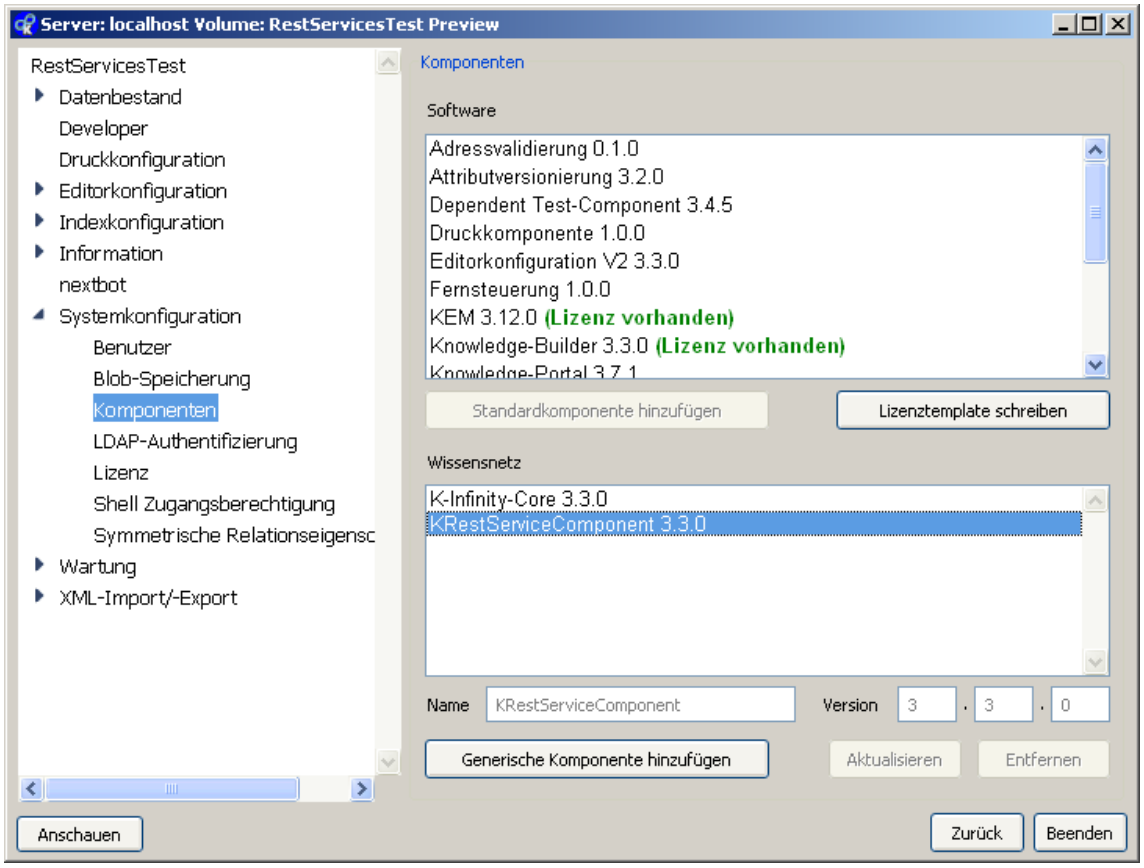
Die Schnittstelle wird vollständig durch Konfigurations-Individuen im Wissensnetz konfiguriert. Der Rückgabewert eines REST-Aufrufes ist eine beliebige Zeichenkette, in der Regel in einem Format, das der aufrufende Client gut weiterverarbeiten kann (z.B. XML oder JSON).

3.3.4.2 Installation

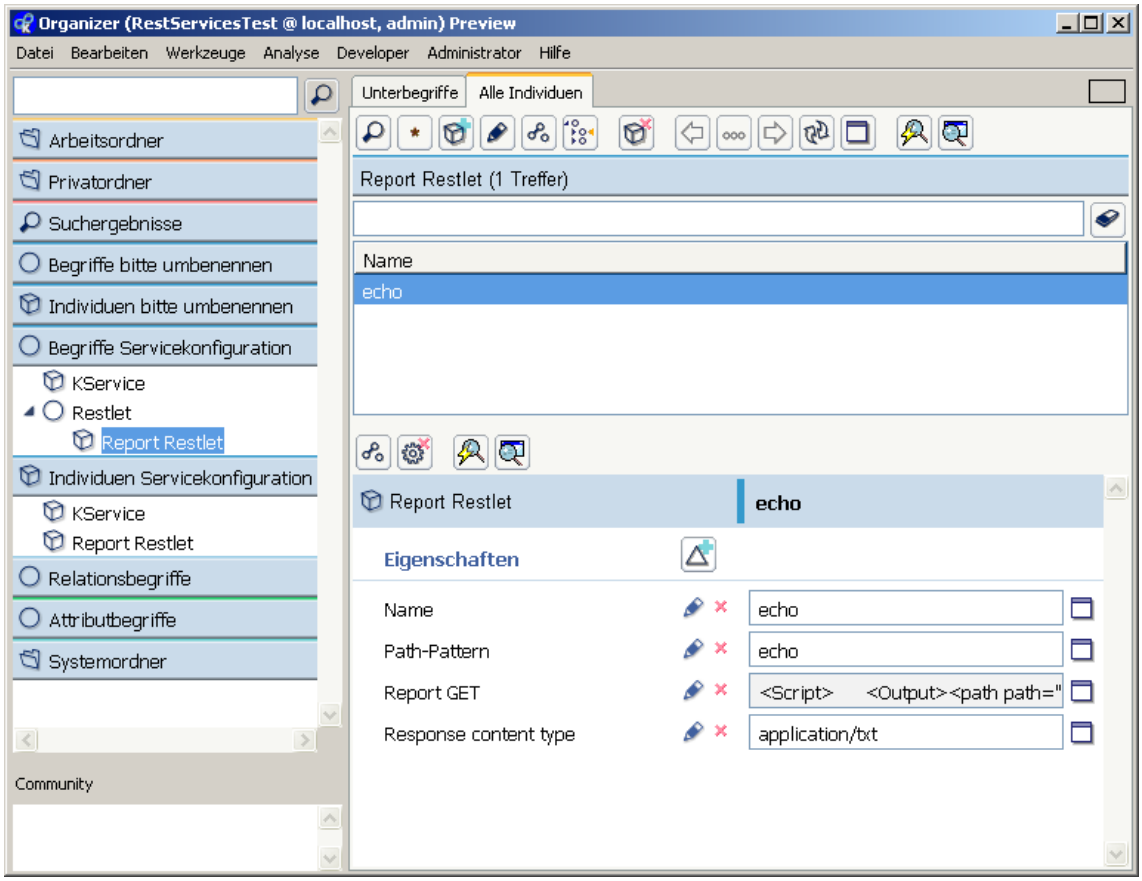
3.3.4.2.1 Volume vorbereiten

Durch das Hinzufügen der Softwarekomponente "KRestServiceComponent" im Admin-Tool wird im Wissensnetz das benötigte Schema angelegt.


Server: localhost Volume: RestServicesTest Preview
RestServicesTest
Datenbestand
Developer
Druckkonfiguration
Editorkonfiguration
Indexkonfiguration
Information
nextbot
Systemkonfiguration
Benutzer
Blob-Speicherung
Komponenten
LDAP-Authentifizierung
Lizenz
Shell Zugangsberechtigung
Symmetrische Relationseigensc
Wartung
XML-Import/-Export
Komponenten
Software
Adressvalidierung 0.1.0
Attributversionierung 3.2.0
Dependent Test-Component 3.4.5
Druckkomponente 1.0.0
Editorkonfiguration V2 3.3.0
Fernsteuerung 1.0.0
KEM 3.12.0 (Lizenz vorhanden)
Knowledge-Builder 3.3.0 (Lizenz vorhanden)
Knowledge-Portal 3.7.1
KRestServiceComponent 3.3.0 (Lizenz vorhanden)
Licensed Test-Component 2.3.4 (Lizenz vorhanden)
Standardkomponente hinzufügen
Lizenztemplate schreiben
Wissensnetz
K-Infinity-Core 3.3.0
Name
Version
0
0
0
Generische Komponente hinzufügen
Aktualisieren
Entfernen
Anschauen
Zurück
Beenden



Das Schema wird in einem Teilnetz des Wissensnetzes namens "Servicekonfiguration" angelegt:



3.3.4.2.2 Bridge konfigurieren

Die REST-Schnittstelle wird durch die Standard-Bridge-Komponente von i-views bereitgestellt, sofern in der zugehörigen Konfigurationsdatei **bridge.ini** eine Kategorie **KHTTPRestBridge** bzw. **KHTTPSRestBridge** eingetragen ist:

```
[KHTTPRestBridge] volume=name des Wissensnetzes port=port, unter dem der Service erreichbar sein s
```

Für die HTTPS-Version müssen in den Konfiguirationsdatei zusätzlich die Dateipfade für Zertifikat und Private Key angegeben werden:

```
[KHTTPSRestBridge] volume=name des Wissensnetzes port=port, unter dem der Service erreichbar sein
```

In der Konfigurationssektion "KHTTPRestBridge" oder "KHTTPSRestBridge" können außerdem noch die folgenden speziellen Konfigurationsoptionen eingetragen werden:

Name	Beschreibung

realm	Name, der bei aktivierter Authentifizierung als Realm-Name an den Client zurückgegeben wird. Web-Browser zeigen den Realm-Namen typischerweise in Dialogfenstern zur Authentifizierung als Applikationsnamen an, damit der Benutzer weiß, wer die Authentifizierung fordert. Standardwert: REST

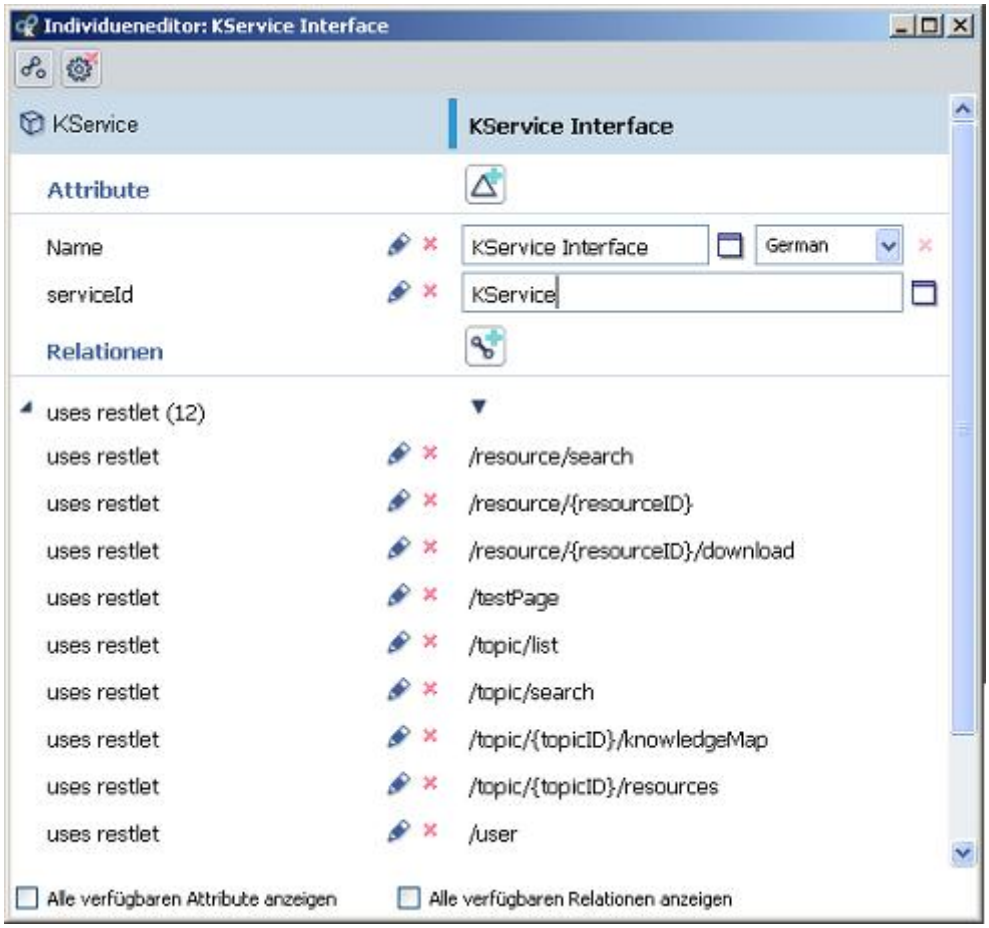
3.3.4.3 Konfiguration

3.3.4.3.1 Service Instanz definieren

Zunächst muss eine neue Service-Instanz angelegt werden. Sie haben die Möglichkeit mehrere Service-Instanzen zu konfigurieren, die von unterschiedlichen KService-Applikationen angesprochen werden. Erzeugen Sie dazu ein neues Individuum vom Typ "KService".

An dieser Instanz muss ein frei wählbarer Name und die serviceId vergeben werden. Die serviceId muss dem serviceIdentifier in der kservice.properties entsprechen. Haben Sie diesen nicht geändert, ist der Defaultwert "KService".

Diese Instanz dient als Einstiegspunkt für die Webapplikation. Alle Restlets, die unter dieser Service-Instanz verfügbar sein sollen, müssen auch über die Relation "uses restlet" zugeordnet werden!



Reinitialisieren des Services

In diesem Abschnitt wird beschrieben, wie eine Reinitialisierung des Services durchgeführt werden kann, ohne die *Bridge* neu starten zu müssen.

Dieser Vorgang wird in zwei Schritte durchgeführt:

1. An dem *Rest-Service*, der reinitialisiert werden soll, muss das boolesche Attribut *Reinitialize service* mit gesetztem Häkchen angelegt werden.

2. Ein Request für ein bereits registriertes Restlet muss an die Bridge gesendet werden.

   -> Reinitialisieren des Services wird durchgeführt. Das Häkchen wird automatisch entfernt. Die erfolgreiche Durchfühung kann man im Bridge.log überprüfen.

Anmerkung: Sind mehrere Bridges für denselben Service konfiguriert, muss der obige Vorgang für jede Bridge einzeln wiederholt werden.

3.3.4.3.2 Restlets definieren

Restlets allgemein

Jede Methode, die Bestandteil der Service-API ist, wird durch ein eigenes Restlet-Individuum repräsentiert. Das Restlet wird durch einen frei definierbares Pfad-Muster adressiert. Hier sollten die allgemeinen Regeln für RESTful Services eingehalten werden. Im Pfad-Muster kön-

nen in geschweiften Klammern variable Teile spezifiziert werden, die vom Restlet ausgewertet werden kann.

Beispiel:

```
echo/{message}
```

Variablen

Im Path-Pattern können Variablen definiert werden. Diese werden in geschweiften Klammern geschrieben. Diese Variablen sind required und somit fester Bestandteil des Path-Patterns. Optionale Variablen werden im QueryString übergeben.

Beispiel:

Pattern: http://servername:8080/api/customer/***{customerID}***/address
Aufruf: http://servername:8080/api/customer/***C4317***/address?***show=private***

customerID ist eine required Variable. Für die Verarbeitung des Aufrufs, wird die Variable customerID mit dem Wert "*C4317*" befüllt. Die definierte Logik kann so den passenden Datensatz selektieren.

show ist eine optionale Variable. Sie kann, muss beim Aufruf aber nicht angegeben werden. In diesem Beispiel würde das Restlet die Variable abfragen. Wenn sie gesetzt ist, wird nur die Private Adresse geliefert. Fehlt die Angabe, werden alle Adressen geliefert.

Vordefinierte Variablen

Zusätzlich zu den Variablen, die im Path-Pattern oder durch den Aufruf (z.B. im Body) angegeben werden, werden Variablen automatisch vom System gesetzt bzw. befüllt.

requestBody

Mit der Variablen *requestBody* kann auf den Inhalt des Bodys eines Requests zugegriffen werden.

Beispielaufruf zur Kontrollausgabe:

```
<Script>     <If test="var(requestBody)/size() > 0">         <do> <Output><path path="var(req
```

requestHeader

In der Variablen *requestHeader* ist ein Dictionary enthalten, welches die Headereinträge des HTTP-Requestes enthält.

Mit Hilfe der Funktionen *groupKey* und *groupValue* kann auf die Inhalte zugegriffen werden.

