



Hochschule **RheinMain**  
Fachbereich Design Informatik Medien  
Studiengang Medieninformatik

## Abschlussarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

---

# Interaktive Analyse von Softwarekonfigurationen

---

Vorgelegt von   Benedikt Ringlein

am                   19. Oktober 2016

Referent           Prof. Dr. Wolfgang Weitz

Korreferent       Prof. Dr. Jörg Berdux

## **Erklärung gemäß ABPO**

Ich erkläre hiermit, dass ich

- die vorliegende Abschlussarbeit selbstständig angefertigt,
- keine anderen als die angegebenen Quellen benutzt,
- die wörtlich oder dem Inhalt nach aus fremden Arbeiten entnommenen Stellen, bildlichen Darstellungen und dergleichen als solche genau kenntlich gemacht und
- keine unerlaubte fremde Hilfe in Anspruch genommen habe.

Wiesbaden, 19. Oktober 2016

---

Benedikt Ringlein

## Erklärung zur Verwendung der Bachelorthesis

Hiermit erkläre ich mein Einverständnis mit den im folgenden aufgeführten Verbreitungsformen dieser Abschlussarbeit:

| Verbreitungsform   | Ja | Nein |
|--|----|------|
| Einstellung der Arbeit in die Hochschulbibliothek mit Datenträger  |    | ×    |
| Einstellung der Arbeit in die Hochschulbibliothek ohne Datenträger | ×  |      |
| Veröffentlichung des Titels der Arbeit im Internet                 | ×  |      |
| Veröffentlichung der Arbeit im Internet                            |    | ×    |

Wiesbaden, 19. Oktober 2016

\_\_\_\_\_

Benedikt Ringlein

## **Zusammenfassung**

Die komplexen Softwareaspekte auf Rechnersystemen müssen für verschiedene Zwecke erfasst und analysiert werden. Durch eine interaktive Analyse kann das Wissen einer Person für aussagekräftige Analyseergebnisse genutzt werden. Dafür ist eine Anwendung notwendig, die diesen Analysevorgang unterstützt.

Diese Arbeit bietet eine Lösung in Form einer erweiterbaren, modularen Plattform an, die eine Analyse verschiedener Aspekte der Software unter einer Programmoberfläche vereint. Diese Plattform kann an verschiedene Analysezwecke, Softwareaspekte und Datenquellen angepasst werden und unterstützt die analysierende Person durch verschiedene Ansichten, Interaktionsmöglichkeiten und Filter.

# Inhaltsverzeichnis

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Einführung</b>                          | <b>4</b>  |
| <b>2</b> | <b>Problemumfeld</b>                       | <b>6</b>  |
| 2.1      | Wichtige Begriffe . . . . .                | 6         |
| 2.1.1    | Anwendungen . . . . .                      | 6         |
| 2.1.2    | Konfiguration . . . . .                    | 7         |
| 2.1.3    | Dateien und Verzeichnisse . . . . .        | 7         |
| 2.1.4    | Abhängigkeiten . . . . .                   | 8         |
| 2.1.5    | Nutzerkonten . . . . .                     | 8         |
| 2.1.6    | Automatisch ausgeführte Aufgaben . . . . . | 8         |
| 2.1.7    | Netzwerkports . . . . .                    | 9         |
| 2.1.8    | Provisionierung . . . . .                  | 9         |
| 2.2      | Probleme . . . . .                         | 10        |
| 2.2.1    | Dokumentation . . . . .                    | 10        |
| 2.2.2    | Vergleich . . . . .                        | 10        |
| 2.2.3    | Portierung . . . . .                       | 11        |
| <b>3</b> | <b>Anforderungsanalyse</b>                 | <b>12</b> |
| <b>4</b> | <b>Lösungsansatz</b>                       | <b>14</b> |
| <b>5</b> | <b>Benutzungsoberfläche und Funktionen</b> | <b>16</b> |
| 5.1      | Funktionen . . . . .                       | 16        |
| 5.2      | Screens . . . . .                          | 17        |

|          |  |           |
|----------|--|-----------|
| 5.2.1    | Importscreen . . . . .                                   | 17        |
| 5.2.2    | Hauptscreen . . . . .                                    | 18        |
| <b>6</b> | <b>Architektur</b>                                       | <b>21</b> |
| 6.1      | Modell . . . . .   | 22        |
| 6.2      | Module . . . . .   | 24        |
| 6.2.1    | Importsubmodule . . . . .                                | 25        |
| 6.2.2    | Interaktive Submodule . . . . .                          | 25        |
| 6.2.3    | Modulaktionen . . . . .                                  | 26        |
| 6.2.4    | Filter . . . . .   | 29        |
| 6.2.5    | Exportsubmodule . . . . .                                | 29        |
| 6.2.6    | Kommunikation und Interaktion zwischen Modulen . . . . . | 30        |
| 6.2.7    | Eingebaute Module . . . . .                              | 30        |
| 6.3      | Benutzungsoberfläche . . . . .                           | 32        |
| <b>7</b> | <b>Implementierung</b>                                   | <b>34</b> |
| 7.1      | Implementierungsdetails . . . . .                        | 34        |
| 7.1.1    | Filtereffizienz . . . . .                                | 34        |
| 7.1.2    | Parameter-Annotation . . . . .                           | 35        |
| 7.1.3    | Datenimport . . . . .                                    | 36        |
| 7.1.4    | FXML Oberflächenstruktur . . . . .                       | 37        |
| 7.1.5    | Prozedurale Farbgenerierung . . . . .                    | 37        |
| 7.2      | Bibliotheken . . . . .                                   | 37        |
| 7.2.1    | Boilerplate-Code vermeiden . . . . .                     | 37        |
| 7.2.2    | Dependency Injection . . . . .                           | 38        |
| 7.2.3    | Weitere Bibliotheken . . . . .                           | 38        |
| <b>8</b> | <b>Beispielverwendung</b>                                | <b>39</b> |
| <b>9</b> | <b>Zusammenfassung und Ausblick</b>                      | <b>46</b> |
|          | <b>Anhang</b>  | <b>48</b> |

**A UML Diagramme****48**

# Kapitel 1

## Einführung

Rechnersysteme enthalten Software, die aus vielen, komplexen Aspekten bestehen kann. Unterschiedliche Ziele machen es notwendig, auf einem System vorhandene Softwareaspekte zu betrachten und zu analysieren. Zu diesen Zielen zählen die Dokumentation, der Vergleich und die Portierung von Systemen.

Ein konkretes Beispiel dafür ist ein Server, der verschiedene Anwendungen und Projekte beinhaltet, die auf andere Server (physisch oder virtuell) übertragen werden soll. Dabei sind verschiedene zusammengehörende Dateien, Pakete und weitere Daten auf unterschiedliche Systeme zu verteilen.

Wenn die Software dabei jeweils an die Umgebung angepasst werden soll, oder Änderungen der Konfigurationen oder der Austausch von Softwarekomponenten notwendig ist, wird mehr als nur eine Kopie der Software benötigt. In diesem Fall muss die auf dem Server vorhandene Softwareausstattung zunächst analysiert werden und dann Zielkonfigurationen festgelegt und angewendet werden.

Dabei muss beispielsweise auch darauf geachtet werden, dass zusammengehörende Softwarekomponenten, wie Dateien und die zur Verarbeitung notwendigen Programme, nicht getrennt werden. Eine solche Analyse manuell durchzuführen ist eine mühsame und fehleranfällige Arbeit.

Um die Informationen über ein System handhabbar zu machen, ist es notwendig, diese zu abstrahieren und auf verschiedene Arten aufzubereiten. Eine interaktive Analyse kann das Wissen einer analysierenden Person nutzen, um die Informationen in passenden Abstraktionsschichten anzubieten — beispielsweise durch Gruppierung von Elementen. Dadurch wird die Analyse vereinfacht und die Ergebnisse werden aussagekräftiger. Eine Anwendung zu entwickeln, die solch eine interaktive Analyse der Software auf einem Rechnersystem ermöglicht, ist das Ziel dieser Arbeit.



In Kapitel 2 werden zunächst wichtige Begriffe und Konzepte erklärt, die zum Verständnis des Problems erforderlich sind. In Kapitel 3 werden die Anforderungen an eine Anwendung dargestellt, die beim Lösen des Problems hilft. Der Lösungsansatz mit den grundsätzlichen Konzepten, die zur Lösung zum Einsatz kommen, wird in Kapitel 4 vorgestellt. Kapitel 5 zeigt UI-Entwürfe und Funktionen der Anwendung. Auf die Anwendungsarchitektur geht Kapitel 6 ein. In Kapitel 7 werden interessante Implementierungsdetails besprochen. Eine beispielhafte Verwendung in Kapitel 8 zeigt, wie die Anwendung in einem konkreten Fall verschiedene Aspekte der Analyse und Modellbildung unterstützen kann. Kapitel 9 fasst das Ergebnis zusammen und gibt einen Ausblick.

Zur besseren Lesbarkeit und Verständlichkeit wurden Leerzeilen, Umbrüche und deutsche Kommentare in Listings eingefügt, wodurch sie sich vom original Quelltext unterscheiden. Außerdem befinden sich einige Diagramme, um den Textfluss nicht zu stören, im Anhang.

