Humboldt-Universität zu Berlin
Institut für Kunst- und Bildgeschichte

Transformation von Thesauri nach SKOS

– Beispiel imago-Thesaurus –

Dipl. Inform. Jörg Busse

Berlin, 21.11.2013

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang

Abkürzungen

AAT	Art and Architecture Thesaurus
API	Application Programming Interface
BOM	Byte Order Mark
CoNE	Control of Named Entities
C#	Programmiersprache "c-sharp" von Microsoft
HTML	Hypertext Markup Language
ISO-8859-1	8-Bit-Zeichenkodierung Latin-1 (Windows-1252 nutzt diesen Standard)
PDF	Portable Document Format
RDF	Resource Description Framework
SKOS	Simple Knowledge Organisation System
Tag	Bezeichner, Auszeichner z.B. <Tag></Tag>
URI	Uniform Resource Identifier
UTF-8	8-Bit-Unicode-Zeichenkodierung UCS Transformation Format
W3C	World Wide Web Consortium
XML	Extensible Markup Language

1 Motivation

Ziel ist die Migration des *imago*-Thesaurus in das Bildverwaltungssystem *imeji*[1], welches die aktuelle veraltete Bilddatenbank des Instituts für Kunst- und Bildgeschichte *imago*[2] ablösen soll.
Zugleich soll in *imeji* eine allgemeine Schnittstelle für Thesauri geschaffen werden.
Hierbei erscheint der Standard „Simple Knowledge Organization System“ (SKOS[3]) aus zwei Gründen als besonders geeignet. Zum einen, da er vom W3C als Standard zur Veröffentlichung u. a. von Thesauri, in einem maschinenlesbaren Standardformat für das semantische Web, angeboten wird und zum anderen, weil bereits verschiedene Thesauri in SKOS überführt wurden, wie z.B. der Art and Architecture Thesaurus[4] (AAT), der gerade für den kunsthistorischen Bereich von enormer Bedeutung ist.

Ein frei zugänglicher Thesaurus des Instituts für Kunst- und Bildgeschichte der Humboldt-Universität zu Berlin über das Web, fördert den wissenschaftlichen Diskurs in der Community der Kunsthistoriker. Darüber hinaus bieten Inhalt und Aufbau sowie die Verknüpfbarkeit[5] des *imago*-Thesaurus mit anderen Thesauri im SKOS-Format, Grundlagen für die interdisziplinäre Forschung.

2 Überführung des *imago*-Thesaurus

Die Überführung von Thesauri in SKOS lässt sich im großen und ganzen auf drei Schritte reduzieren[6]:

1. Analyse der Struktur des Thesaurus, der enthaltenen Terme und Relationen zwischen den Termen
2. Mapping der Elemente und Relationen des Thesaurus auf äquivalente SKOS-Elemente und -Relationen
3. technische Konvertierung des im zweiten Schritt festgelegten Mappings

---

1 http://imeji.org (21.11.2013)
2 Vgl. Reifenrath 1995
3 http://www.w3.org/2004/02/skos/ (21.11.2013)
4 Vgl. Vitzthum 2009
5 Vgl. Mayr et al. 2010
6 Strukturierte Methode nach Assem et al. 2006

2.1 Thesaurus-Analyse

Der *imago*-Thesaurus ermöglicht es, Bilder nach inhaltlichen Kategorien zu verschlagworten. Er enthielt im November 2013 etwa 12300 Terme in elf Hauptkategorien:
ca. 200 Terme zu Stil, 3000 Künstlernamen, 1000 Datierungsangaben, 1000 Terme zur Kunstgattung, 5300 Terme zur Ikonographie, 850 Standorte, 250 Terme zu Materialien, 200 Terme zur Künstlerischen Gestaltung, 200 Kunstlandschaften, 100 Terme zu angewandten Techniken und 200 Terme zu Form und Funktion.
Der *imago*-Thesaurus hat eine monohierarchische Struktur. Jedes Element kennt nur seinen direkten Vorgänger. Einzelne Elemente können alternative Bezeichner (andere Beschreibung oder Schreibweise) haben. Des weiteren ist in *imago* die Möglichkeit der Mehrsprachigkeit vorgesehen. Davon wurde bisher kein Gebrauch gemacht, wodurch der *imago*-Thesaurus komplett in deutscher Sprache vorliegt.

2.2 Mapping der Thesaurus-Strukturen auf SKOS

Das Simple Knowledge Organisation System (SKOS) ist konzeptbasiert. Das bedeutet pro Thesaurus-Eintrag muss ein SKOS-Konzept (**skos:Concept**) erzeugt werden, welches durch eine URI eindeutig identifiziert und durch weitere SKOS-Elemente näher beschrieben wird.

Der Inhalt eines Elements findet sich in SKOS unter „**skos:prefLabel**“ und seine alternativen Bezeichner unter „**skos:altLabel**“. Für die Abbildung von Hierarchien stehen die Bezeichner „**skos:broader**“ (hier Elternelement) und „**skos:narrower**“ (hier Kindelement) zur Verfügung. Assoziative Verbindungen, die quer zu hierarchischen Strukturen verlaufen können, erhalten in SKOS den Bezeichner „**skos:related**“. Für weiterführende Informationen zu einem Element, kann dieses mit Notizen (z.B. **skos:scopeNote** als Beschreibung des aktuellen Elements in seiner direkten Umgebung) versehen werden.