Beispielaufruf zur Kontrollausgabe:

```
<XMLElement name="RequestHeader"> <Path path="var(requestHeader)"> <Each> <XMLElement name="HT
```

Ausschnitt des Results des Beispielaufrufes:

```
<RequestHeader> <HTTPHeader name="connection">keep-alive</HTTPHeader> <HTTPHeader name="accept
```

Tipp: Für die Zerlegung eines GroupValues kann die Funktion *AnalyzeString* verwendet werden:

```
<AnalyzeString string="groupValue()" regex=",">
    <matchingSubstring/>
    <nonMatchingSubstring>
```

```
    <XMLElement name="value">
        <path path="regexNonMatch()"/>
    </XMLElement><cr/>
</nonMatchingSubstring> </AnalyzeString>
```

responseHeader

Die Variable *responseHeader* wird initial mit einem leeren Dictionary belegt. Somit ist diese Variable vorhanden und kann mit dem KPath-Ausdruck *atKeyPut()* befüllt werden.

```
<path path="var(responseHeader)/atKeyPut(einSchluessel, einWert)"/>
```

Wichtig: Der Schlüssel (*key*) muss eine Zeichenkette sein. Der Wert (*value*) darf eine Zeichenkette oder ein Ganzzahl sein, aber keine Menge.

httpStatusCode

Die Variable *httpStatusCode* kann auf eine beliebige Ganzzahl zwischen 100 und 599 gesetzt werden. Diese Variable wird dem HTTP-Response-Request als Statuscode gesetzt.

Operation

Zu den Regeln der RESTFul Services gehört die Unterscheidung der Operation durch die Art der Anfrage. Ein Http-GET-Request steht dabei für einen lesenden Zugriff. POST für eine modifikation, PUT für neu Erzeugen und DELETE für löschen.

Die Kservice-Applikation unterscheidet zwischen diesen Request-Typen und erlaubt bei GET-Request keine Modifikation der Daten! Wählen Sie deshalb bitte immer den passenden Request-Typ.

Report Restlet

Das Report-Restlet führt einen KSkript-Report aus und schickt die generierten Ausgabe als Antwort zurück. Alle Variablen aus dem Pfad-Muster und die URL-Queryparameter stehen im KSkript als Variablen zur Verfügung.

Beim ReportRestlet können folgende Eigenschaften konfiguriert werden:

Eigenschaft	Erforderlich	Beschreibung
Path-Pattern	Ja	Pfad zum Restlet ab der Basis-URL. Variablen sind mit geschweiften Klammern zu versehen. *Bsp: user/{userID}/details*
Report GET	Nein. Standard: GET liefert http 404	Report, der bei einem GET-Request ausgeführt werden soll.
Report POST	Nein. Standard: POST liefert http 404	Report der bei einem POST-Request ausgeführt werden soll.

Report PUT	Nein. Standard: PUT liefert http 404	Report der bei einem PUT-Request ausgeführt werden soll.
Report DELETE	Nein. Standard: DELETE liefert http 404	Report der bei einem DELETE-Request ausgeführt werden soll.
Response content type	Nein. Standard: text/plain	Content-Type des Response-Headers (z.B. text/html)
Requires authentication	Nein. Standard: false	nein: Request ist offen zugänglich und läuft ohne Rechteprüfung ja: User muss sich (per http Basic authorization) authentifizieren, der Request wird mit den Zugriffsrechten dieses Users ausgewertet.
Part of Service	Ja	Relation zur Service-Instanz, unter der das Restlet verfügbar sein soll

Konfigurationsbeispiel

Blob Restlet

Das Blob-Restlet ermöglicht den Upload, Download und Löschen eines Blob-Attributes.

Beim Restlet lassen sich zwei Skript-Attribute anlegen, die folgende Objekte zurückliefern müssen.

Element	**Upload**		**Download**	**Löschen**
	neu	bestehend		
Report to resolve attribute (rest.reportToResolveBlob)	Proto des neu anzulegenden Blob-Attributes	Existierendes Blob-Attribute, das verändert werden soll	Existierendes Blob-Attribute, das zurückgegeben werden soll	Existierendes Blob-Attribute, das gelöscht werden soll
Report to resolve topic (rest.reportToResolveTopic)	Topic an dem das Blob-Attribut angelegt werden soll	-	-	-

Beim Upload müssen im Body des HTTP-Requests folgende Parameter befüllt sein:

file-name	Name bzw. Datei-Name des anzulegenden bzw. zu modifizierenden Blob-Attributes
con-tent	Binärdaten des Blobs

Liste der Standardparameter:

Variable	Required	Beschreibung
Path -Pattern	Ja	Pfad zum Restlet ab der Basis-URL. Variablen sind mit geschweiften Klammern zu versehen. *Bsp: download/{blobID}*
Requires autheticaton	Default : false	Gibt an ob sich der User authentifizieren muss
Execute as system user	Default: false	Ermöglicht den Download als Systemuser.
Part of Service	Ja	Relation zur Service-Instanz, unter der das Restlet verfügbar sein soll

Static File Restlet

Mit Hilfe des Static File Restlet können statische Dateien über den REST-Server ausgeliefert werden. Dieser Restlet-Typ kann z.B. genutzt werden, um einfache Anwendungen vollständig durch den REST-Server und ohne Verwendung eines weiteren Webservers zu bedienen.

Zur Konfiguration wird lediglich das Attribut 'Path-Pattern' benötigt, in dem der Name eines Verzeichnisses im Arbeitsverzeichnis der REST-Bridge angegeben wird. Die Dateien in diesem und in untergeordneten Verzeichnissen können dann über die URL

```
http://{server:port}/{service name}/{path pattern}/{relative file path}
```

abgerufen werden.

3.3.4.3.3 Vordefinierte Services

Referenzen auf i-views Objekte

Wenn eine Referenz auf ein i-views-Objekt in einer URL spezifiziert wird, werden folgende Formate berücksichtigt.

1. Für Werte der Form **ID**<**Zahl**>_<**Zahl**> wird angenommen, dass es sich um eine Objekt-ID handelt über die das Objekt aufgelöst wird.
2. Ist mindestens ein **Tilde ("~")** im Wert enthalten, so wird der Teil vor dem ersten Tilde als interner Name eines Attributs interpretiert und der Teil hinter dem Tilde als Attributwert. Es wird ein eindeutiges Objekt gesucht, dass für dieses Attribut den vorgegebenen Wert hat.
3. Ansonsten wird der Wert als **interner Name** interpretiert. Es wird er Typ mit dem entsprechenden internem Namen gesucht.

Beispiele:

ID135_71384	Referenziert das Objekt mit der Objekt-ID "ID135_71384"
Person	Referenziert den Typ mit dem internen Namen "Person"
templateId~abc	Referenziert ein Objekt, dass ein Attribut mit dem internen Namen "templateId" und dem Attributwert "abc" hat.

Content-Disposition

Über den Query-Parameter "**content-disposition**" eines Requests kann das Response-Header-Field "**Content-Disposition**" der zugehörigen "Response" gesetzt werden. Der Default-Wert ist ansonsten "inline".

Topic Icon

Über den folgenden Pfad kann man die Bilddatei zu einem gegebenen Topic laden. Wenn ein Individuum keine eigene Bilddatei hat, wird auf die Bilddatei des Typs zurückgegriffen, welche wiederum vererbbar ist. Über den optionalen Parameter "size" kann man die Bilddatei mit der am ehesten passenden Größe selektieren, vorausgesetzt im Wissensnetz sind mehrere Bildgrößen hinterlegt.

http://{server:port}/baseService/topicIcon/{topicID}?size=10

Objektliste

Über den folgenden Pfad kann eine Objektliste im JSON-Format angefordert werden:

http://{server:port}/baseService/{conceptLocator}/objectList

Der Typ der Objektliste wird über den Parameter "**conceptLocator**" referenziert, dem Format für Topic-Referenzen in der Rest-URL folgt. (siehe Verknüpfung)

Alternativ kann der "conceptLocator" auch den einen Prototyp (Individuum oder Typ) des zu verwendenden Typs referenzieren.

Der optionale Parameter "**name**" bestimmt die Objektliste, die für die Ausgabe verwendet werden soll.

Filter

Über den optionalen und mehrwertigen Query-Parameter "**filter**" kann die Objektliste gefiltert werden. Ein Filter hat zwei mögliche Formen:

1. <Spalten-Name/Spalten-Nr.> ~ <Operator> ~ <Wert>
2. <Spalten-Name/Spalten-Nr.> ~ <Wert>

Mögliche Operatoren sind: equal, notEqual, greater, less, greaterOrEqual, lessOrEqual, equalCardinality, containsPhrase, covers, isCoveredBy, distance, fulltext, equalGeo, equalPresentTime, greaterOverlaps, greaterPresentTime, lessOverlaps, lessPresentTime, equalMaxCardinality, equalMinCardinality, overlaps, unmodifiedEqual.