# Kapitel 2

## Problemumfeld

Rechnersysteme werden für verschiedene Zwecke eingesetzt, beispielsweise auch für die Entwicklung und den Betrieb von Software. Die Rechnersysteme nutzen dazu selbst Software, die verwaltet werden muss. Software auf Rechnersystemen kann mit vielen Anwendungen, Konfigurationen, Nutzdaten und Abhängigkeiten sehr komplex werden. Diese Komplexität zu verringern und das System auf verschiedenen Abstraktionsebenen zu verstehen ist jedoch notwendig, um das System beurteilen, Entscheidungen treffen oder Fehler finden zu können.

### 2.1 Wichtige Begriffe

In diesem Abschnitt werden zunächst für die Problemstellung wichtige Begriffe und Konzepte erklärt. Das sind vor allem die wichtigsten zu analysierenden Softwareaspekte. Hier werden hauptsächlich Systeme mit UNIX-artigen Betriebssystemen betrachtet. Die Konzepte sind aber auch auf andere Systeme übertragbar.

#### 2.1.1 Anwendungen

Eine wichtige Art von Software sind ausführbare Programme, beziehungsweise Anwendungen und Dienste. Diese werden mit Benutzungsoberflächen oder Kommandozeilenschnittstellen gestartet und führen bestimmte Aufgaben aus.

Es gibt verschiedene Quellen und Möglichkeiten für die Installation von Anwendungen auf einem System. Eine häufig genutzte Variante ist die Verwendung von Paketmanagern wie Dpkg [10] oder RPM [26]. Dabei bestehen Anwendungen aus einem oder mehreren Paketen, die Abhängigkeiten (siehe Abschnitt 2.1.4) und Anforderungen haben können.

Ein Paketmanager kann durch Auflösen der Abhängigkeiten entscheiden, welche Pakete installiert werden müssen, damit eine zu installierende Anwendung funktioniert. Der Paketmanager kann die Softwarepakete dann aus bekannten Quellen laden und gemäß Anweisungen aus dem Paket oder der Paketmanagerkonfiguration installieren.

Manche Anwendungen können nicht über einen Paketmanager installiert werden. Der Grund dafür kann sein, dass die Anwendung in den Repositories des Paketmanagers nicht verfügbar ist oder generell auf keine Paketrepositories zugegriffen werden kann (beispielsweise als Sicherheitsmaßnahme). Diese Anwendungen müssen manuell oder über andere Automatismen installiert werden.

### 2.1.2 Konfiguration

Software kann oftmals an verschiedene Anforderungen und Umgebungen angepasst werden. Dazu stellen Anwendungen Parameter bereit, die in Konfigurationsdateien mit Werten belegt werden können. Eine Konfiguration kann das Verhalten einer Anwendung erheblich beeinflussen. Art und Format der Konfiguration unterscheidet sich oft stark von Anwendung zu Anwendung.

Konfiguration kann einerseits eine Anwendung an die Umgebung anpassen, beispielsweise indem Pfade oder Ports je nach System eingerichtet werden. Andererseits können aber auch Präferenzen konfiguriert werden, wie Formate oder Anzeigoptionen.

Unter Unix-artigen Betriebssystemen werden Konfigurationsdateien häufig im Verzeichnis `/etc` abgelegt [5, 21]. Aber Anwendungen können unterschiedliche Orte und Formate für die Ablage der Konfigurationsdateien wählen.

### 2.1.3 Dateien und Verzeichnisse

Neben der ausführbaren Anwendung und der Anwendungskonfiguration gehören auch Nutzdaten zur Software. Das sind einerseits Daten, die als Quelle für die Verarbeitung mit der Software dienen, und andererseits Daten, die von der Software erzeugt oder verarbeitet wurden. Diese Daten können von verschiedenen Programmen verwendet werden oder nur mit bestimmter Software kompatibel sein.

Solche Daten werden im Dateisystem abgelegt. Es wird dabei zwischen verschiedenen Arten von Einträgen unterschieden. Nach Dateiverwendung können Programme, Skripte, Konfigurationsdateien, Dokumente, Medien und sonstige Binärdateien unterschieden werden. Unter Unix-artigen Betriebssystemen gibt es auch noch die

Unterscheidung zwischen regulären Dateien, Verzeichnissen, symbolischen Links (Symlinks), Gerätedateien, Sockets und FIFO-Queues (Pipes) [11].

#### **2.1.4 Abhängigkeiten**

Zwischen Softwareteilen kann es verschiedene Arten von Abhängigkeiten geben. Einerseits sind das Abhängigkeiten zwischen Paketen und Softwarekomponenten. Diese entstehen, wenn eine Anwendung oder ein Paket Funktionalität oder Ressourcen aus anderen Anwendungen, Bibliotheken oder Paketen benötigt. Das Berücksichtigen der Abhängigkeiten ist für die Funktion der Software essenziell. Fehlende Abhängigkeiten haben ein Nicht-Funktionieren der Software zur Folge.

Zum Anderen können auch verschiedene Programme, Daten und Konfigurationen zusammen für einen Zweck verwendet werden. Ein Beispiel sind Webprojekte, die Server, Webanwendungen, Datenbanken und Anwendungsdaten benötigen. Einige solcher Zusammenhänge und Abhängigkeiten können automatisch ermittelt werden, während andere nur den Personen, die diese Zusammenhänge eingerichtet haben oder nutzen, bekannt sind.

#### **2.1.5 Nutzerkonten**

Für verschiedene Anwendungen werden unterschiedliche Rechte und Privilegien benötigt. Nutzerkonten bieten eine Möglichkeit, Anwendungen mit einem bestimmten dafür vorgesehenen Satz an Rechten und Privilegien auszuführen. Dafür werden Anwendungen mit einem Nutzerkonto ausgeführt, das nur die benötigten Rechte für den Zugriff auf Dateien und die Ausführung von Programmen hat.

Die Existenz und Verwendung des richtigen Benutzerkontos für die Ausführung einer Anwendung kann entscheidend für die Funktionalität der Anwendung sein. Daher ist es wichtig, neben den Anwendungen auch die Nutzerkonten und deren Rechte zu betrachten.

#### **2.1.6 Automatisch ausgeführte Aufgaben**

Vor allem auf Servern werden Programme und Dienste oft automatisch gestartet, sodass Aufgaben selbstständig durchgeführt und Dienste bereitgestellt werden können.

Zu den Auslösern für automatische Dienst- und Aufgabenstarts zählen der Start des Rechnersystems, einmalige oder regelmäßige Zeitpunkte sowie Systemereignisse. Zu wissen, wann und warum welche Dienste und Anwendungen gestartet werden,

kann sowohl für Sicherheitsaspekte als auch für das Identifizieren von Problemen wichtig sein.

Cron ist unter UNIX-artigen Betriebssystemen ein Programm zur Automatisierung von Aufgaben. Eine Aufgabe wird dabei Cronjob genannt und besteht aus einer Bedingung für den Ausführungszeitpunkt und einem auszuführenden Befehl. Cronjobs werden in einer Konfigurationsdatei hinterlegt. Die Ausführung der Aufgaben wird entsprechend dieser Konfiguration automatisch vorgenommen.

Eine andere häufig genutzte Möglichkeit zur Aufgabenautomatisierung sind Integrationsserver wie Jenkins [20]. Dabei werden ebenfalls Jobs konfiguriert, die manuell, zeitgesteuert oder durch andere Jobs ausgelöst werden können.

### **2.1.7 Netzwerkports**

Netzwerkports spielen auf einem System eine Rolle sowohl für die Lauffähigkeit von Software mit Netzwerkkommunikation, als auch für die Sicherheit des Systems.

Anwendungen können einzelne Netzwerkports belegen, an die eingehende Informationen aus dem Netzwerk weitergereicht werden. Dadurch können mehrere Anwendungen gleichzeitig über verschiedene Ports Informationen aus dem Netzwerk lesen oder mehrere Verbindungen parallel aufgebaut werden.

Für die Portbelegung von Diensten gibt es Standards, wodurch Dienste über Verwendung der richtigen Ports miteinander kommunizieren können. Ports können jedoch auch anders konfiguriert oder aus Sicherheitsgründen geschlossen werden, sodass eine Kommunikation über bestimmte Ports nicht möglich ist.

Wird die Sicherheit eines Systems überprüft, so ist es auch wichtig zu wissen, ob Ports geöffnet sind, die vom System nicht benötigt werden, da diese ein potentiell Sicherheitsrisiko darstellen können. Wird hingegen die Funktionsfähigkeit eines Systems geprüft, so ist von Interesse, ob alle benötigten Ports geöffnet und richtig konfiguriert sind.

### **2.1.8 Provisionierung**

Um Softwarezustände einfach auf Maschinen verteilen und diese verwalten zu können wird heute oftmals Provisionierung eingesetzt. Dabei wird ein Softwareinventar in Form eines Manifests erstellt, das dann automatisiert auf die Maschinen angewendet werden kann. Das hat den Vorteil, dass die Konfiguration einfach verwaltet und auch mit üblichen Versionierungstools verwendet werden kann. Zudem kann die Konfiguration mehrfach oder auf verschiedene Rechner angewendet werden.

Es gibt zahlreiche Tools für die Provisionierung, beispielsweise Puppet [24] und Chef [7].