2.2.1 imago-Thesaurus und adäquate SKOS-Elemente

Um den *imago*-Thesaurus in SKOS abzubilden, müssen: ein eindeutiger Identifikator, der Inhalt eines Elements sowie alle alternativen Bezeichner zu einem Element aus *imago* übernommen werden.

Die Bezeichner „skos:related“ und „skos:scopeNote“ bieten eine interessante Erweiterung des *imago*-Thesaurus und werden daher in der zukünftigen Speicherumgebung CoNE[7] mit berücksichtigt.

Im *imago*-Thesaurus ist die hierarchische Struktur nur in eine Richtung abgebildet, von unten nach oben. Bei der Transformation des *imago*-Thesaurus in SKOS wird die bidirektionale Verortung eines Terms im Thesaurus-Baum durch Verweise zu Eltern- (skos:broader) und Kindelementen (skos:narrower) realisiert.

imago-Thesaurus-Element	Beschreibung / Funktion	SKOS Klasse / Poperty
Knoten im Thesaurus Baum	Wurzelknoten, innerer Knoten oder Blatt	skos:Concept
Inhalt	Schlagwort oder Phrase	skos:prefLabel
Elternelement	Direktes Elternelement	skos:broader
Kindelement	Direktes Kindelement	skos:narrower
Alternative Beschreibung	Alternative Beschreibung oder Schreibweise z.B.: Rinderschädel für Bukranie oder Photoalbum für Fotoalbum	skos:altLabel
Verwandter Term *interessante Erweiterung der Funktionalität*	Assoziierte Terme Im *imago*-Thesaurus nicht vorgesehen, aber für künftige Erweiterbarkeit sinnvoll	skos:related
Notiz *interessante Erweiterung der Funktionalität*	Notiz zum aktuellen Term Im *imago*-Thesaurus nicht vorgesehen, aber für künftige Erweiterbarkeit sinnvoll	skos:scopenote

Tabelle 1: Allgemeines Mapping

2.2.2 imago-Thesaurus Elemente

Der *imago*-Thesaurus wurde in einer relationalen Datenbank realisiert. Für die Übertragung der einzelnen *imago*-Thesaurus-Elemente auf adäquate SKOS-Elemente, müssen diese zunächst identifiziert werden.

Wichtigste Tabellen sind KeyText, mit den enthaltenen Termen und Keys, mit den enthaltenen Verlinkungen der einzelnen Thesaurus-Elemente.

[7] Service for Control of Named Entities: http://colab.mpdl.mpg.de/mediawiki/Service_for_Control_of_Named_Entities (21.11.2013)

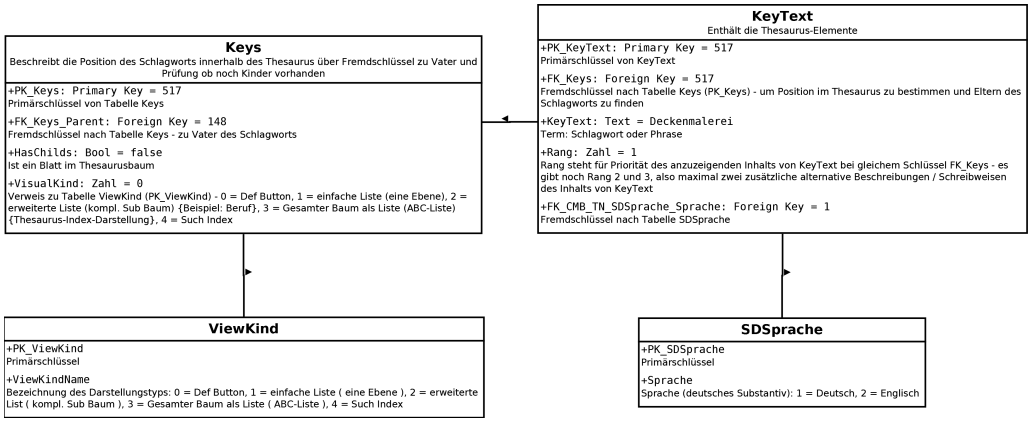

Abbildung 1: imago-Thesaurus - Tabellen in relationaler Datenbank

Des weiteren existiert ein Link zur Tabelle Sprache. Da dieser für alle Elemente auf die Sprache deutsch zeigt, kann diese Tabelle vernachlässigt werden. Hinzu kommt noch ein Link zur Tabelle „ViewKind“. Hierüber konnte in *imago* für einzelne Thesaurus-Elemente bestimmt werden, wie ihre direkten Kindelemente dargestellt werden sollen (Standardansicht, Listenansicht, Listenansicht mit Suchindex). Dieses Feld wird für *imeji* nicht benötigt und kann daher ebfs. vernachlässigt werden.

2.2.3 Identifikatoren

Die Identifikatoren werden als relative Links in der von CoNE geforderten Reihenfolge "<Vokabular>/resource/<ID>" eingefügt. Um eindeutige ID's zu erhalten, werden sie zusätzlich mit dem Prefix „imago“ versehen.

ID eines Thesaurus-Elements = FK_Keys in KeyText und PK_Keys in Keys

Da die Tabellen „KeyText“ und „Keys“ in *imago* durch den Schlüssel „FK_Keys“ (in „KeyText“) bzw. „PK_Keys“ (in „Keys“) verbunden sind, wird dieser als eindeutiger Identifikator für ein Thesaurus-Element eingesetzt.