Sortierung

Über den optionalen und mehrwertigen Query-Parameter "**sort**" kann die Objektliste sortiert werden. Die Reihenfolge der Sortierparameter bestimmt die Sortierpriorität. Die Angabe der Sortierung hat zwei mögliche Formen:

1. <Spalten-Name>
2. {-}<Spalten-Nr.>

Stellt man in Variante 2 ein Minus vor, wird absteigend sortiert, sonst aufsteigend.

Startmenge der Liste setzen

Über den optionalen und QueryParameter "**elements**"kann eine Komma-separierte Liste von Topic-Referenzen übergeben werden, die als Listenelemente verwendet werden sollen.

Da die Liste der Element ggf. sehr lang ist, kann der Request auch als POST geschickt und die Parameter als Form-Parameter übergeben werden.

Startmenge der Liste über KPath setzen

Über die optionalen Query-Parameter "**elementsPath**" und "**startTopic**" können die initialen Elemente der Liste berechnet werden. Sind diese Parameter nicht gegeben, besteht die Ausgangsmenge aus allen Individuen bzw. allen Untertypen (im Falle einer Typ-Objektliste) des per "conceptLocator" festgelegten Typs.

Dabei ist "elementsPath" ein KPath-Ausdruck und "startTopic" eine Referenz auf das Topic, mit dem die Auswertung des KPath gestartet werden soll. Die Form des Parameters "startTopic" entspricht der des "conceptLocator".

Vererbung

Über den optionalen Query-Parameter "**disableInheritance**" kann die Vererbung unterdrückt werden. Der Paramater macht nur Sinn, wenn kein "elementsPath" gesetzt ist.

JSON-Ausgabeformat (Beispiel)


```
{
 rows: [{
  topicID: "ID850_337000250",
  row: ["CS",
  "Schuckmann",
  "Christian",
  "240",
  "c.schuckmann@i-views.de",
  "10",
  "",
  "",
  "IT-Cluster P4, IT-Cluster P5"]
 },
```

```
{
 topicID: "ID12068_152328826",
 row: ["",
 "Smith",
 "John",
 "",
 "",
 "",
 "",
 "",
 ""]
},
{
 topicID: "ID2438_366678497",
 row: ["SY",
 "Stoye",
 "Sabine",
 "421",
 "s.stoye@i-views.de",
 "",
 "",
 "",
 "Knowledge-Portal 3, Presales, SemGoRiCo, Support (HSNR)"]
}],
columnDescriptions: [{
 label: "Login",
 type: "string",
 columnId: "1"
},
{
 label: "Nachname",
 type: "string",
 columnId: "2"
},
{
 label: "Vorname",
 type: "string",
 columnId: "3"
},
{
 label: "Telefon",
 type: "string",
 columnId: "4"
},
{
 label: "Email",
 type: "string",
 columnId: "5"
},
{
 label: "Verfügbarkeit",
 type: "number",
 columnId: "6"
```

```
 },
 {
  label: "Aufwand",
  type: "string",
  columnId: "7"
 },
 {
  label: "erstellt am",
  type: "dateTime",
  columnId: "8"
 },
 {
  label: "Projekt",
  type: "string",
  columnId: "9"
 }]
}
```

Objektlistendruckvorlage

Über den folgenden Pfad kann eine Objektliste in eine 'Druckvorlage für Liste' (siehe Kapitel 'Schema der Druckvorlagen) gefüllt und das Resultat heruntergeladen werden:

http://{server:port}/baseService/{conceptLocator}/objectList/printTemplate/{templateLocator}/{filenar

Der Service funktioniert genau wie der Abruf einer Objektliste, trägt aber als zuätzlichen Parameter eine Referenz auf das Individuum des Typs 'Druckvorlage für Liste' im Wissensnetz.

"**templateLocator**" muss eines der unter "Allgemeines" beschriebenen Formate haben

Der optionale Pfad-Parameter "filename" wird nicht ausgewertet und dient dem besseren Browser-Handling.

Über das Header-Field "**Accept**" wird gesteuert, in welches Ausgabeformat konvertiert werden soll. Fehlt das Header-Field oder ist der Wert "***/***", findet keine Konvertierung statt. Mehrwertige Accept werden nicht unterstützt und resultieren in einer Fehlermeldung.

Über den optionalen Query-Parameter "**targetMimeType**" kann der Wert des "Accept" Header-Fields überschrieben werden. Dies ist notwendig, wenn man den Request aus einem Browser aufrufen möchte und dort keinen Einfluss auf die Header-Fields hat.

Topic drucken

Über den folgenden Pfad kann ein Topic in ein Drucklistentemplate gefüllt und das Resultat heruntergeladen werden:

http://{server:port}/baseService/{topicLocator}/printTemplate/{templateLocator}/{filename}

"**templateLocator**" muss eines der unter "Allgemeines" beschriebenen Formate haben

Der optionale Pfad-Parameter "filename" wird nicht ausgewertet und dient dem besseren Browser-Handling.

Über das Header-Field "**Accept**" wird gesteuert, in welches Ausgabeformat konvertiert werden soll. Fehlt das Header-Field oder ist der Wert "***/***", findet keine Konvertierung statt. Mehrwertige Accept werden nicht unterstützt und resultieren in einer Fehlermeldung.

Über den optionalen Query-Parameter "**targetMimeType**" kann der Wert des "Accept" Header-Fields überschrieben werden. Dies ist notwendig, wenn man den Request aus einem Browser aufrufen möchte und dort keinen Einfluss auf die Header-Fields hat.

Dokumentformatkonvertierung

Über den folgenden Pfad kann ein Dokument in ein anderes Format umgewandelt werden (z.B. odt in pdf):

http://{server:port}/baseService/jodconverter/service

Der Service bildet den JOD-Konverter (siehe http://sourceforge.net/projects/jodconverter/) ab und dient der Abwärtskompatibilität für Installationen, die bisher mit dem JOD-Konverter betrieben wurden.

Damit der Service funktioniert muss Open/LibreOffice (ab Version 4.0) installiert sein und die Konfigurationsdatei "bridge.ini" muss einen Eintrag enthalten, der auf die Datei "soffice" verweist:

```
[file-format-conversion] sofficePath="C:\Program Files (x86)\LibreOffice 4.0\program\soffice.exe"
```

3.3.5 KEM-Bridge

KEMBridge

Abschnittsname:

[KEMBridge]

port = <portnumber>

Angabe des Ports, unter welchem die KEMBridge reagiert. Bei Nichtangabe gilt der Defaultwert von 4713.

ldapHost = <hostname:portnumber>

Angabe des LDAP-Hostes, welcher kontaktiert werden soll, für den Fall, dass die Authentifikation über LDAP stattfinden soll. Ist dieser Parameter angegeben, so muss die Authentifikation über LDAP abgewickelt werden.

maxLoginCount = <number>

Angabe der maximalen Fehlversuche beim Einloggen, bis der entsprechende Benutzer im Netz gesperrt wird. Danach ist ein Einloggen nur nach Entsperren per Knowledge-Builder möglich. Falls der Wert nicht gesetzt ist, ist ein fehlerhaftes Einloggen praktisch beliebig oft möglich.

Um ein Sperren des Benutzers im Wissensnetz zu ermöglichen, muss für Individuen des Personenkonzepts ein boolesches Attribut mit internem Namen userlock und Defaultwert false definiert sein.

KEMrestrictToIPAddress = <IP-Adresse>

Wenn dieser Parameter gesetzt ist, werden nur Verbindungen von dem hier angegebenen Host akzeptiert.

trustedLoginEnabled = <true/false>

Erlaubt ein Einloggen ohne Passwort mittels des Requests: „newAuthenticatedUser(username)“.

preventSessionReplay=<true/false>

[default=false]

Dieser Parameter gibt an, dass jede schreibende Session ihren eigenen geschützten Wissensnetzzugriff erhält, so dass der sonst übliche Mechanismus, die Aktionen einer deaktivierten Session beim Reaktivieren erneut auszuführen, um den letzten aktuellen Editorstand zu erhalten, unnötig wird.

KEMStreamingBridge

Abschnittsname:

[KEMStreamingBridge]

port = <portnumber>

Angabe des Ports, unter welchem die KEMStreamingBridge reagiert. Bei Nichtangabe gilt der Defaultwert von 4714.

3.3.6 KLoadBalancer

Der KLoadBalancer kann eingesetzt werden, um die Services und Verfügbarkeit der KEMBridge und KEMStreamingBridge zu skalieren.