Solche Manifeste müssen aber zunächst durch formulieren von Regeln oder Beschreiben des gewünschten Systemzustandes erstellt werden. Systeme, die manuell — also ohne Provisionierung — aufgesetzt wurden, haben noch keine Manifeste. Hier ist in der Regel eine manuelle Auswertung und Manifesterstellung notwendig.

## 2.2 Probleme

Im Zusammenhang mit der Verwaltung von Software auf Rechnersystemen treten einige Probleme auf. Eine Auswahl wichtiger Probleme, deren Lösung sich diese Arbeit widmet, wird in diesem Abschnitt dargestellt.

### 2.2.1 Dokumentation

Wenn verschiedene Personen an der Konfiguration der Software auf einem Rechner arbeiten ist es wichtig, die Bestandteile der Konfiguration sowie die Gründe für diese Zusammenstellung zu dokumentieren, damit implizites Wissen dazu nicht verloren geht und zwischen den daran arbeitenden Personen ausgetauscht werden kann.

Die Dokumentation muss separat zum System gepflegt werden. Daher kann es passieren, dass die Dokumentation nicht mehr dem tatsächlichen Stand des Systems entspricht; entweder weil sich der Stand geändert hat oder weil eine unvollständige oder inkorrekte Dokumentation angefertigt wurde.

Zudem kann es unterschiedliche Sichten auf das System geben. Daher sind teilweise verschiedene Dokumentationen mit unterschiedlichen Zielen und Schwerpunkten notwendig, beispielsweise eine Dokumentation zur Sicherheit oder zu Automatisierungsaspekten.

### 2.2.2 Vergleich

Manchmal ist es notwendig, Softwarekonfigurationen zu vergleichen — um Probleme zu finden oder zu beurteilen, ob eine Konfiguration als Ersatz für eine andere geeignet ist.

Ein Vergleich kann hierbei verschiedene Ansichtsebenen betreffen. So kann auf Projektebene verglichen werden, welche Projekte auf einem Server vorhanden sind, oder bis auf Dateiebene die Struktur verglichen werden, abhängig davon, wie genau

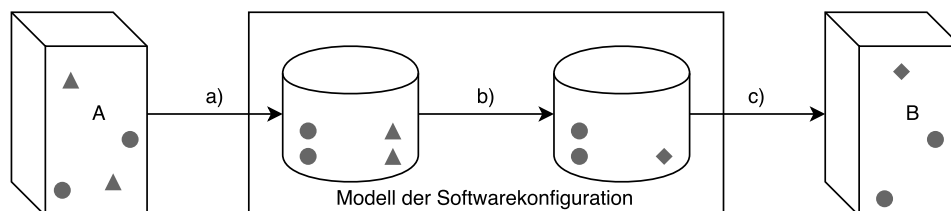
Unterschiede erkannt werden sollen und wie hoch die Toleranz für Abweichungen ist.

Dafür müssen zunächst jedoch Daten in diesen Abstraktionsebenen gesammelt werden. Es wird ein Modell benötigt. Dieses kann durch eine interaktive Analyse durch einen Nutzer erstellt werden, die diese Abstraktionsebenen kennt.

### 2.2.3 Portierung

Software auf Rechnern soll manchmal auch auf andere physische oder virtuelle Rechner umgezogen, aufgeteilt oder verändert werden. Das können Server, aber auch beispielsweise Workstations sein. Wenn Anpassungen an die Umgebung notwendig sind, wie eine andere Konfiguration von Anwendungen und Ports, oder die Softwarekonfiguration verändert werden soll, ist ein einfaches Kopieren des Systemzustandes nicht geeignet.

Stattdessen muss — sofern keine Provisionierungsskripte vorhanden sind — zunächst ermittelt werden, wie das alte System konfiguriert ist und dann bestimmt werden, wie das neue System konfiguriert sein soll. Schließlich muss die gewünschte Konfiguration auf den Rechner angewendet werden. Hilfreich für diesen in Abbildung 2.1 dargestellten Vorgang ist eine Unterstützung bei der Analyse des bisherigen Zustandes, der Konfiguration des neuen Systems und dem Anwenden der neuen Konfiguration, beispielsweise durch Generierung von Provisionierungsskripten.



- a) Analyse der Softwarekonfiguration auf Rechner A
- b) Anpassen der Softwarekonfiguration
- c) Anwenden der Softwarekonfiguration auf Rechner B

Abbildung 2.1: Portierung von Rechner A nach B mit Konfigurationsänderung

## Kapitel 3

# Anforderungsanalyse

Aus den in Abschnitt 2.2 gezeigten Problemen lassen sich eine Reihe von Anforderungen an eine Anwendung, die diese lösen soll, ableiten. Die *kursiv dargestellten* Anforderungen werden in Kapitel 5 für die Ermittlung sinnvoller Anwendungsfunktionen wieder aufgegriffen.

Die Probleme haben gemeinsam, dass zunächst eine genaue Betrachtung und Analyse des Ist-Zustands der Software auf einem System durchgeführt werden muss. Ziel einer solchen Analyse ist es einerseits, die vorhandenen Softwarekomponenten zu erfassen. Andererseits sollen diese auch gruppiert und beurteilt werden können. Die Analyse kann dabei in drei Schritten durchgeführt werden.

In einem ersten Schritt wird ein Modell des Softwarezustands eines Systems erstellt, indem Daten und Eigenschaften importiert werden. Die Analyse kann dann auf dem Modell durchgeführt werden. Das vereinfacht die Analyse bereits, da in das Modell nur die relevanten Informationen übernommen werden können und unwichtige Informationen weggelassen werden.

Der zweite Schritt ist die eigentliche Analyse, bei der Informationen gefunden, bewertet oder verknüpft werden. Hier kann ein gezieltes Suchen nach bestimmten Informationen oder ein Entdecken der verfügbaren Daten gewünscht sein. Das ursprüngliche Modell mit den rohen Daten kann während dem Analysevorgang mit weiteren bei der Analyse gefundenen Erkenntnissen verfeinert werden.

Der letzte Schritt ist ein Export der Analyseergebnisse. Je nach Analysezweck können hier unterschiedliche Ausgabeformen und -formate sinnvoll sein. Hier sind menschenlesbare Dokumente aber auch Skripte und Konfigurationen möglich.

Eine Kombination aus interaktiver und automatischer Analyse ist hier aus mehreren Gründen sinnvoll. So gibt es verschiedene Zusammenhänge und Abstraktionen (wie



Gruppen- oder Projektzugehörigkeiten), die nicht automatisch erfasst, sondern nur durch eine Person, die diese Zusammenhänge kennt, beigesteuert werden können.

Zudem können aufgrund bestimmter Befunde unterschiedliche weitere Analysen oder Aktionen notwendig sein. Diese Entscheidungen können in einer interaktiven Analyse von der analysierenden Person getroffen werden. Auch ein zeitlicher Faktor kann eine Rolle spielen, wenn nur ein Teil der möglichen Analyseschritte benötigt wird, um ein Analyseziel zu erreichen.

In Kapitel 2 wurden einige Aspekte des Problemumfeldes genannt. Die Anwendung sollte den Anwender jeweils gezielt unterstützen können. *Daher muss es möglich sein, mit Daten verschiedener Aspekte der Software zu arbeiten.* Wenn verschiedene Daten in einer Anwendung verfügbar sind, ist es für den Anwender einfacher, diese in Verbindung zu setzen und zusammen zu betrachten.

Die vorgestellten Problemfälle erfordern zum Teil eine oder mehrere abstrahierte Sichten auf ein System. Diese können von einem Nutzer durch Interaktion geliefert werden. *Dafür muss der Anwender Informationen zu Elementen oder Zusammenhängen zwischen Elementen bereitstellen können.* Durch Gruppierung von Elementen kann so eine Abstraktion dieser Elemente hergestellt werden.

Die Daten aller Softwareaspekte auf einem Rechner können sehr umfangreich werden. Schnell die relevanten Daten zu finden, ist essenziell für eine effiziente Analyse. *Eine Suche nach Informationen soll daher ermöglicht werden.*

Es ist jedoch nicht immer bekannt, welche Daten überhaupt verfügbar sind. *Ein Nutzer der Anwendung soll daher entdecken können, welche Informationen vorhanden sind.*

Es ist vor allem bei komplexen Analysen möglich, dass der interaktive Analysevorgang einige Zeit beansprucht. *Daher soll die Arbeit an der Analyse unterbrochen und später fortgesetzt werden können.* So kann auch ein Anwender dort die Arbeit fortsetzen, wo ein anderer aufgehört hat.

Dateien und Programme bilden den Grundstein für Software auf einem Rechnersystem. Daher sollen diese als erstes berücksichtigt werden. *Es müssen Informationen zu Dateien und Paketen gesammelt werden können.*

# Kapitel 4

## Lösungsansatz

Um dem Nutzer gezielt Unterstützung bei der Analyse in all den zuvor genannten Bereichen geben zu können, ist einiges an Wissen und speziellen Funktionen für die einzelnen Aspekte notwendig. Eine einfache, starre Anwendung kann eine Unterstützung so vielfältiger Daten und Nutzungsmöglichkeiten nicht bieten. Stattdessen ist eine Plattform notwendig, die durch Erweiterbarkeit an beliebige Daten und Nutzungsmöglichkeiten angepasst werden kann. Eine Plattform, die den Import der verschiedensten Daten aus einem System ermöglicht und Möglichkeiten zur Analyse, Verarbeitung und dem Export der Daten bietet.