ID in SKOS = Prefix „imago/resource/imago“ und PK_Keys

Beispiel für „Kunstgewerbe/Design“

imago_KeyText	`<FK_Keys>113</FK_Keys>`
imago_Keys	`<PK_Keys>113</PK_Keys>`
SKOS	`<skos:Concept rdf:about="imago/resource/imago113">`

ID eines Thesaurus-Elternelements = FK_Keys_Parent

ID in SKOS = Prefix „imago/resource/imago“ und FK_Keys_Parent

Das Elternelement wird in SKOS als „broader“ bezeichnet.

Beispiel für „Kunstgattung“ als Elternelement von „Kunstgewerbe/Design“

imago_Keys	`<FK_Keys_Parent>49</PK_Keys>`
SKOS	`<skos:broader rdf:resource="imago/resource/imago49" />`

ID eines Thesaurus-Kindelements

ID in SKOS = Prefix „imago/resource/imago“ und PK_Keys

Da der *imago*-Thesaurus monohierarchisch ist, wird für die Transformation nach SKOS pro Elternelement eine temporäre Liste mit allen Kindelementen benötigt. (Routine: finde in gesamter Tabelle „Keys“ alle Elemente mit „FK_Keys_Parent“ == „PK_Key“ und füge sie zu temporärer Tabelle für aktuellen Knoten hinzu)

Beispiel für „Buchdeckel“ als Kind von „Kunstgewerbe/Design“

imago_Keys	`<PK_Keys>2417</PK_Keys>` `<FK_Keys_Parent>113</FK_Keys_Parent>`
SKOS	`<skos:narrower rdf:resource="imago/resource/imago2417" />`

2.2.4 Thesaurus-Terme

Der Text der Thesaurus-Elemente ist in der Tabelle „KeyText“ im Feld „KeyText“ enthalten. Der Text wird in SKOS als prefLabel eingefügt, zugehörige alternative Beschreibungen als altLabel.

Term

prefLabel in SKOS = Text aus KeyText

Beispiel für „Kunstgewerbe/Design“

imago_KeyText	`<KeyText>Kunstgewerbe/Design</KeyText>`
SKOS	`<skos:prefLabel xml:lang="de">Kunstgewerbe/Design` `</skos:prefLabel>`

Alternative Beschreibung zu einem Term

altLabel in SKOS = Text aus KeyText

Da im *imago*-Thesaurus pro Element mehrere alternative Bezeichner möglich sind, wird pro Element eine temporäre Liste mit allen alternativen Bezeichnern benötigt.

(Routine: finde in gesamter Tabelle „KeyText“ alle weiteren Elemente mit „FK_Keys“ == „FK_Keys“ und füge sie zu temporärer Tabelle für aktuellen Knoten hinzu)

Beispiel für „Kunstgewerbe/Design“ mit drei alternativen Bezeichnern

imago_KeyText

```
<PK_KeyText>113</PK_KeyText>
<FK_Keys>113</FK_Keys>
<KeyText>Kunstgewerbe/Design</KeyText>
<Rang>1</Rang>
<PK_KeyText>10190</PK_KeyText>
<FK_Keys>113</FK_Keys>
<KeyText>Angewandte Kunst</KeyText>
<Rang>2</Rang>
<PK_KeyText>13541</PK_KeyText>
<FK_Keys>113</FK_Keys>
<KeyText>Design</KeyText>
<Rang>3</Rang>
<PK_KeyText>13544</PK_KeyText>
<FK_Keys>113</FK_Keys>
<KeyText>Gebrauchskunst</KeyText>
<Rang>4</Rang>
```

SKOS

```
<skos:prefLabel xml:lang="de">Kunstgewerbe/Design
</skos:prefLabel>
<skos:altLabel xml:lang="de">Angewandte Kunst
</skos:altLabel>
<skos:altLabel xml:lang="de">Design
</skos:altLabel>
<skos:altLabel xml:lang="de">Gebrauchskunst
</skos:altLabel>
```

Die nachfolgende Abbildung zeigt einen Auszug aus der SKOS-Repräsentation des Thesaurus-Elements "Kunstgewerbe/Design.

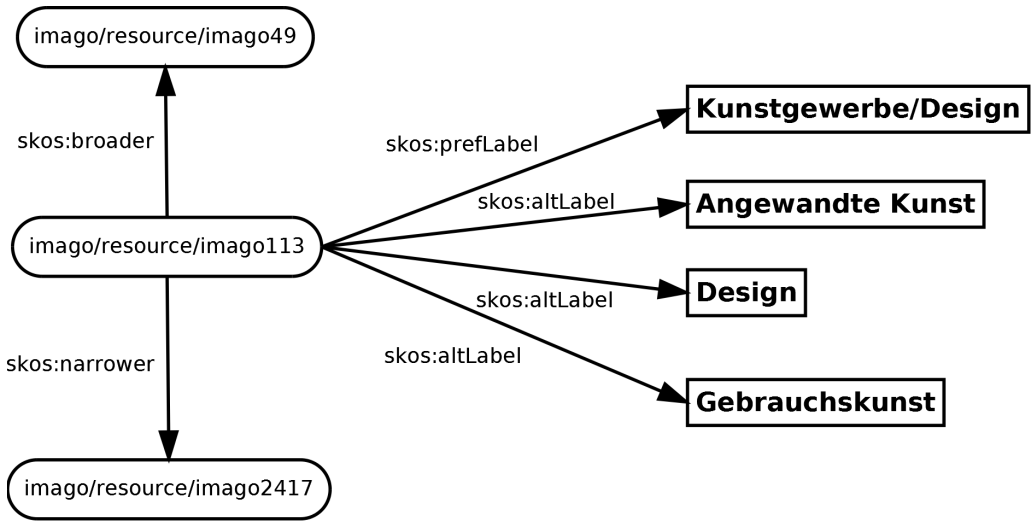

Abbildung 2: Thesaurus-Element „Kunstgewerbe/Design“ in SKOS
Enthält Link zum Elternelement, alle alternativen Bezeichner und Link zu einem Kindelement