Im Abschnitt [KLoadBalancer] können/müssen die folgenden Angaben gemacht werden, um den gewünschten Betriebsmodus zu erreichen:

- allowRemoteShutdown (Default-Wert false)
- autoRestart (Default-Wert true)
- directory (Default-Wert aktuelles Arbeitsverzeichnis, in dem der KLoadBalancer gestartet wurde)
- executable (Default-Wert 'bridge.exe')
- image (Default-Wert 'bridge.im')
- vm (Default-Wert 'visual')
- hostname (Default-Wert Localhost)
- **configNames (benötigter Wert, nicht optional)**
- parameters (Default-Wert leer)

Der Parameter #configNames dient der weiteren Konfiguration der zu startenden KEMBridges und KEMStreamingBridges, je Einzel-Konfiguration wird ein Bridge-Typ gesteuert. Die Konfigurationsnamen sind durch Komma zu trennen.

Hier ein Beispiel für eine KLoadBalancer-ini-Datei:

```
[Default] [KLoadBalancer] hostname=ws01 port=30003 directory=C:\3.2\balancing executable=bridge.ex
```

Beim Start werden gemäß der beiden Konfigurationen KEMBridges und KEMStreamingBridges gestartet. Da zum Betrieb dieselbe Software wie für den Betrieb des KLoadBalancers verwendet wird, sind in diesem Abschnitt die Angabe der Parameter #executable, #image und #vm (für Linux-Betrieb), #hostname, #directory und #parameters nötig.

executable / image, vm; directory: Angaben, wie die einzelnen Bridges gestartet werden können. Unter Windows wird die Angabe von #executable und #directory benötigt, unter Linux die Angabe von #image, #vm und #directory.

hostname / port: Der Hostname, der den zu startenden Bridges als für Verwaltungszwecke

zu kontaktierender KLoadBalancer genannt wird. Falls hier keine Angabe gemacht wird, wird der Rechnername ermittelt und dieser verwendet. Der Port gibt an, unter welchem Port die Bridges den Balancer ansprechen, Default-Wert ist 4715.

Vorsicht: Der Name des jeweiligen Mediators, den die Bridges zum Abrufen von Daten kontaktieren, ist in den jeweiligen ini-Dateien gemäß Konfigurationsabschnitt einzutragen!

parameters: Ein Feld, mit dem zusätzliche Angaben in die Kommandozeile der zu startenden Bridges eingefügt werden können, ist für alle zu startenden Bridges gleich.

allowRemoteShutdown: Parameter, der angibt, ob der KLoadBalancer per shutdown-Request per remote-Zugriff zu beenden ist.

autoRestart: Parameter, der angibt, ob eine gestoppte KEMBridge nach dem shutdown erneut zu starten ist, mit neuer ID.

In jedem Konfigurationsabschnitt müssen zusätzliche Angaben gemacht werden:

- bridgeClientClassName (nicht optional, nur eine Angabe je Abschnitt möglich. Bitte obige Schreibweise beachten!)
- inifile (ini-Datei mit Einstellungen für diesen Typ zu startende Bridge)
- bridgeLogfile (Muster eines Logfile-Namens, in den ein Platzhalter eingefügt wird, <id>, über den sich die Log-Dateien der einzelnen Bridges auseinanderhalten lassen, wird mit der laufenden Nummer der gestarteten Bridge ersetzt)
- maxBridges (Anzahl der maximal zu startenden Bridges des angegebenen Typs, nicht optional!)
- sslEnabled (Angabe, ob die Bridges dieses Typs SSL für den Verbindungsaufbau verwenden sollen, Default-Wert false)

Zur Beachtung: Der Parameter #directory gibt das Arbeitsverzeichnis an, in dem die in den Konfigurationsabschnitten angegebenen Dateien gesucht und ggfs. angelegt werden. Software und ini-Datei für den Start des KLoadBalancers können sich an anderer Stelle befinden.

Die ini-Dateien der jeweiligen Bridges müssen wie gewohnt aufgebaut werden. Ein Beispiel für die im obigen Konfigurationsabschnitt KEM referenzierte ini-Datei ist hier angefügt:

```
[Default] host=mediator-hostname:30053 [KEMBridge] trustedLoginEnabled=true preventSessionReplay=t
```

Für Details sei auf Kapitel 5 "Konfigurationsdatei bridge.ini" verwiesen.

3.4 Jobclient

3.4.1 Allgemeines

Der Job-Client erbringt zum einen Dienste für andere i-views-Clients, um diese von rechenzeit- oder datenintensiven Aufgaben zu entlasten. Zum anderen dient er als Brücke zwischen i-views-Clients und externen Systemen.

Zu seinen wichtigsten Aufgaben gehört die Ausführung aller Arten von Suchen sowie die Auslieferung der Suchergebnisse an die Clients (Sortierung, textuelle Aufbereitung, Rechtefilterung).

Im Normalfall wartet der Client auf die Fertigstellung eines Auftrags (Synchronbetrieb).

Für die Ausführung komplexer Suchen, das Erstellen von Statistiken, Batch-Abgleiche, Datenaufbereitungen, Datenbereinigungen, etc. muss der Client nicht auf die Fertigstellung warten (Asynchronbetrieb). Das Ergebnis wird vom Service bereitgestellt und der Client wird be-

nachrichtigt. Das Ergebnis kann dann beliebige Zeit später eingesehen werden. Da das Ergebnis auch persistent gemacht wird, ist es auch nach einem Neustart des Systems bzw. im Falle eines Fail-Overs weiterhin verfügbar.

Funktionsweise:
In dem vom i-views-Mediator bereitgestellten geteilten Objektraum werden die Aufträge der Clients an die Services in sogenannten Pools abgelegt. Alle i-views Job-Clients werden über neue Aufträge notifiziert und bewerben sich - sofern sie aktuell frei sind - für die Bearbeitung des neuen Auftrags. Nach Bearbeitung des Auftrags wird das Resultat wieder im geteilten Objektraum bereitgestellt, der beauftragende Client wird benachrichtigt und das Ergebnis kann abgerufen und zur Anzeige gebracht werden. Somit beauftragt der Client zwar logisch einen Job-Client, physikalisch läuft die Kommunikation aber immer über den i-views-Server. Für den Client ist es transparent, welcher Job-Client seinen Auftrag ausführt, sowie es für den Job-Client transparent ist, wo der Auftrag herkommt und wie viele parallele Job-Clients zurzeit aktiv sind. Für Administratoren ist die Installation und Wartung der Job-Clients daher sehr einfach und flexibel. Job-Clients lassen sich beliebig skalieren, auf verschiedene Rechner verteilen und dynamisch zu- und abschalten. Eine externe Clusterung oder sonstige Orchestrierung ist nicht erforderlich.

Technische Daten:
Multi-Platform Executable auf Basis der VisualWorks Smalltalk Virtual Machine (jobclient.exe bzw. jobclient.im)
Benötigt eine TCP/IP-Verbindung zum i-views-Server
Automatische Lastverteilung zwischen den Services
Job-Clients können zu jeder Zeit zugeschaltet oder heruntergefahren werden
Standby-Modus bei zeitweiliger Nicht-Verfügbarkeit benötigter Ressourcen

3.4.2 Konfiguration des Job-Clients

3.4.2.1 Konfigurationsdatei "jobclient.ini"

Die Konfiguration des Job-Clients wird in der Ini-Datei vorgenommen. Falls nicht durch den Aufrufparameter "-inifile" spezifiziert wird "jobclient.ini" als Konfigurationsdatei verwendet.

`host=<Hostname:Portnummer>`

Name / IP-Adresse des Servers

`volume=<Volumename>`

Der Name des Wissensnetzes, auf dem gearbeitet werden soll.

`jobPools=Jobname1 [,Jobname2, ...]`

Angabe, welche Jobs der Job-Client abarbeiten soll. Die Namen der zu startenden Job-Pools sind hier kommagetrennt anzugeben. Alternativ kann auch die Kategorie (z.B. "index") angegeben werden. Es werden dann alle Job-Pools dieser Kategorie ausgewählt.

Beispiel:

`jobPools=KScriptJob, query`

Die möglichen Typen werden in den Unterkapiteln dargestellt.

`cacheDir=<Verzeichns>`

Beschreibung des Ortes, an dem der Cache für den Job-Client angelegt wird.