Modularität ist für die Anwendung ein wichtiges, grundlegendes Konzept. Nur so kann die Anwendung auf eine Art erweitert werden, die eine Verarbeitung beliebiger Daten für unterschiedliche Zwecke ermöglicht. Eine Struktur für Module anzubieten ist hier also ein essenzieller Teil des Anwendungsdesigns.

Da durch den Plattformcharakter und die Anforderungen eine Vielzahl verschiedener Daten berücksichtigt werden kann, ergeben sich besondere Anforderungen an das Datenmodell. So müssen beliebige Daten in das Modell aufgenommen und abgefragt werden können. Es soll zudem mit zum Erstellungszeitpunkt der Plattform unbekannten Typen und Attributen gearbeitet werden können. Dafür ist es notwendig, dass die Art der Datenablage und die Schnittstellen zum Modell flexibel sind.

Für diesen Zweck bietet sich die Verwendung von einer allgemeinen Wissensrepräsentation für die Darstellung des Modells an. Das Resource Description Framework [16, 4] (RDF) ist ein Konzept der Wissensrepräsentation, das die Darstellung von Wissen und Zusammenhängen als eine Reihe von Aussagen ermöglicht. Dadurch muss es kein festes Schema geben, das zur Entwurfszeit bereits alle Fälle und Möglichkeiten berücksichtigt. Stattdessen kann ein Modell beliebige Objekte, Attribute und Relationen enthalten.

Aussagen werden in RDF Modellen Statements genannt. Sie bestehen aus drei Teilen. Der erste Teil ist der Uniform Resource Identifier (URI) der Ressource, auf die sich die Aussage bezieht. Darauf folgt die URI eines Prädikates, also einer Eigenschaft der Ressource. Zuletzt wird der Wert der Eigenschaft für die Ressource angegeben. Dabei kann es sich um Literale oder um Referenzen auf andere Ressourcen oder Statements handeln. Es ist auch möglich Blank Nodes (BNodes) zu verwenden. Dabei handelt es sich um Ressourcen, die keine URI besitzen und verwendet werden können, um innerhalb eines Attributes mehrere Werte im Zusammenhang abzulegen.

Da die URIs der Ressourcen und Prädikate Strings sind, können problemlos weitere Ressourcen mit beliebigen Attributen hinzugefügt werden. Es ist zum Verarbeiten dieser Informationen lediglich Kenntnis der verwendeten URIs notwendig.

Für RDF gibt es standardisierte Persistierungsformate und Abfragesprachen. Außerdem gibt es bereits Tools für die Erstellung, Betrachtung und Verarbeitung von RDF Modellen, wie Fluent Editor [12] oder die Kommandozeilentools von Apache Jena [19]. Dadurch ist RDF eine sinnvolle Wahl, um die Grundlage für die Datenanalyseplattform zu bilden.

## Kapitel 5

# Benutzungsoberfläche und Funktionen

In diesem Kapitel wird die Anwendung aus Nutzersicht betrachtet und die Funktionen und die Nutzungsoberfläche beschrieben.

### 5.1 Funktionen

In Kapitel 3 wurden Anforderungen an die Anwendung aufgestellt. Diese Anforderungen sollen mit den folgenden Funktionen erfüllt werden.

*Es muss möglich sein, mit Daten verschiedener Aspekte der Software zu arbeiten.* Das wird erreicht, indem die Arbeitsoberfläche vom Nutzer selbst aus einzelnen Aspektspezifischen Modulen zusammengesetzt werden kann. Die Module stellen Daten, Filter und Aktionen bereit.

*Der Anwender muss Informationen zu Elementen oder Zusammenhängen zwischen Elementen bereitstellen können.* Dies wird durch ein Taggingkonzept erreicht. Elemente können mit beliebigen Markierungen (Tags) versehen werden. Ein Tag kann dabei eine Eigenschaft des Elements darstellen oder zur Markierung von Gruppenzugehörigkeiten verwendet werden.

*Eine Suche nach Informationen soll ermöglicht werden.* Dazu kann der Anwender eine Vielzahl an Filtern auf die Daten anwenden, die generelle oder fachlich relevante Daten berücksichtigen können.

*Der Nutzer soll entdecken können, welche Informationen vorhanden sind.* Dies wird durch ein Modul erreicht, das alle vorhandenen Daten mit allen bekannten Eigenschaften anzeigen kann.

*Die Arbeit an der Analyse soll unterbrochen und später fortgesetzt werden können. Dafür kann das Modell, das der Anwender erstellt, abgespeichert und geladen werden.*

*Es müssen Informationen zu Dateien und Paketen gesammelt werden können. Module für Dateien und Pakete bieten entsprechende Möglichkeiten für den Import und die Verarbeitung dieser Informationen an.*

## 5.2 Screens

Die Interaktion mit der Anwendung teilt sich in zwei Bereiche auf: Einerseits müssen Informationen zur Analyse ausgewählt und importiert werden und andererseits ist eine Betrachtung und Analyse der importierten Daten durchzuführen. Daher gibt es zwei Screens, die die jeweiligen Interaktionsmöglichkeiten bündeln: Den Import- und den Hauptscreen.

### 5.2.1 Importscreen

Der Import von Daten ist notwendig, bevor mit der Betrachtung und Analyse begonnen werden kann. Daher wird dem Nutzer beim Start der Anwendung zunächst der Importscreen präsentiert.

Um Daten importieren zu können sind zunächst zwei Angaben erforderlich. Es muss einerseits eine Datenquelle angegeben werden, von der die Informationen bezogen werden sollen. Andererseits muss ausgewählt werden, welche Informationen importiert werden sollen. Für diese Angaben gibt es zwei Bereiche im Importscreen, dessen grober Aufbau in Abbildung 5.1 dargestellt ist.

Der linke Bereich enthält ein Auswahlfeld zum Selektieren der Art der Datenquelle, sowie (falls notwendig) weitere Felder zur Angabe von Parametern. Das Gestaltungsgesetz der Nähe [2] wird hier durch Gruppierung zusammengehörender Elemente umgesetzt. Für die SSH Datenquelle werden so Host und Port sowie Nutzernamen und Passwort jeweils zu einer logischen Einheit zusammengefasst. Sollten Angaben unvollständig oder fehlerhaft sein, werden dem Anwender entsprechende Fehlermeldungen präsentiert.

Der rechte Bereich enthält eine Liste der verfügbaren Importmodule. Damit kann eingestellt werden, welche Informationen importiert werden. Nicht benötigte Informationen können so ausgeschlossen werden, um den Importvorgang zu beschleunigen. Tooltips in der Liste bieten eine Beschreibung der Importmodule an.

Das Starten des Importvorganges ist die wichtigste und letzte Aktion im Importscreen. Daher gibt es einen großen Start-Button in der unteren rechten Ecke. Eine

Datenimport x

Datenquelle auswählen

Art der Datenquelle SSH ▼

Host und Port

localhost 2222

Nutzername und Passwort

vagrant \*\*\*\*\*

Importmodule auswählen

☒ Dateien und Verzeichnisse

☒ Pakete (DPKG)

☐ Paket-Dateien (DPKG)

Importiere Datei- und Verzeichnisinformationen in das interne Modell

Daten importieren

Abbildung 5.1: Mockup des Importscreens

Deaktivierung dieses Buttons nach dem Klicken signalisiert, dass der Vorgang gestartet wurde. Zudem wird links unten ein Spinner sichtbar, der durch einen runden Fortschrittsindikator ersetzt wird, sobald ein genauer Fortschritt angegeben werden kann. Neben der Fortschrittsanzeige informieren kurze Texte darüber, was gerade passiert. Erfolgreich (beziehungsweise erfolglos) beendete Importmodule werden grün (beziehungsweise rot) markiert. Erfolgreich ausgeführte Importmodule werden, sofern vorhanden, nach dem Importvorgang aufgelistet.

Da es auch die Möglichkeit gibt, Modelle zu speichern und zu laden, kann der Importscreen mit einem Button in der oberen rechten Ecke geschlossen werden. Dadurch kann der Nutzer direkt in den Hauptscreen gelangen und dort die Daten laden.

### 5.2.2 Hauptscreen

Die restliche Interaktion mit der Anwendung findet innerhalb eines Screens — des Hauptscreens — statt. Hier können die importierten Daten betrachtet, analysiert, verarbeitet und exportiert werden.

Dazu gibt es interaktive Submodule, Ansichten und Perspektiven. Interaktive Submodule stellen bestimmte Daten (beispielsweise Dateien, Verzeichnisse oder Pakete) und Aktionen auf diesen Daten bereit. Ansichten sind mit interaktiven Submodulen verknüpft und zeigen diese Daten in einer jeweils speziellen Form (als Liste, Baum oder Tabelle) an. Ansichten interaktiver Submodule können in Perspektiven grup-

piert werden, sodass zwischen verschiedenen Anordnungen und Konfigurationen von Ansichten gewechselt werden kann.

Der Screen ist, wie in Abbildung 5.2 gezeigt, vertikal in zwei Bereiche aufgeteilt. Am oberen Bildschirmrand befindet sich eine Menüleiste, die Funktionen zum Laden, Speichern und Exportieren des Modells, zum Wechseln der Perspektive und zum Hinzufügen von Ansichten interaktiver Submodule bietet und den Namen der aktuellen Perspektive anzeigt. Die Menüfunktionen sind auch über Tastaturkurzbe-  
fehle aktivierbar. Das ist besonders für Expertennutzer eine wichtige Funktion [1]. Darunter liegt der Modulbereich, in dem die einzelnen Module zur Interaktion mit den Daten platziert werden.