2.3 Technische Umsetzung

Da der *imago*-Thesaurus in einer Sybase-Datenbank gespeichert ist und durch ein Microsoft Access Programm gesteuert wird, das seit 2001 nicht mehr aktualisiert wurde, kann die technische Umsetzung nicht in einem Schritt erfolgen. Es ist eine Abfolge von verschiedenen Workarounds notwendig, die sich als Best practice für die Thesaurus-Konvertierung herausgestellt haben.

2.3.1 *imago-Export durch Mitarbeiter der Mediathek*

Der Workflow ist den Mitarbeitern bekannt. Es kommt ein Java-Programm zum Einsatz, das aufgrund mangelnder Entwicklerkapazitäten nicht für den direkten Thesaurus-Export angepasst werden kann.
Ergebnis des Exports ist ein Ordner mit Textdateien im XML-Format. Jede Textdatei repräsentiert eine Tabelle in der *imago*-Datenbank, die Tags stehen für die einzelnen Spalten der jeweiligen Tabelle.

Beispiel für „Kunstgewerbe/Design“ in Tabelle KeyText

imago_KeyText

```
<KeyText>
<PK_KeyText>113</PK_KeyText>
<FK_Keys>113</FK_Keys>
<KeyText>Kunstgewerbe/Design</KeyText>
<Rang>1</Rang>
<FK_CMB_TN_SDSprache_Sprache>1</FK_CMB_TN_SDSprache_Sprache>
</KeyText>
```

2.3.2 *Vorbereitung für C#-Programm*

- Dateien Keys.txt und KeyText.txt mit Editor öffnen
  (z.B. OpenSource-Editor für Windows: Notepad++[8])
- valide XML-Deklaration einfügen:
  `<?xml version="1.0" encoding="ISO-8859-1"?>`
- Umschließenden XML-Tag einfügen
  `<AllKeys>` //an Anfang
  `</AllKeys>` //an Ende
- Alle enthaltenen „&“ durch HTML-Entität „&“ ersetzen, da sonst unerwartete Probleme beim Parsen der XML-Dateien auftreten.
  In Notepad++ durch Funktion Suchen und Ersetzen
- Beispiel:
  `<KeyText>Gropius & Schmieden</KeyText>`
  `<KeyText>Gropius & Schmieden</KeyText>`

[8] http://notepad-plus-plus.org/ (21.11.2013)

- Beim Abspeichern der Dateien im XML-Format, werden die Dateinamen zur besseren Unterscheidung mit dem Postfix des aktuellen Datums (JJJJMMTT – z.B. 20131120 für 20.11.2013) versehen: „KeyText20131120.xml“, „20131120Keys.xml“

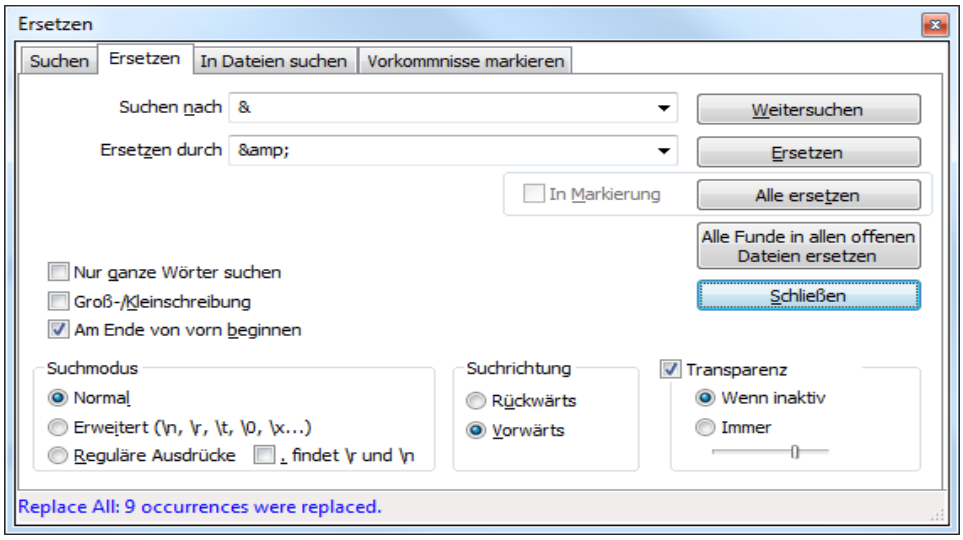

Abbildung 3: Suchen von „&“ und Ersetzen durch „&“ in Notepad++

2.3.3 C#-Programm XMLConversion mit grafischer Benutzeroberfläche

Zur Transformation des *imago*-Thesaurus in valides SKOS, werden im C#-Programm „XMLConversion.exe“ die APIs XML[9] (`using System.Xml;`) und Linq[10] (`using System.Linq;`) eingesetzt. Zur internen Verwaltung der Temporären Listen, wird die Collections[11] API (`using System.Collections.Generic;`) genutzt. Zur Darstellung der grafischen Benutzeroberfläche kommen noch weitere APIs zum Einsatz.