`volumeAccessor=CatBSBlockFileVolumeAccessor oder CatCSVolumeFileStorageAccessor`

Beschreibung der Speicherart des Caches. Wenn nicht angegeben wird CatBSBlockFileVolumeAccessor verwendet. Diese Speicherart ist vor allem bei großen Netzen zu empfehlen, da CatCSVolumeFileStorageAccessor eine große Anzahl an Dateien anlegen würde.

`maxCacheSize=<Größe in MB>`

Zielgröße des Caches

`shutDownTimeout=<Sekunden>`

Dauer, auf die beim Herunterfahren des Job-Clients auf die Beendigung der aktiven Jobs gewartet wird. Nach Ablauf werden die Jobs abgebrochen. Der Standardwert ist 10 Sekunden.

`enableLowSpaceHandler=true/false`

Mit dieser Option wird der LowSpaceHandler eingeschaltet. Dieser sollte auf jeden Fall bei großen Netzen eingeschaltet werden.

`useProxyValueHolder=true/false`

Mit dieser Option kann gesteuert werden, ob der Job-Client Indexzugriffe per RPC durchführt (true), oder Indizes in den Speicher lädt (false). Diese Option sollte ausgeschaltet werden, wenn der Mediator entlastet werden soll. Dabei sollte allerdings darauf geachtet werden, dass der Job-Client genug Speicher zur Verfügung hat. Falls der Job-Client für schreibende Jobs konfiguriert wurde, hat diese Option keinen Effekt, da dann der Indexzugriff immer per RPC durchgeführt wird. Es wird beim Start im Log eine Meldung ausgegeben, falls man den Wert auf false gesetzt hat.

`loadIndexes=true/false`

Seit 4.2 gibt es die neue Option loadIndexes=true. Indizes werden dann ebenfalls in den Speicher geladen. Im Gegensatz zur Option useProxyValueHolder ist aber auch weiterhin schreibender Zugriff möglich. Die Option kann bei allen Clients inkl. Knowledge-Builder aktiviert werden.

`name=<Job-Client-Name>`

Dieser Name wird verwendet, um den Job-Client im Admin-Tool in der Übersichtsliste aller Job-Clients zu identifizieren.

Speicher-Einstellungen:

Die folgenden drei Parameter dienen zur Konfiguration der Speicherzuteilung und -nutzung. Erlaubt ist die Angabe von Werten entweder in Megabyte oder in tatsächlichen Byte, wobei die Annahme gilt, dass sich Werte kleiner als 1048576 auf Megabyte-Angaben beziehen.

`maxMemory=<Integer, in MB>`

Maximal erlaubte Hauptspeicherbelegung. Minimal 50 MB, standardmäßig gesamter physikalisch vorhandener Hauptspeicher (unter Windows) bzw. 512 MB.

`growthRegimeUpperBound=<Integer, in MB>`

Hauptspeicherbelegung ab der verstärkt versucht wird, Speicher freizugeben. Standardmäßig 0.6 * maxMemory.

`freeMemoryBound=<Integer, in MB> [10]`

Falls belegter, aber nicht mehr benötigter Speicher diese Grenze überschreitet, wird er wieder freigegeben.

`minAge=<Integer> [30]`

Mindestdauer (in Sekunden), die ein Cluster im Speicher bleibt. Ein Cluster ist eine Menge

von Objekten, die immer zusammen am Stück geladen werden (z.B. ein Individuum mit all seinen (Meta)eigenschaften. Cluster, die längere Zeit nicht mehr verwendet werden, werden bei Bedarf ausgelagert.

`unloadInterval=<Integer> [10]`

Mindestdauer (in Sekunden) zwischen zwei Cluster-Auslagerungen

`unloadSize=<Integer> [4000]`

Mindestanzahl an geladenen Cluster, ab der ausgelagert wird

`keepSize=<Integer> [3500]`

Zahl der Cluster, die beim Auslagern behalten werden.

Logging-Einstellungen:

Zu den Konfigurationsmöglichkeiten des Loggings siehe Logging-Einstellungen im Kapitel 11.1.2 Konfigurationsdatei.

Lucene-Server-Konfiguration:

Die Einbindung von Lucene erfolgt über einen Job Client, dessen jobclient.ini-Datei dafür entsprechend konfiguriert werden muss. Nachfolgend eine Beispielkonfiguration:

```
[lucene]
directory=lucene-index
port=5100
pageSize=100
; Wildcards am Wortanfang sind standardmäßig verboten, da sehr langsam
; In dieser Konfiguration erlauben
allowLeadingWildcards=true

[JNI]
classPath=lucene-6.4.1\core\lucene-core-6.4.1.jar;lucene-6.4.1\analysis\common\lucene-analyzers-co
```

Das Verzeichnis *lucene-6.4.1* enthält die Binaries von Lucene. ImVerzeichnis *lucene-index* wird der Index gespeichert.

3.4.2.1.1 Indexjobs

- Kategorie(n): **index**

Werden für den Jobpool die unten angezeigten Jobklassen oder **index** angegeben, dann werden die Indexierungsaufträge vom Job-Client ausgeführt. Die Indexierungsaufträge sollten nur von einem einzigen Job-Client durchgeführt werden. Statt alle Jobklassen einzeln im Job-Pool aufzuzählen, kann auch der symbolische Name **index** verwendet werden.

KAddAllToIndexJob

- Bezeichnung: Attribute zum Index hinzufügen

KLightweightIndexJob

- Bezeichnung: Externen Index aktualisieren

  Ein externer Index wird über den KLightweightIndexJob gepflegt.

KLuceneAdminJob

- Bezeichnung: Lucene Verwaltungsaufgabe

  Der KLuceneIndexJob verwaltet einen extern aufgebauten Lucene-Index.

KRemoveIndexJob

- Bezeichnung: Attribute aus dem Index entfernen

KSyncIndexJob

- Bezeichnung: Index synchronisieren

KAddAllToIndexJob, KRemoveIndexJob und KSyncIndexJob werden benötigt, um die internen Indizes zu pflegen.

3.4.2.1.2 KBrainbotJob

- Kategorie(n): <keine>
- Bezeichnung: KBrainbotJob

Der KBrainbotJob führt Aktionen zur Pflege eines Brainbot-Indexes aus.

Falls innerhalb der Konfiguration im Admin-Tool angegeben wird, dass Pflegeaktionen von einem Jobclient ausgeführt werden sollen ("Jobclient benutzen"), so muss ein Jobclient gestartet werden, damit die Pfege des externen Index ausgeführt wird.

Der KBrainbotJob hat keine weiteren Konfigurationsparameter in der ini-Datei, da die gesamte Konfiguration im Admin-Tool stattfindet.

3.4.2.1.3 KExternalCommandJob

- Kategorie(n): <keine>
- Bezeichnung: Externer Aufruf

Mit Hilfe des **KExternalCommandJobs** ist es möglich ausführbare Programme, die sich mit der Abarbeitung oder Veränderung von Dateien beschäftigen oder einfach nur aufgerufen werden sollen, anzusteuern. Eine Konfiguration in der INI-Datei des JobClients ist nicht notwendig. Der Job wird durch einen Skriptaufruf eingeworfen.

Das Hauptelement des Skriptaufrufes ist das Element ***ExternalCommandJob***. Mit dem Attribut *execution* kann eingestellt werden, ob der Job lokal ohne JobClient (Wert: *local*) oder mit JobClient (Wert: *remote*) ausgeführt werden soll. Der Standardwert ist *remote*.

Anmerkung zur Remote-Ausführung:

Eine Kontrolle über den Zugriff auf lokale Programme findet über den Aufruf einer Batchdatei statt. Bevor sich der JobClient einen KExternalCommanJob zur Ausführung nimmt, über-

prüft er, ob er diesen Job ausführen kann. Das ist der Fall, wenn im aktuellen Verzeichnis des JobClients die Batchdatei vorhanden ist, die im Element *Command*angegeben ist. Wird der aktuell anstehende Job von keinem JobClient zur Bearbeitung angenommen, ist die Job-Warteschlange für den Benutzer, der den Job eingeworfen hat, blockiert. Dieser Job muss von Hand gelöscht werden.

Das notwendige, erste Unterelement im Skript:

- **Command**: gibt an welche Batchdatei aufgerufen werden soll

  ```
  <Command>convert.bat</Command>
  ```

In dem Element Command wird der Name der Batchdatei angeben. In der Batchdatei ist das Verzeichnis und das auszuführende Programm selbst angegeben. ***Wichtig:*** *Die Batchdatei muss auf der gleichen Ebene wie das Programm (z.B. JobClient oder KB) liegen. Verzeichnisangaben im Element Command werden ignoriert.*

Die weiteren Unterelemente werden von oben nach unten abgearbeitet. Falls die Reihenfolge der Parameter im externen Programm eine Rolle spielt, sollte dies berücksichtigt werden.