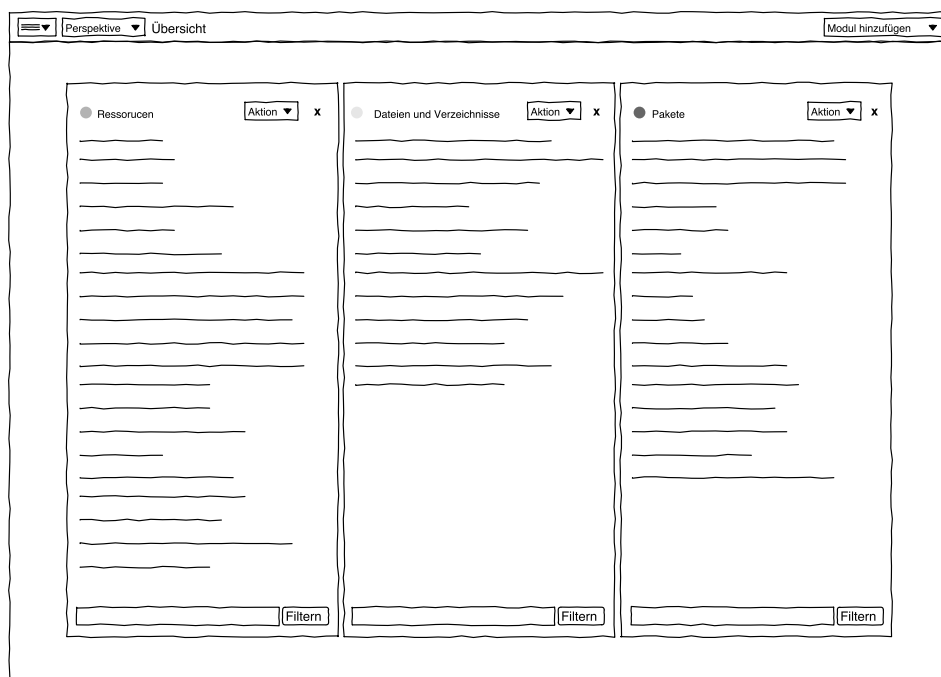


Abbildung 5.2: Mockup des Hauptscreens

Die Menüs sind mit Icons versehen, um die relevanten Einträge schnell finden zu können. Durch Klicken auf den Namen der Perspektive kann dieser direkt bearbeitet werden, sodass zur Umbenennung einer Perspektive kein Menüeintrag notwendig ist. Interaktive Submodule werden nach Modulzugehörigkeit gruppiert. Zur schnelleren Identifikation und Wiedererkennung wird jedem Submodul dabei eine Farbe zugewiesen und angezeigt. Zur Orientierung wird im Perspektivenauswahlmenü neben dem Namen der Perspektive auch die Anzahl enthaltener Submodule angezeigt.

Die generelle Erscheinung der Ansichten interaktiver Submodule ist einheitlich. Sie werden in einem Kasten dargestellt, der sich mit einem leichten Schatten gut vom

Hintergrund abhebt. Aufgebaut ist jedes Submodul dabei aus einer Titelzeile, dem Hauptbereich und einer Filterzeile.

Die Titelzeile enthält den Titel und die Farbe des Submoduls, ein Menü mit Aktionen sowie einen Button zum Schließen des Submoduls. Außerdem wird dort ein Fortschrittsindikator angezeigt, wenn in der Ansicht gerade Daten geladen werden. Der benötigte Platz eines Submoduls kann sich je nach Datenstruktur und Menge der angezeigten Daten verändern. Daher gibt es die Möglichkeit, durch Ziehen mit der Maus an den Submodulrändern dessen Größe zu verändern. Durch Ziehen des Submodultitels nach links oder rechts können zudem Submodule vertauscht und damit die Reihenfolge an die Bedürfnisse des Nutzers angepasst werden.

Der Hauptbereich unterscheidet sich je nach Art der Datenstruktur, die von dem Submodul produziert wird. Es können Listen, Bäume und Tabellen dargestellt werden. In Listen und Bäumen werden in jeder Zeile, die eine Ressource repräsentiert, die Tags der Ressource als einzelne bunte beschriftete Flächen aufgeführt.

Generell ist in den Ansichten eine Mehrfachselektion möglich. Auf die Selektion können bestimmte Aktionen aus dem Aktionsmenü (oder einem entsprechenden Kontextmenü der Einträge) angewendet werden. Da nicht alle Aktionen auf alle Arten von Einträgen angewendet werden können, wird im Aktionsmenü für jede Aktion angezeigt, wie viele Elemente von einer Durchführung der Aktion betroffen wären.

Die Filterzeile ermöglicht die Filterung der dargestellten Daten. In ein Eingabefeld kann eine Filterabfrage in der YAML-Objekt-Notation oder ein einfacher Suchbegriff eingegeben werden. Durch Verknüpfung von Filtern kann die Auswahl erweitert (oder-Verknüpfung) oder eingeschränkt (und-Verknüpfung) werden. Eine Filterabfrage, die nach Elementen filtert, die „.txt“ oder „.log“ enthalten und dem regulären Ausdruck „log[0-9]“ entsprechen sieht beispielsweise so aus:

```
und: [regex: "^log[0-9]", oder: [ .txt, .log ]]
```

Bestätigt werden kann die Filtereingabe mit der Eingabetaste oder dem neben dem Eingabefeld angezeigten Filterbutton. Über ein Kontextmenü am Filterbutton kann auch eine Liste mit verfügbaren Filtern angezeigt werden. Wird die Filterabfrage fehlerhaft eingegeben, so wird dies nach Bestätigung der Eingabe durch die Anzeige eines Symbols im Filtereingabefeld gekennzeichnet.



# Kapitel 6

## Architektur

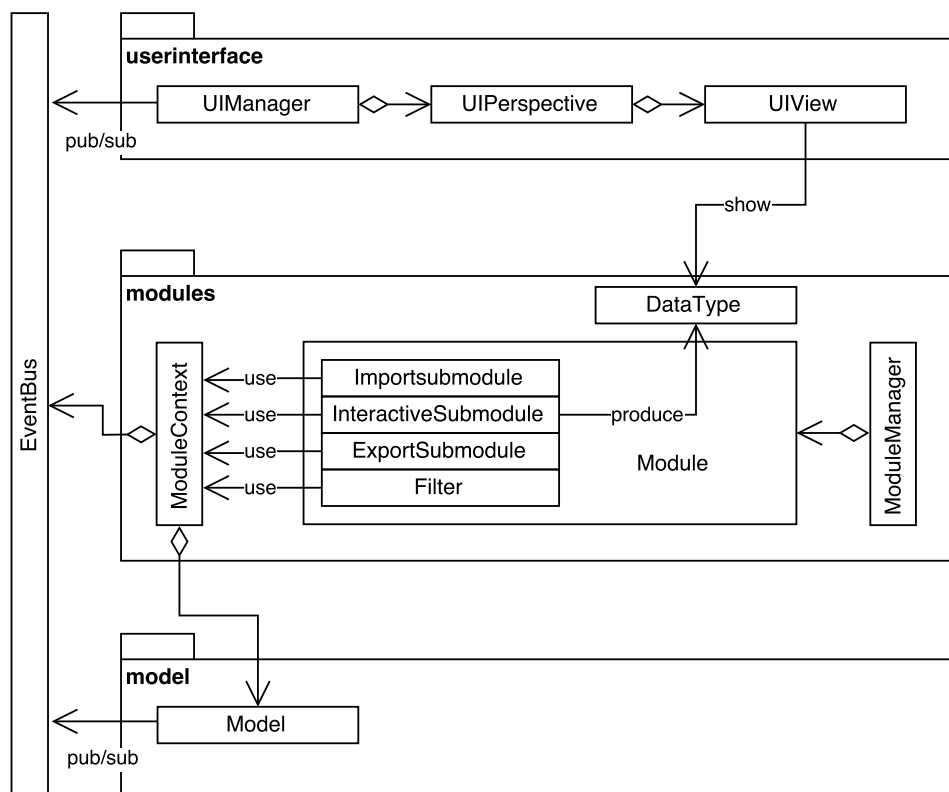


Abbildung 6.1: Die grobe Anwendungsarchitektur

Die Gesamt-Architektur besteht aus drei Komponenten (siehe Abbildung 6.1). Die Modellkomponente enthält die Verwaltung der Daten und hat keine Abhängigkeiten zu den anderen Schichten. Die Modulkomponente enthält Module, die auf die Daten zugreifen, sie importieren, exportieren oder filtern. Hier bestehen nur Abhängigkeiten zur Modellschicht. Die letzte Komponente ist die UI-Komponente, die die

Daten aus dem Modell und den Modulen anzeigt und Interaktion damit ermöglicht. Hier bestehen Abhängigkeiten zur Modul- und teilweise zur Modellkomponente.