Im *imago*-Thesaurus enthaltene Duplikate – Elemente, die sich an der selben Position im Thesaurus-Baum mehrfach wiederholen – werden im Programm abgefangen und entfernt (Funktion AddWord in Klasse DataNode.cs). In der Status-Ausgabe der grafischen Benutzeroberfläche kann der Vorgang nachvollzogen werden.

Der komplette Programmcode von XMLConversion befindet sich im Anhang.

[9] http://msdn.microsoft.com/de-de/library/System.Xml%28v=vs.110%29.aspx (21.11.2013)
[10] http://msdn.microsoft.com/en-us/library/system.linq%28v=vs.110%29.aspx (21.11.2013)
[11] http://msdn.microsoft.com/de-de/library/System.Collections.Generic%28v=vs.110%29.aspx (21.11.2013)

Workaround zur Umwandlung

0. Starten des Programms XMLConversion.exe
1. Einfügen der XML-Datei „KeyText.xml“ bei Datafile → Load
2. Einfügen der XML-Datei „Keys.xml“ bei Indexfile → Load
3. Festlegen von Speicherort und Dateinamen des SKOS-Thesaurus bei Outputfile → Set
   Dateiname zur besseren Unterscheidung mit dem Postfix des aktuellen Datums (JJJJMMTT – z.B. 20131121 für 21.11.2013) versehen „Thesaurusimago20131121.xml“
4. Starten der Transformation über Start
5. Status der Verarbeitung und ggfs. gefundene Probleme werden im unteren Textfeld ausgegeben
6. Nach Abschluss der Transformation erscheint im rechten Textfeld der komplette Thesaurus-Baum um Veränderungen bzw. Probleme direkt nachprüfen zu können.

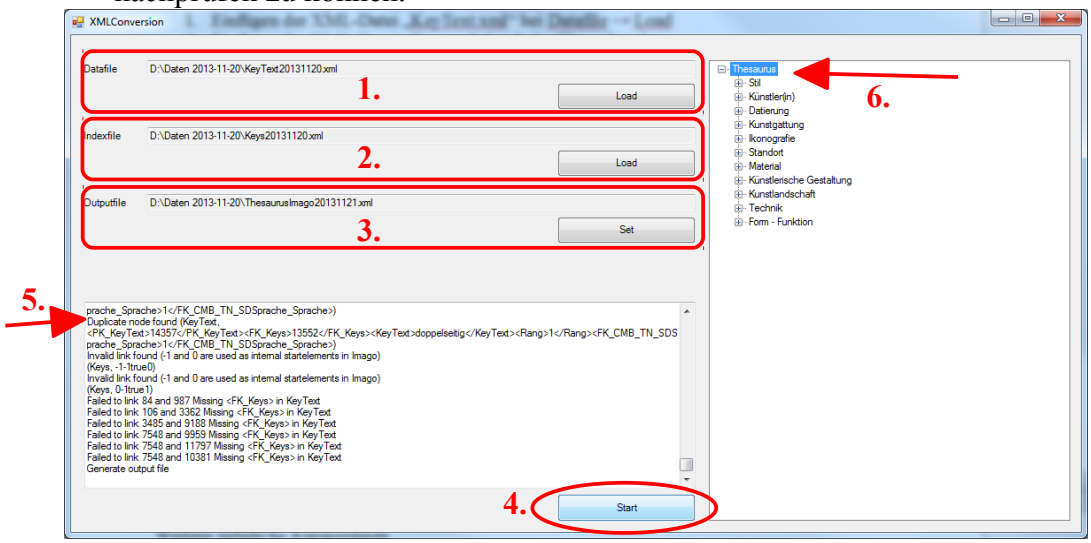

Abbildung 4: XMLConversion.exe - Benutzeroberfläche

2.3.4 Konvertierung des SKOS-Thesaurus nach UTF-8

Workaround mit Notepad++

- Thesaurus in Notepad++ öffnen
- Deklaration für UTF-8 anpassen `<?xml version="1.0" encoding="UTF-8"?>`
- Inhalt nach UTF-8 ohne Byte Order Mark (BOM – wird bei UTF-8 nicht benötigt und kann bei nicht-UTF-8-fähigen Texteditoren zu fehlerhaften Ausgaben führen) konvertieren

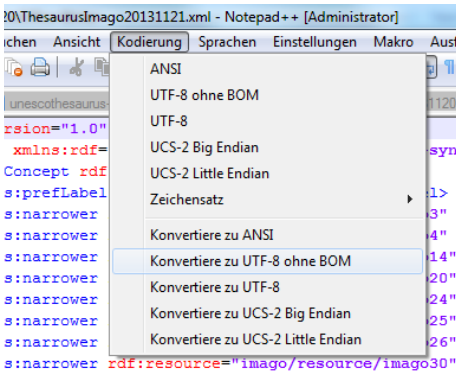

Abbildung 5: Notepad++ Konvertierung zu UTF-8 ohne BOM

2.3.5 Validierung des Thesaurus

Um sicher zu stellen, dass der so entstandene dublettenfreie Thesaurus in validem SKOS vorliegt und damit für das Semantic Web einsetzbar ist, stehen im Internet diverse Tools zur SKOS-Validierung bereit. Dazu wurde „SKOS Play“ von Thomas Francart verwendet, welches über die Webseite: http://labs.sparna.fr/skos-play/ erreichbar ist. Hier können neben lokalen SKOS-Dateien auch Webresourcen direkt abgerufen werden.