Skriptelemente, die die Parameter für den Aufruf bilden:

- **OptionString**: kann mehrfach verwendet werden. Es werden Parameter des aufzurufenden externen Programms als Zeichenketten angegeben. Die Parametereinträge müssen vollständig angegeben werden.
  `<OptionString>-size 100x100</OptionString>`
- **OptionPath**: der angegebene Path-Ausdruck wird ausgewertet und als Zeichenkette in den Kommandoaufruf eingebaut
  `<OptionPath path="./topic()/concept()/@$size$"/>`

Skriptelemente, die sich mit dem Handling von Attributen beschäftigen

- **SourceBlob**: Angabe des Blobattributes, das als Datenquelle verwendet wird
  `<SourceBlob><Path path="$bild$"/></SourceBlob> <SourceBlob path="$bild$"/>`
- **ResultAttribute**: Angabe der Parameter für die Erzeugung eines neuen oder die Veränderung eines bestehenden Blobattributes mit dem Inhalt der Datei bzw. der Datei selbst, die das Ergebnis des extern aufgerufenen Programms ist.
  Attributwerte:
  name: Name bzw. interner Name des anzulegenden Attributes
  topic: Zielindividuum des anzulegenden Attributes
  modifyExisting: verändern (*true*) oder neu anlegen (*false,* Standardwert)
  filename: Dateiname des anzulegenden Blobattributes

  ```
  <ResultAttribute     name="$bild2$"     topic="./topic()"     modifyExisting="true"     filen
      <Path path="$bild2$"></ResultAttribute>
  ```

Beispiel 01:

Skript:

```
<Script>     <ExternalCommandJob execution="local">         <Command>convert.bat</Command> <Op
```

Inhalt der Batchdatei unter Windows:

```
"C:\Program Files\ImageMagick-6.2.6-Q16\convert.exe" %*
exit /B %ERRORLEVEL%
```

Inhalt der Batchdatei unter Linux:

```
#!/bin/bash
convert $*
```

Beispiel 02:

Skript:

```
<Script>     <ExternalCommandJob execution="local">         <Command>convert2.bat</Command> <
```

Inhalt der Batchdatei unter Windows:

```
"C:\Program Files\ImageMagick-6.2.6-Q16\convert" -size 100x100 %1
-geometry +5+10 %2 -geometry +35+30 -composite %3
exit /B %ERRORLEVEL%
```

Inhalt der Batchdatei unter Linux:

```
#!/bin/bash

convert -size 100x100 $1 -geometry +5+10 $2 -geometry +35+30 -composite $3
```

Anmerkung: Die beiden Beispiele liefern als Ergebnis die gleiche Datei. In den Windows-Batchfiles dient der Exit-Befehl dazu, den Exit-Code von "convert" an den Aufruf zurückzuliefern.

Hier noch ein Beispiel für ein erweitertes Konvertierungsskript, welches mit den Parametern "Quelldatei", "Bildbreite" und "Zieldate" aufgerufen werden kann und welches nur breitere Bilder auf die angegebene Breite verkleinert. Das Script schreibt außerdem eine Protokolldatei über die Konvertierung wobei auch Fehlermeldungen von Image Magick in die Logdatei geschrieben werden:

```
set MONTH_YEAR=%DATE:~-8%
echo Converting %1 to %3 (width: %2) >> convert%MONTH_YEAR%.log
convert.exe %1 -resize "%~2>" %3 2>> convert%MONTH_YEAR%.log
echo Conversion finished with exit code %ERRORLEVEL% >> convert%MONTH_YEAR%.log
exit /B %ERRORLEVEL%
```

Und hier noch die Version für Linux (Bash):

```
#!/bin/bash
FULLDATE=‘date +%c‘
MONTH_YEAR=‘date +%m.%Y‘
LOGFILE="convert.$MONTH_YEAR.log"
echo "$FULLDATE: Converting $1 to $3 (width: $2)">>$LOGFILE
convert "$1" -resize "$2>" "$3" 2>>$LOGFILE
EXITCODE="$?"
echo $FULLDATE: Conversion finished with exit code $EXITCODE>>$LOGFILE
exit $EXITCODE
```

3.4.2.1.4 KExtractBlobTextJob

- Kategorie(n): <keine>

- Bezeichnung: Blob in ein Textattribut umwandeln. Aus dem Blobattribut wird mithilfe der im Admin-Tool auf dem Reiter "Indexkonfiguration -> Externer Volltextfilter" angegebenen Batch-Datei der Textinhalt extrahiert und in einem neuen Attribut des angegebenen Textattributtyps abgelegt. Weitere mögliche Parameter für den Job sind das Topic, an dem der Extrakt angelegt werden soll, sowie die Sprache des anzulegenden Attributs, in dem Fall, dass das angegebene Textattribut mehrsprachig ist. Dieser Job wird von einem Trigger eingeworfen, der so angelegt sein sollte, dass er auf Erzeugen und Modifizieren von Blobattributen reagiert. Die dabei anzugebende KSkript-Regel lautet "ExtractBlobText" und gestattet die Angabe der oben genannten Parameter.

3.4.2.1.5 KQueryJob

- Kategorie(n): query
- Bezeichnung: Suche

Dient der ausgelagerten Ausführung von einfachen und Expertensuchen auf einem Jobclient. Wird je nach Bedürfnissen der betrachteten Suche ausgestattet und ausgeführt.

3.4.2.1.6 KScriptJob

- Kategorie(n): script
- Bezeichnung: KScriptJob

Mithilfe des KScriptJobs lassen sich KSkripte aus KSkript heraus so aufrufen, dass sie auf dem Job-Client ausgeführt werden. Dabei geschieht das Erzeugen des Jobs durch die KSkript-Regel "ScriptJob", welche ausgestattet mit Script und den zu diesem Zeitpunkt errechneten Startobjekten als Ausgangspunkt den resultierenden KScriptJob in die Job-Queue einstellt. So lassen sich Arbeiten asynchron auf Job-Clients verteilen. Anwendungsbeispiele sind die Auslagerung von Tätigkeiten, die bei sequenzieller Ausführung den aufrufenden Klienten zu lange blockieren würden.

3.4.2.2 Beispiel für eine Ini-Datei

```
volume=MeinNetz
host=localhost
jobPools=query, index
cacheDir=jobcache
logfile=jobclient01.log
maxMemory=400
name=jobclient01
```

3.4.2.3 Performance-Optimierungen

Vorab laden

Die JobClients können beim Hochfahren durch die Konfiguration auswählbare Strukturen vorab laden. Durch diesen Vorgang steigt der Speicherbedarf des JobClients. Im Gegenzug kann der JobClient Jobs schneller ausführen.

In die Ini-Datei des JobClients muss der Eintrag **keepClusterIDs** angegeben werden. Mögliche Werte für diesen Eintrag sind:

- **index** - Bei den Einstellungen der zusammensteckbaren Indexern gibt es die Möglichkeit, das Häkchen bei *Jobclient soll Index in den Hauptspeicher laden* zu setzen. Für die aktivierten Indexer wird ein Teil Ihrer Indexstruktur geladen.
- **protoOfSizes** - Die Anzahl der Individuen für jedes Konzept werden bereits beim Start ermittelt.
- **accessRights** - Das Root-Objekt des Rechtesystems wird in den Speicher geladen.

Wichtig: Für den Eintrag *useProxyValueHolder* muss der Wert *false* gesetzt sein. Sonst versucht der JobClient RPCs (Anfragen, die der Mediator beantworten kann) an den Mediator abzusetzen. Der Client soll jedoch die Cluster selber laden und unter Umständen auch im Speicher behalten.

Anmerkung: Es ist ebenfalls von Vorteil, für eine Performanceverbesserung den Festplattencache für den JobClient einzuschalten.