## 6.1 Modell

Das Modell bietet die Grundlage für die Verwaltung der Daten, die vom Nutzer oder von Modulen analysiert, ausgewertet und verarbeitet werden. Dazu können Daten im Modell hinzugefügt, geändert und abgefragt werden. Neben Ressourcen- und Attributobjekten, die eine Manipulation der Daten ermöglichen, gibt es allgemeine Schnittstellen für die Interaktion mit dem auf RDF [16] basierenden Modell. Ein UML-Diagramm der relevanten Klassen und Interfaces ist in Abbildung A.3 zu sehen.

Daten werden mit Turtle-Syntax [6] (einer vom W3C empfohlenen Syntax für die Erstellung von RDF Modellen) eingefügt. Aus dem Turtle String kann ein Modell erzeugt werden, das mit dem bereits bestehenden Modell vereinigt wird, um die neuen Statements hinzuzufügen. So können beliebige Ressourcen, Eigenschaften, Referenzen und Werte mit der `insert` Methode zum Modell hinzugefügt werden.

Ressourcen im Modell werden über URIs identifiziert, die einen Namensraum (Namespace) enthalten. Im Modell wird ein mit „isa“ bezeichneter Namespace `<http://interactivesoftwareanalysis/>` für URIs verwendet. Der Namespace „rdf“ `<http://www.w3.org/1999/02/22-rdf-syntax-ns#>` wird verwendet, um den Typ einer Ressource mit dem Prädikat `rdf:type` anzugeben. Die URIs `isa:file`, `isa:directory`, `isa:symlink` und `isa:package` werden als Typ genutzt. Für die URIs von Ressourcenattributen, wird ebenfalls der Namespace „isa“ eingesetzt. Tabelle 6.1 zeigt, welche Ressourcentypen welche Prädikate haben können.

Die URIs von Ressourcen werden aus Eigenschaften der Ressourcen generiert. Bei Dateien wird dazu ein Hashwert des Pfades und bei Paketen der des Paketnamens jeweils zusammen mit einem Präfix verwendet. Der Hashwert verkürzt die URIs und sorgt dafür, dass keine unerlaubten Zeichen verwendet werden. Dass die URIs generiert werden, macht es unterschiedlichen Modulen leichter, zusammenzuarbeiten. Sie können so die URIs gemeinsam genutzter Ressourcen schnell und einfach ermitteln.

Für die Abfrage von Daten kommt SPARQL [28], eine vom W3C empfohlene Abfragesprache für RDF Modelle, zum Einsatz. Damit können beliebige Statements aus dem Modell ermittelt werden oder nach der Existenz von Statements gefragt werden (mit den `executeSelectQuery` und `executeAskQuery` Methoden).

| isa:file   | isa:directory | isa:symlink | isa:package                   |
|--|---------------|-------------|-------------------------------|
| isa:humanReadableName „String“                         |               |             |                               |
| isa:tag [isa:tagName „String“, isa:tagDetail „String“] |               |             |                               |
| isa:path „String“                                      |               |             | packageName „String“          |
|  |               |             | isa:packageFile <URI>         |
|  |               |             | isa:packageVersion „String“   |
|  |               |             | isa:packageSection „String“   |
|  |               |             | isa:packageEssential „String“ |
|  |               |             | isa:packagePriority „String“  |
|  |               |             | isa:packageDepends „String“   |

Tabelle 6.1: Prädikate der Ressourcentypen in Turtle Notation

Das Modell bietet auch Funktionen zum Laden und Speichern im RDF/XML Format an. Statt eine generelle Schnittstelle zu benutzen, um die Daten anzufragen und dann zu speichern, wenn das Modell persistiert werden soll, können so direkt die effizienten Mechanismen des Modells genutzt werden. Die Methoden heißen `load` und `save`.

Das Modell kann über einen Eventbus Änderungen am Modell bekannt geben. Es werden dann `ModelChangedEvents` gefeuert. Module können sich für diese Events registrieren und auf Änderungen reagieren. Wenn viele Änderungen auf einmal am Modell durchgeführt werden sollen, kann ein Stapelmodus (Batchmode) des Modells aktiviert werden. Dann werden nicht bei jeder einzelnen Änderung Events gefeuert, sondern nur ein Event, wenn der Batchmode wieder beendet wird.

Als Library für die RDF Modell Funktionalität kommt hier Apache Jena [19] zum Einsatz. Durch die Entkopplung des Modells über Interfaces ist auch die Verwendung anderer Bibliotheken für diesen Zweck möglich.

## 6.2 Module

Module sind die Basis für die Erweiterbarkeit und Funktionalität der Anwendung. Abbildung A.4 zeigt ein UML-Diagramm der relevanten Klassen. Ein Modul kapselt bestimmtes domänenbezogenes Wissen, wie beispielsweise Wissen über Dateien oder Pakete. Dieses Wissen kann dann genutzt werden, um dem Nutzer spezialisierte unterstützende Funktionalität anzubieten.

Dafür kann ein Modul eine Vielzahl an Submodulen beinhalten, die auf unterschiedliche Art und Weise in die Anwendung integriert werden. Dazu gehören (wie in Abbildung 6.2 dargestellt) Importmodule, Exportmodule, Interaktive Module und Filter. Neben den verschiedenen Submodulen bieten Module auch einen Namen und eine Beschreibung an, sodass Module in der Benutzungsoberfläche präsentiert werden können.

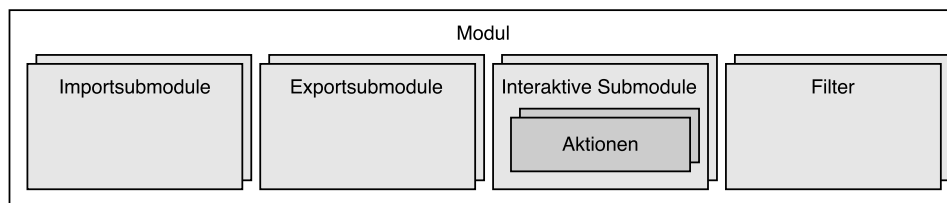


Abbildung 6.2: Aufbau von Modulen

Die Module haben Zugriff auf einen Modulkontext, der alle von Modulen benötigten Komponenten enthält. Dazu gehören das Modell für den Datenzugriff, der Eventbus für die Kommunikation und eine Datenquellenverwaltung (`DataSourceManager`) für die Datenquelle von Importmodulen.

Eine Modulverwaltung (`ModuleManager`) verwaltet alle Module zentral und kann Listen mit Modulen und bestimmten Submodularten bereitstellen. Diese erzeugt auch Modulinstanzen und gibt ihnen einen injizierten Modulkontext mit. Module und Submodule sind zustandslos. Daher reicht es aus, jedes nur ein Mal zu instanziiieren, um es dennoch mehrmals verwenden zu können. Für die Modulerzeugung kann der `ModuleManager` auf verschiedene `Moduleloader` zurückgreifen, die Module aus unterschiedlichen Quellen laden können.

Der `BuiltInModuleLoader` lädt eingebaute Module, bei denen es sich um einfache Klassen handelt, die das Interface `Module` implementieren. Der `BytecodeModuleLoader` lädt Module, die als `.class` Dateien im Verzeichnis `modules/bytecode` abgelegt werden. Die so geladenen Module sind ebenfalls Klassen, die das `Module`-Interface implementieren. Beim Kompilieren dieser Klassen muss lediglich die Jar-Datei der Anwendung selbst im Classpath liegen.

So ist die Anwendung durch neue Module erweiterbar. Und durch Hinzufügen weiterer ModuleLoader können mehr Möglichkeiten zur Erweiterung, beispielsweise durch Skriptsprachen, hinzugefügt werden.

### 6.2.1 Importsubmodule

Importsubmodule können Daten von einer Datenquelle in das Modell importieren. Ein UML-Diagramm mit den relevanten Klassen ist in Abbildung A.5 zu finden. Ein Importsubmodul kann der Datenquelle über die Methode `execute` Befehle übermitteln, die ausgeführt werden und deren Ergebnisse an das Importmodul zurückgegeben werden. Dieses wertet die Daten aus und übernimmt sie in das Modell des Modulkontextes.

Verschiedene Datenquellen können die Ausführung unterschiedlich handhaben. Beispielsweise können sie die Befehle auf dem lokalen oder einem entfernten Rechner ausführen oder Skripte generieren und deren Ausgabe zu einem späteren Zeitpunkt aus Dateien einlesen. Wie genau die Datenquelle die Befehle ausführt ist für die Importmodule irrelevant.

Importsubmodule werden nacheinander ausgeführt und bekommen ein Progress-Objekt übergeben, das sie nutzen können, um der Benutzungsoberfläche den Fortschritt des Importvorganges mitzuteilen. Dies wird in der Nutzeroberfläche verwendet, um den aktuellen Status des Imports anzuzeigen. Der Fortschritt besteht dabei aus zwei Informationen: Einerseits einem Prozentwert, der das Verhältnis von bereits verarbeiteten Elementen (beispielsweise Zeilen der Befehlsausgabe) zu den insgesamt zu verarbeitenden Elementen angibt. Und andererseits aus einer Zeichenfolge, die aktuell ausgeführte Aktionen beschreibt.

### 6.2.2 Interaktive Submodule

Interaktive Submodule können bestimmte Daten aus dem Modell abfragen und dem Nutzer bereitstellen. Dazu passende Aktionen können ebenfalls angeboten werden, sodass eine Interaktion mit den Daten möglich ist.