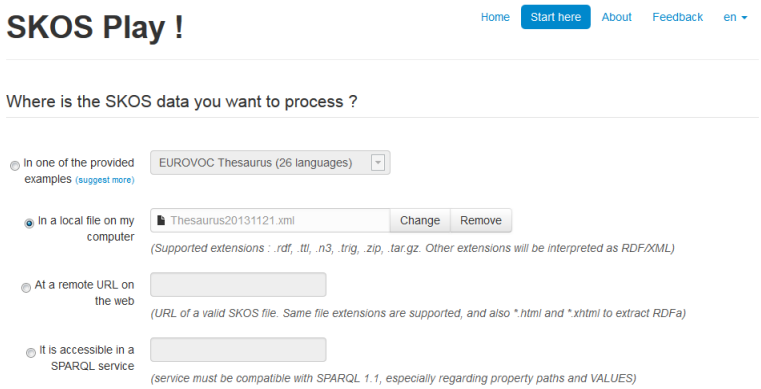

Abbildung 6: SKOS Play ! - SKOS Thesaurus einfügen

Ist die SKOS-Datei valide, so kann man sie als HTML im Browser betrachten oder direkt als PDF abspeichern. Fehlermeldungen werden mit Zeilenangaben versehen ausgegeben, einziger Nachteil ist, dass diese in französischer Sprache sind.

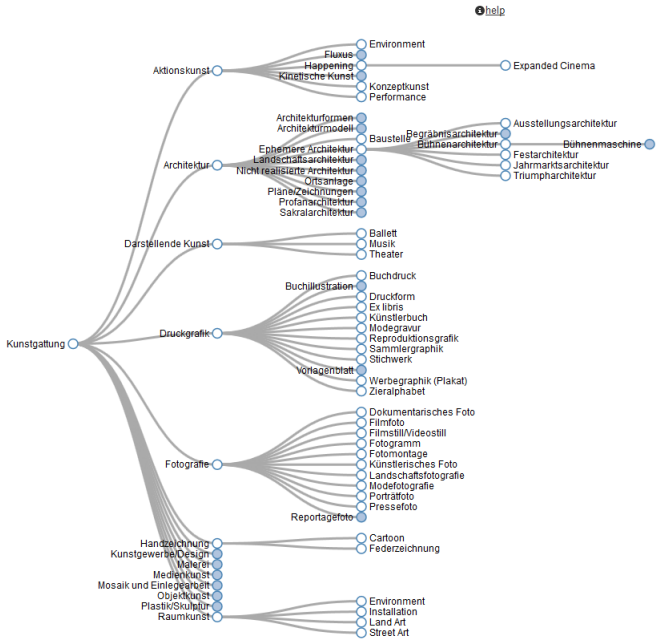

Abbildung 7: imago-Thesaurus in SKOS Play !

3 Nachträgliche Anpassungen für CoNE

- Anfang des Identifikators von imago/... in z.B. thesaurus/... umbenennen, um allgemeine Thesaurusfunktion in CoNE zu repräsentieren
- Einfügen von Dublin Core Termen mit allgemeinen Informationen zum *imago*-Thesaurus
- Alphabetische Sortierung der einzelnen Thesaurus-Ebenen (diese sind derzeit nach den *imago*-IDs sortiert)

4 Literaturverzeichnis

Assem et al. 2006
Assem, Mark van; Malaisé, Véronique; Miles, Alistair; Schreiber, Guus: A Method to Convert Thesauri to SKOS. In: The Semantic Web: Research and Applications (3rd European Semantic Web Conference, ESWC 2006), S. 95–109

Mayr et al. 2010
Mayr, Philipp; Zapilko, Benjamin; Sure, York: Ein Mehr-Thesauri-Szenario auf Basis von SKOS und Crosskonkordanzen. In: Oberhofer Kolloquium, (Magdeburg/Barleben 2010)

Reifenrath 1995
André Reifenrath: Kunstgeschichte digital. Über die Probleme einer geisteswissenschaftlichen Bilddatenbank und deren Lösung, In: humboldt-spektrum 2/1, 1995, 38–41.

Vitzthum 2009
Vitzthum, Axel: Art and Architecture Thesaurus skosified In: ATHENA WP4 SKOS Workshop (Rome, ICCU, 2009)

5 Anhang

Anhang I: XMLConversion Hauptklasse Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Xml;

namespace XmlConversion
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            if (this.openFileDialog1.ShowDialog(this) ==
DialogResult.OK)
            {
                this.textBox1.Text = this.openFileDialog1.FileName;
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            if (this.openFileDialog1.ShowDialog(this) ==
DialogResult.OK)
            {
                this.textBox2.Text = this.openFileDialog1.FileName;
            }
        }

        private void button4_Click(object sender, EventArgs e)
        {
            if (this.saveFileDialog1.ShowDialog(this) ==
DialogResult.OK)
            {
                this.textBox4.Text = this.saveFileDialog1.FileName;
            }
        }

        private void button3_Click(object sender, EventArgs e)
        {
            if (this.textBox1.Text.Length > 0 &&
this.textBox2.Text.Length > 0 && this.textBox4.Text.Length > 0)
            {
                // Dictionary of all existing nodes (accessable by
KEY)
                System.Collections.Generic.Dictionary<uint, DataNode>
nodes = new Dictionary<uint, DataNode>();
```