Beispiel für die Einträge in der INI-Datei:

```
[Default]
...
useProxyValueHolder=false
keepClusterIDs=index,protoOfSizes,accessRights
cacheDir=jobcache
maxCacheSize=1000
...
```

3.5 BLOB-Service

3.5.1 Einführung

Der Blob-Service dient der Datenhaltung von großen Dateien außerhalb des Wissensnetzes, aber verknüpft mit den Datei-Attributen, in denen diese Dateiinhalte abgelegt werden sollen. Dies hat mehrere Vorteile:

- Das Wissensnetz enthält dadurch nur noch die semantischen Informationen, die auf den Dateien aufsetzen und bleibt gut sicher- und übertragbar.
- Speicherorte von Wissensnetz und Dateiinhalten können unterschiedlich konfiguriert werden.
- Es lassen sich mehrere Blob-Services an ein Wissensnetz anschließen, so dass theoretisch je Attributdefinition ein Speicherort vorgehalten werden kann.

Im folgenden Kapitel wird das Einrichten und der Betrieb von Blob-Services erläutert.

3.5.2 Konfiguration

Um festzulegen, unter welcher Netzwerk-Adresse (Host und Port) der Blobservice erreichbar sein soll, muss in der Datei "blobservice.ini" die Option "interfaces" eingetragen werden. Prinzipiell gibt es dabei zwei Möglichkeiten:

1. Der BLOB-Service soll nur von dem Rechner aus erreichbar sein, auf dem der BLOB-Service installiert ist
2. Der BLOB-Service soll über das Netzwerk auch von anderen Rechnern aus erreichbar sein.

Hier ein Konfigurationsbeispiel für Variante 1, wobei der BLOB-Service-Port (30000) auch frei wählbar ist:

```
interfaces=http://localhost:30000
```

Zur Konfiguration von Variante 2 muss man anstelle von "localhost" die IP-Adresse des Netzwerk-Adapters eintragen, über den der BLOB-Service aus dem Netzwerk ansprechbar sein soll. Möchte man, dass der BLOB-Service über alle Netzwerk-Adapter erreichbar ist, die auf dem Rechner aktiv sind, so muss man als IP-Adresse "0.0.0.0" eintragen. Beispiel:

```
interfaces=http://0.0.0.0:30000
```

Wird der BLOB-Service über das Netzwerk angesprochen, so sollte die Kommunikation verschlüsselt werden. Die verschlüsselte Kommunikation über HTTPS kann ebenfalls in der Option "interfaces" konfiguriert werden, indem "http://" durch "https://" ersetzt wird. Beispiel:

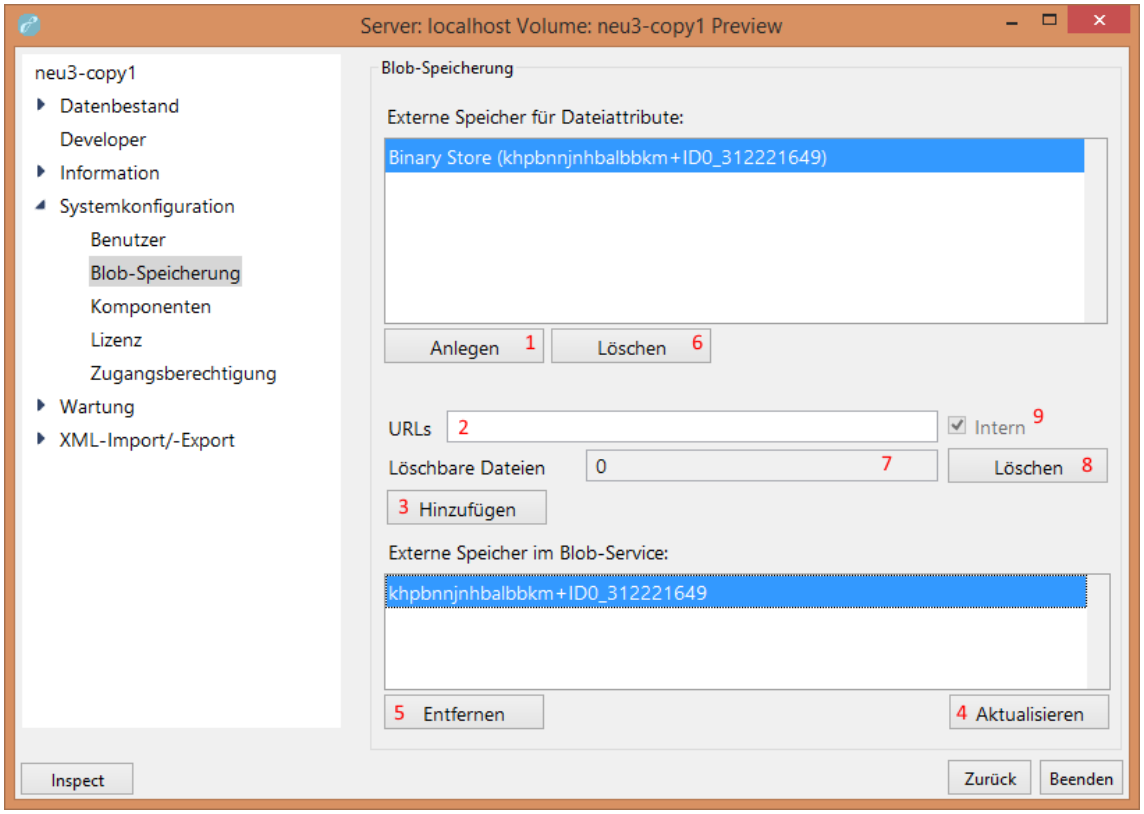
```
interfaces=https://0.0.0.0:30000
```

Für den verschlüsselten Fall siehe auch das nachfolgende Kapitel SSL Zertifikate.

Um den Betrieb zu gewährleisten, muss zusätzlich im Arbeitsverzeichnis die DLL des SQLite Frameworks "sqlite3.dll" vorhanden sein. Ohne diese DLL kann die intern benötigte Verwaltungsstruktur nicht aufgebaut und gepflegt werden.

Danach kann der Blobservice gestartet werden und steht ab sofort zur Verfügung.

Um den Blobservice mit einem Blobstore in der semantischen Graph-Datenbank zu verknüpfen,bietet das Admin-Tool unter "Systemkonfiguration -> Blob-Speicherung" die nötigen Werkzeuge:



Durch Klicken auf "Anlegen" (1) wird eine neuer logischer Store erzeugt. Danach muss in dasEingabefeld "URL" (2) die in der ini-Datei angegebene URL des Blobservices eingetragen werdenund dann auf "Hinzufügen" (3) geklickt werden. Der neu gebaute Blobstore für externe-Haltung von Dateiattributen ist danach mit dem Blobservice verknüpft, was durch Klicken auf"Aktualisieren" (4) im unteren Darstellungsbereich kontrolliert werden kann.

Im Bereich "URLs" (2) kann auch eine Liste alternativer URLs per Komma getrennt angegebenwerden. i-views bevorzugt bei alternativen URLs wenn möglich eine Verbindung überLoopback-Device.

Der Bereich "Löschbare Dateien" (7) zeigt die Zahl der aus Sicht des Wissensnetzes nichtmehr benötigten Dateien an. Mit "Löschen" (8) werden diese im Blob-Service dereferenziertund ggf. entfernt.

Der Indikator "Intern" (9) zeigt an, dass es sich um einen in einen Mediator integrierten Store-handelt. Interne Stores werden bei Volume-Transfers (upload, download, copy, backup, re-cover)automatisch mit dem Volume transferiert.

Will man die Verknüpfung eines Blobstores zu einem Blobservice aufheben, so selektiertman den gewünschten Blobstore in der Liste "Externe Speicher im Blob-Service" und klickt"Entfernen" (5). Danach kann man den Blobstore im oberen Bereich "Externe Speicher für Dateiattribute" selektieren und durch Klicken auf "Löschen" (6) ganz entfernen, oder man kann durch Angabe einer neuen URL den Blobstore mit einem anderen Blobservice wie oben beschrieben neu verknüpfen.

ACHTUNG!

Durch das Auflösen der Verknüpfung eines Blobstores zu einem Blobservice gehen alle dort gespeicherten Dateien verloren!

3.5.3 SSL Zertifikate

Zur Konfiguration der HTTPS-Verbindung müssen das Zertifikat und der Private-Key abgelegt werden.

Das Zertifikat muss unter **certificates/server.crt** liegen.

Der Private-Key muss unter **private/server.key** liegen. Es ist darauf zu achten, dass server.key als RSA-Key vorliegt, d.h. die erste Zeile der Datei muss
—–BEGIN RSA PRIVATE KEY—–
lauten. Wenn der Key in einem anderen Format vorliegt, muss er konvertiert werden. Mittels OpenSSL ist dies bspw. mittels "*openssl rsa -in input.kez -out private/server.key -outform PEM*" möglich.