Ein interaktives Submodul interagiert dabei nicht direkt mit der Benutzungsoberfläche. Es erzeugt nur auf Anfrage eine Datenstruktur (DataType), die dann in der Benutzungsoberfläche zur Anzeige verwendet werden kann. Ein Modul kann angeben, welche DataTypes es unterstützt. Ein UML-Diagramm der DataTypes wird in Abbildung A.6 gezeigt.

Es gibt die DataTypes `DataList`, `DataTree` und `DataTable`, die jeweils in unterschiedlichen Darstellungen in der Benutzungsoberfläche resultieren. Alle DataTypes

enthalten einzelne Elemente (`DataItems`). Diese Elemente enthalten einen oder mehrere darstellbare Strings und optional eine durch den Eintrag dargestellte Ressource sowie deren Tags. So kann das Submodul Ressourcen oder andere Daten bereitstellen. Die Angabe der Ressource ist auch wichtig, um Modulaktionen auf den Ressourcen ausführen zu können.

Die Angabe der unterstützten `DataTypes` wird verwendet, um dem Nutzer eine Auswahlmöglichkeit für die Darstellungsart der Daten eines interaktiven Submoduls zu geben. Dieser Auswahl entsprechend wird dann ein bestimmter `DataType` beim interaktiven Submodul angefordert. Dieses kann die entsprechende Datenstruktur dann effizient erstellen. Für eine Liste müssen beispielsweise weniger Daten geladen werden, als für eine Tabelle mit detaillierten Informationen.

Interaktive Submodule sind zustandslos, da sie nur Modulaktionen und Funktionalität zur Erzeugung der Datenstrukturen enthalten. Jedes wird nur einmal instanziiert. Wird ein Modul vom Nutzer mehrmals hinzugefügt, so verwenden die Modulansichten das selbe interaktive Submodul, selbst wenn — wie in Abbildung 6.3 gezeigt — unterschiedliche Darstellungsformen benutzt werden. Die Ansichten können dennoch unabhängig voneinander dargestellt und gefiltert werden, da die Art der Datenstruktur von der Ansicht bestimmt wird und die Filterung auf die Datenstruktur angewendet wird.

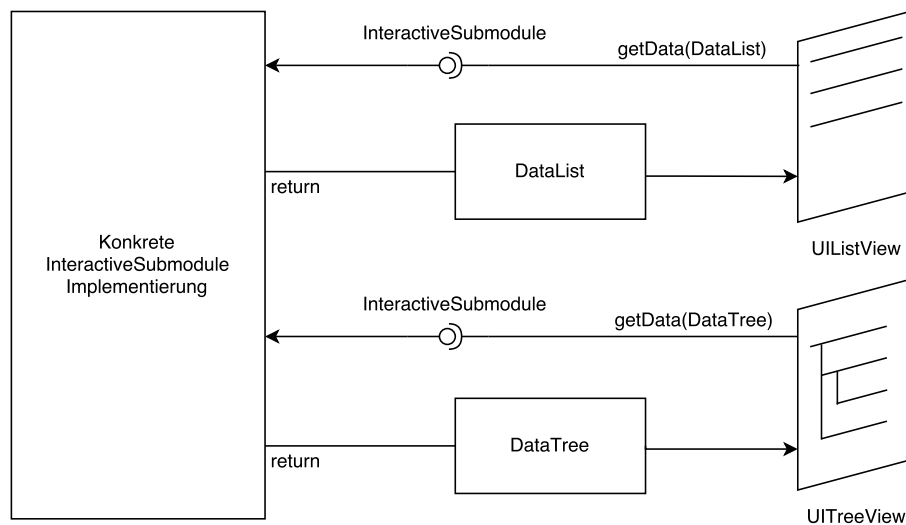


Abbildung 6.3: Ein Modul mit verschiedenen Darstellungsformen

### 6.2.3 Modulaktionen

Modulaktionen werden von interaktiven Submodulen zur Verfügung gestellt und stellen selbstständige Aktionen dar, die auf die Daten in der Moduldatenstruktur

oder im Modell angewendet werden können. Abbildung A.7 stellt alle relevanten Klassen in einem UML-Diagramm dar.

Es werden — wie in Abbildung 6.4 dargestellt — drei Typen von Modulaktionen unterschieden: Erstens Aktionen, die keine Auswahl benötigen und auf das gesamte Modell angewendet werden oder selbst entscheiden, welche Daten modifiziert werden sollen. Zweitens Aktionen, die auf eine beliebige Selektion von Elementen angewendet werden. Und letztlich Aktionen, die auf eine Selektion von Ressourcen angewendet werden.

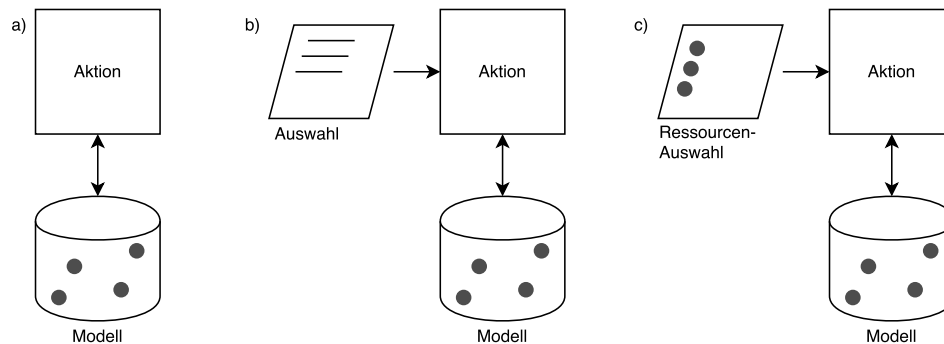


Abbildung 6.4: Aktionstypen: a) ohne Selektion, b) mit beliebiger Selektion, c) mit Ressourcenselektion

Die Aktionsarten haben gemeinsam, dass sie ausführbar sind. Sie unterscheiden sich aber darin, welche Elemente übergeben werden und wann sie ausgeführt werden können. Um Kaskaden von `if`, `else` und `instanceof` zu vermeiden, wird hier das Visitor Pattern [3] eingesetzt. Die `visit` Methode einer Aktion bekommt einen `ModuleActionVisitor` übergeben. Die Aktion (beziehungsweise die Basisklasse des Aktionstyps) entscheidet dabei, welche Methode auf dem Visitor aufgerufen wird. Der Visitor kann die Aktionen dann typsicher übergeben bekommen und die entsprechende `execute` Methode der Aktion mit passenden Parametern aufrufen.

Ein weiterer Vorteil des Visitor Patterns ist hier, dass verschiedene Visitor Objekte verwendet werden können, um unterschiedliche Operationen auf den Modulaktionen durchzuführen. So kann ein Visitor ein Aktionsauswahlmenü erstellen, das sich abhängig vom Aktionstyp anpasst; und ein anderer kann entscheiden, ob eine Aktion ausführbar ist und sie mit den passenden Elementen ausführen.

Es gibt auch Aktionen, die vom Benutzer zu setzende Parameter zur Ausführung benötigen. Dafür können Modulaktionen Parameterobjekte verwenden. Verschiedene Arten von Parametern, wie Strings oder Dateien sind über entsprechende Parameterklassen nutzbar. Die Parameterobjekte ermöglichen es, die Parametrisierung von Aktionen flexibel zu gestalten. Zudem können den Parametern weitere semantische Informationen wie ein Name, eine Beschreibung und ein Standardwert gegeben wer-

den. Diese Informationen können dann genutzt werden, um eine aussagekräftige, sinnvolle Nutzungsoberfläche für die Eingabe der Parameter zu generieren.

Dazu wird auch hier das Visitor Pattern eingesetzt. Die Modulaktion wird dabei von einem ParameterVisitor besucht und kann diesen auf alle Parameter anwenden. Die Aktion entscheidet so, welche Parameter relevant sind und die Parameter entscheiden, wie sie den Visitor aufrufen. Der Visitor für die UI Generierung entscheidet schließlich, wie ein Parameter dargestellt und eingegeben werden soll.

Jedes interaktive Submodul kann eigene Aktionen bereitstellen. Es gibt jedoch auch Standardaktionen, die in allen interaktiven Submodulen verfügbar sind. Diese werden in Tabelle 6.2 aufgeführt.

| Name, Klasse  | Typ | Parameter  |
|---|-----|--|
| <b>Alle Tags entfernen</b><br>RemoveAllTagsModuleAction | R   |  |
| <b>Ausblenden</b><br>HideModuleAction                   | R   |  |
| <b>Mit Tag versehen</b><br>TagModuleAction              | R   | StringParameter tagDetail<br>StringParameter tagName |
| <b>Text kopieren</b><br>CopyStringModuleAction          | b   |  |
| <b>URI kopieren</b><br>CopyURIModuleAction              | R   |  |

b = beliebige Selektion, R = Ressourcenselektion

Tabelle 6.2: Standard Modulaktionen

Die Aktion „Alle Tags entfernen“ entfernt alle Tags von selektierten Ressourcen, indem die entsprechenden Statements aus dem Modell entfernt werden. „Ausblenden“ versteckt selektierte Ressourcen, indem es sie im Modell mit dem Tag „ausgeblendet“ markiert. Dieser Tag wird standardmäßig in allen Ansichten herausgefiltert. „Mit Tag versehen“ ist eine parametrisierte Aktion, die ausgewählte Ressourcen im Modell mit einem oder mehreren Tags markiert. Die Aktion „Text kopieren“ kann auf alle Selektionen angewendet werden und kopiert den angezeigten Text in die Systemzwischenablage. „URI kopieren“ kopiert die URIs der ausgewählten Ressourcen in die Systemzwischenablage.