```
                textBox3.AppendText("Load all nodes" +
Environment.NewLine);

                // Load all nodes into a XML document (and interprete
them)
                System.Xml.XmlDocument nodesdocument = new
XmlDocument();
                nodesdocument.Load(this.textBox1.Text);
                //nodesdocument.Save(this.textBox1.Text +
".new.xml");
                if (nodesdocument.DocumentElement != null)
                {
                    foreach (System.Xml.XmlNode node in
nodesdocument.DocumentElement.ChildNodes)
                    {
                        // Interprete the current node
                        DataNode newnode =
this.GetDataNodeFromXmlNode(node);
                        if (newnode != null)
                        {
                            DataNode existingnode = null;
                            if (nodes.TryGetValue(newnode.Key, out
existingnode))
                            {
                                existingnode.AddWord(newnode.Word);
                                textBox3.AppendText("Duplicate node
found (" + node.Name + ", " + node.InnerXml + ")" +
Environment.NewLine);
                            }
                            else nodes.Add(newnode.Key, newnode);
                        }
                        else
                        {
                            textBox3.AppendText("Invalid node found
(<Rang>0</Rang> meens hidden in imago)" + Environment.NewLine + "(" +
node.Name + ", " + node.InnerXml + ")" + Environment.NewLine);
                        }
                    }
                }

                this.textBox3.Text.Insert(0, "Load all links" +
System.Environment.NewLine);

                // Load all links into the XML document (and
interprete them)
                System.Xml.XmlDocument linkdocument = new
XmlDocument();
                linkdocument.Load(this.textBox2.Text);
                if (linkdocument.DocumentElement != null)
                {
                    foreach (System.Xml.XmlNode node in
linkdocument.DocumentElement.ChildNodes)
                    {
                        // Prepare the current search
                        uint childkey = 0;
                        uint parentkey = 0;
                        bool childkeyfound = false;
                        bool parentkeyfound = false;

                        // Search for the child elements
                        foreach (System.Xml.XmlNode subnode in
node.ChildNodes)
                        {
```

```
                            if (subnode.Name == "PK_Keys" && !
childkeyfound)
                            {
                                childkeyfound = true;
                                if (!uint.TryParse(subnode.InnerText,
out childkey)) childkeyfound = false;
                            }
                            else if (subnode.Name == "FK_Keys_Parent"
&& !parentkeyfound)
                            {
                                parentkeyfound = true;
                                if (!uint.TryParse(subnode.InnerText,
out parentkey)) parentkeyfound = false;
                            }
                        }

                        // Link the elements
                        if (childkeyfound && parentkeyfound)
                        {
                            DataNode child = null;
                            DataNode parent = null;
                            nodes.TryGetValue(childkey, out child);
                            nodes.TryGetValue(parentkey, out parent);

                            if (child != null && parent != null)
                            {
                                // Link both elements
                                child.SetParentNode(parent);
                            }
                            else
                            {
                                if(parentkey >= 1)
textBox3.AppendText("Failed to link " + parentkey + " and " +
childkey + " Missing <FK_Keys> in KeyText" + Environment.NewLine);
                            }
                        }
                        else
                        {
                            textBox3.AppendText("Invalid link found
(-1 and 0 are used as internal startelements in imago)" +
Environment.NewLine + "(" + node.Name + ", " + node.InnerText + ")" +
Environment.NewLine);
                        }
                    }
                }


                textBox3.AppendText("Generate output file" +
Environment.NewLine);

                // Create the output
                XmlDocument output = new XmlDocument();
                XmlDeclaration declaration =
output.CreateXmlDeclaration("1.0", "ISO-8859-1", null);
                output.AppendChild(declaration);
                XmlElement root = output.CreateElement("rdf:RDF",
"http://www.w3.org/1999/02/22-rdf-syntax-ns#");
                root.SetAttribute("xmlns:rdf",
"http://www.w3.org/1999/02/22-rdf-syntax-ns#");
                root.SetAttribute("xmlns:skos",
"http://www.w3.org/2004/02/skos/core#");
                output.AppendChild(root);

                // Find all nodes without parent (root nodes)
```

```
                System.Collections.Generic.List<DataNode> rootnodes =
new List<DataNode>();
                foreach (DataNode node in nodes.Values) if (node.Key
!= 9053 && node.Parent == null) rootnodes.Add(node);

                // Insert all nodes into the document
                TreeNode rootnode = new TreeNode("Thesaurus");
                foreach (DataNode node in rootnodes)
this.AddNode(rootnode, node);
                this.treeView1.Nodes.Add(rootnode);

                // Insert all nodes into the document
                foreach (DataNode node in rootnodes)
this.InsertDataNode(node, root, output);

                // Save the document
                output.Save(this.textBox4.Text);
            }
        }

        private void AddNode(TreeNode parent, DataNode node)
        {
            TreeNode child = new TreeNode(node.Word);
            parent.Nodes.Add(child);
            foreach (DataNode childnode in node.Children)
AddNode(child, childnode);
        }

        private void InsertDataNode(DataNode node, System.Xml.XmlNode
xmlnode, System.Xml.XmlDocument document)
        {
            // Create the xml node and append to the parent one
            System.Xml.XmlNode mainnode =
document.CreateElement("skos:Concept",
"http://www.w3.org/2004/02/skos/core#");
            System.Xml.XmlAttribute nameattribute =
document.CreateAttribute("rdf:about", "http://www.w3.org/1999/02/22-
rdf-syntax-ns#");
            nameattribute.Value = "imago/resource/imago"+node.Key;
            mainnode.Attributes.Append(nameattribute);
            xmlnode.AppendChild(mainnode);
            System.Xml.XmlNode nextnode =
document.CreateElement("skos:prefLabel",
"http://www.w3.org/2004/02/skos/core#");
            System.Xml.XmlAttribute langattribute =
document.CreateAttribute("xml:lang");
            langattribute.Value = "de";
            nextnode.Attributes.Append(langattribute);
            nextnode.InnerText = node.Word;
            mainnode.AppendChild(nextnode);

            // Check, if the node has alternative names
            foreach (string alternativename in node.AlternativeWords)
            {
                // Create subnodes with alternative names
                System.Xml.XmlNode anode =
document.CreateElement("skos:altLabel",
"http://www.w3.org/2004/02/skos/core#");
                System.Xml.XmlAttribute langanodeattribute =
document.CreateAttribute("xml:lang");
                langanodeattribute.Value = "de";
                anode.Attributes.Append(langanodeattribute);
                anode.InnerText = alternativename;
                mainnode.AppendChild(anode);
```

```
            }

            // Check, if the node has narrower terms
            foreach (DataNode childrensnode in node.Children)
            {
                // Create subnodes with narrowers
                System.Xml.XmlNode narrownode =
document.CreateElement("skos:narrower",
"http://www.w3.org/2004/02/skos/core#");
                System.Xml.XmlAttribute narrowresourceattribute =
document.CreateAttribute("rdf:resource",
"http://www.w3.org/1999/02/22-rdf-syntax-ns#");
                narrowresourceattribute.Value =
"imago/resource/imago"+childrensnode.Key;

narrownode.Attributes.Append(narrowresourceattribute);
                mainnode.AppendChild(narrownode);
            }

            // Check, if the node has a broader term
            if (node.Parent != null)
            {
                // Create subnodes with the broader term
                System.Xml.XmlNode broadernode =
document.CreateElement("skos:broader",
"http://www.w3.org/2004/02/skos/core#");
                System.Xml.XmlAttribute broaderresourceattribute =
document.CreateAttribute("rdf:resource",
"http://www.w3.org/1999/02/22-rdf-syntax-ns#");
                broaderresourceattribute.Value =
"imago/resource/imago" + node.Parent.Key;

broadernode.Attributes.Append(broaderresourceattribute);
                mainnode.AppendChild(broadernode);
            }

            // Insert all nodes into the document
            foreach (DataNode childnode in node.Children)
this.InsertDataNode(childnode, xmlnode, document);
        }

        private DataNode GetDataNodeFromXmlNode(System.Xml.XmlNode
node)
        {
            // Prepare the interpretation
            bool keyfound = false;
            uint key = 0;
            string text = null;

            // Search all childnodes
            foreach (System.Xml.XmlNode subnode in node.ChildNodes)
            {
                if (subnode.Name == "KeyText")
                {
                    // Get the text of the node
                    text = subnode.InnerText.Trim();
                    if (text.Length == 0) text = null;
                }
                else if (subnode.Name == "FK_Keys" && !keyfound)
                {
                    keyfound = true;
                    if (!uint.TryParse(subnode.InnerText, out key))
                    {
```

```
                        textBox3.AppendText("Failed to parse: " +
subnode.InnerText + Environment.NewLine);
                        keyfound = false;
                    }
                }
            }

            // Check, if the node contained enough data
            if (text != null && keyfound) return new DataNode(key,
text);
            else return null;
        }

        private void textBox3_TextChanged(object sender, EventArgs e)
        {

        }
    }
}
```

Anhang II: XMLConversion Nebenklasse mit Konstruktoren Datanode.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace XmlConversion
{
    public class DataNode
    {

        uint key;

        string word;

        DataNode parentnode;

        System.Collections.Generic.List<string> alternativewords;

        System.Collections.Generic.List<DataNode> children;


        public DataNode(uint key, string word)
        {
            // Save all global parameters
            this.key = key;
            this.word = word;
            this.parentnode = null;
            this.alternativewords = new List<string>();
            this.children = new List<DataNode>();
        }

        public string Word
        {
            get
            {
                return this.word;
            }
        }

        public uint Key
        {
```

```
            get
            {
                return this.key;
            }
        }

        public DataNode Parent
        {
            get
            {
                return this.parentnode;
            }
        }

        public System.Collections.Generic.List<DataNode> Children
        {
            get
            {
                return this.children;
            }
        }

        public System.Collections.Generic.List<string>
AlternativeWords
        {
            get
            {
                return this.alternativewords;
            }
        }

        public void AddWord(string word)
        {
            //Kill Duplicades (equal FK_Keys AND KeyText)
            if (!this.alternativewords.Contains(word) && word !=
this.word) this.alternativewords.Add(word);
        }

        public void SetParentNode(DataNode parentnode)
        {
            this.parentnode = parentnode;
            this.parentnode.AddChildNode(this);
        }

        public void AddChildNode(DataNode node)
        {
            if (!this.children.Contains(node))
this.children.Add(node);
        }

    }
}
```