### 6.2.4 Filter

Filter werden benutzt, um Einträge in den Datenstrukturen der Ansichten zu filtern. Ein Filter ist dabei relativ simpel aufgebaut. Er bekommt einen Dateneintrag in die `filter` Methode hereingereicht und entscheidet, ob dieser beibehalten oder herausgefiltert werden soll. Abbildung A.8 zeigt ein UML-Diagramm mit den relevanten Klassen.

Es gibt zwei Arten von Filtern. Entscheidungsfiler entscheiden, ob ein Dateneintrag gefiltert wird. Kombinationsfilter kombinieren das Ergebnis von Entscheidungsfiltern mit einem booleschen Operator. Entscheidungs- und Kombinationsfilter können kombiniert werden, um beliebige komplexe, geschachtelte Filter zu erzeugen. Daher reicht es aus, nur einen Filter auf eine Datenstruktur anwenden zu können.

Filterabfragen werden vom Nutzer in ein Eingabefeld der UIView eingegeben. Ein FilterBuilder parst die Abfrage in der `buildFilter` Methode und baut die entsprechende Filterstruktur auf. Der Filter wird dann auf alle Elemente der in der UIView angezeigten Datenstruktur angewendet. Abbildung A.1 veranschaulicht diesen Filtervorgang in einem Sequenzdiagramm.

Zum Erzeugen der Filterobjekte nutzt der FilterBuilder dabei hereingereichte Factories, die vom ModuleManager bereitgestellt werden. Der YAMLFilterBuilder ist eine konkrete Implementierung, die die Abfrage als YAML-String [31] entgegennimmt. YAML ist hierfür gut geeignet, da es standardisiert ist, Parser Bibliotheken dafür existieren und die notwendigen Sprachkonstrukte (Schlüssel/Wert-Paare, Schachtelung, Aufzählungen) damit möglich sind.

Filter werden — wie Submodule — von Modulen bereitgestellt, sodass spezielle Filter hinzugefügt werden können. In mitgelieferten Modulen enthaltene Filter sind in Abschnitt 6.2.7 aufgeführt. Um in Abfragen verwendet werden zu können, haben Filter einen Namen. Zur Unterstützung des Nutzers liefert jeder Filter auch eine Beschreibung mit, sodass passende Filter ausgewählt werden können.

### 6.2.5 Exportsubmodule

Die Daten aus dem Modell können auch exportiert werden, beispielsweise, um in anderen Anwendungen damit weiterzuarbeiten. Diese Funktionalität wird von Exportsubmodulen bereitgestellt. Diese können auf das Modell zugreifen und beliebige zu exportierende Daten abfragen. Diese Daten können dann auf beliebige Weise aufbereitet oder transformiert werden. So können beispielsweise Exportmodule zum Generieren von Dokumentationen oder Provisionierungsskripten erstellt werden.

Exportsubmodule können auch vom Nutzer setzbare Parameter erhalten. Das können Einstellungen für das Ausgabeformat sein, aber auch Dateien die gelesen oder geschrieben werden sollen. Es kommen hier die gleichen Parameter zum Einsatz, wie bei den Modulaktionen.

### 6.2.6 Kommunikation und Interaktion zwischen Modulen

Zur Kommunikation zwischen Modulen gibt es, wie in Abbildung 6.5 gezeigt, zwei Möglichkeiten. Einerseits kann das Modell geschickt verwendet werden und andererseits ist eine Kommunikation über einen Eventbus möglich.

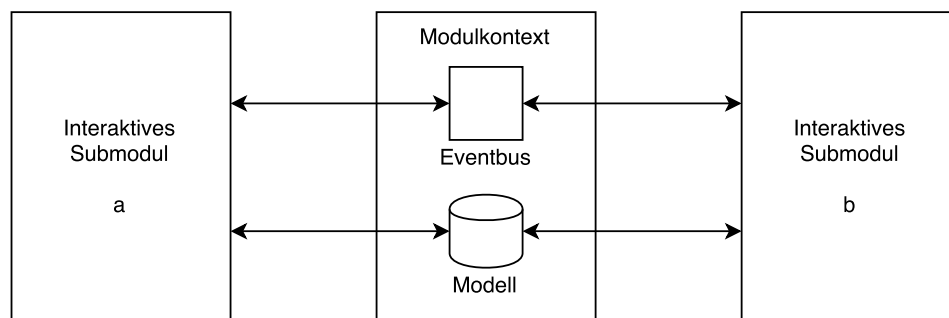


Abbildung 6.5: Kommunikation zwischen interaktiven Submodulen

Über das Modell kann eine indirekte Kommunikation erfolgen, indem Module Daten unter Verwendung von Namenskonventionen so in dem Modell ablegen, dass andere Module diese Daten abrufen und selbst verwenden können. So können Module, deren Kompetenzen sich überschneiden, zusammenarbeiten.

Jedes Modul kann sich über den Modulkontext auch an den Eventbus hängen um so spezielle Events zu feuern oder auf Events anderer Module reagieren. Hierbei können beliebige Eventobjekte verwendet werden. Voraussetzung dafür ist nur, dass der Eventtyp allen Modulen, die es verwenden möchten, zur Compile-Zeit bekannt sein muss.

### 6.2.7 Eingebaute Module

Die Anwendung ist durch die Modularität sehr auf die Erweiterbarkeit der Anwendung ausgelegt. Es werden jedoch auch drei Module mitgeliefert.

Das Modul *Allgemein* ist für allgemeine, nicht fachlich spezifische Funktionen zuständig. Es stellt die interaktiven Submodule *Ressourcen* und *Getaggte Ressourcen* bereit. Diese geben dem Nutzer Zugang zu allen im Modell enthaltenen Ressourcen,

beziehungsweise allen mit Tags versehenen Ressourcen. Außerdem werden die in Tabelle 6.3 aufgeführten Filter von diesem Modul bereitgestellt.

| Filter | Filterungsergebnis  |
|--------|---|
| name   | Einträge, deren Namen den Parameter enthalten                         |
| regex  | Einträge, deren Namen dem regulären Ausdruck im Parameter entsprechen |
| sparql | Ressourcen, für welche die SPARQL ASK-Abfrage im Parameter wahr ist   |
| tag    | Ressourcen, die mit dem exakten Parametertext als Tag markiert sind   |
| text   | Einträge, von denen mindestens einer der Texte den Parameter enthält  |
| uri    | Die Ressource, deren URI exakt dem Parameter entspricht               |

Tabelle 6.3: Filter im Modul *Allgemein*

Das Modul *Dateisystem* stellt fachlich spezifische Funktionen zum Dateisystem bereit. Es enthält das Importmodul *Dateien und Verzeichnisse* und zwei interaktive Submodule *Dateien und Verzeichnisse* und *Symlinks*. Damit werden Pfade von Dateien, Verzeichnissen und Symlinks importiert und dem Nutzer zur Interaktion bereit gestellt.

Das Modul *Pakete* stellt fachlich spezifische Funktionen für Anwendungspakete über den Debian Paketmanager [10] (dpkg) bereit. Es enthält zwei Importmodule *Pakete (DPKG)* und *Paket-Dateien (DPKG)*, die einige Informationen zu installierten Paketen und zugehörigen Dateien importieren. Außerdem ist das interaktive Submodul *Pakete* enthalten, das dem Nutzer die Paketinformationen interaktiv zur Verfügung stellt. Dieses interaktive Submodul stellt auch die Aktion *Zugehörige Dateien mit Tag versehen* bereit, die zu Paketen gehörende Dateien im Modell mit einem dem Paketnamen entsprechenden Tag versieht. Das Exportmodul *DPKG Paketselektionsliste* exportiert die Namen installierter Pakete in eine von Dpkg lesbare Liste. Eine Filterung der Pakete nach Tags ist dabei möglich. Das Modul fügt auch den Filter *dpkg* hinzu, der Dateien nach Zugehörigkeit zu einem Paket und Pakete nach Zugehörigkeit zu einer Datei filtern kann.

## 6.3 Benutzungsoberfläche

Die beiden Screens der Anwendung werden hauptsächlich durch die Controller `MainWindowController` und `ImportScreenController` gesteuert. Ein UML-Diagramm mit den relevanten Klassen ist in Abbildung A.9 dargestellt.

Der Importscreen ist Teil des Hauptfensters und enthält nur die Datenquellen- und die Importsubmodulauswahl und koordiniert die Ausführung der Importsubmodule. Die Oberfläche für die Datenquellenparameter wird nicht automatisch generiert, sondern pro Datenquellentyp fest definiert, um jeweils ein passendes, klares Layout nach gestalterischen Aspekten bereitstellen zu können.

Im Hauptscreen gibt es statische und dynamische Anteile. Der grundsätzliche Screenaufbau ist statisch. Allerdings kann der Nutzer die Oberfläche verändern und an die Bedürfnisse der Tätigkeit anpassen. Dazu können Ansichten interaktiver Submodule hinzugefügt, entfernt und vertauscht sowie in ihrer Größe verändert werden.

Die Ansichten (`UIView`) der interaktiven Submodule werden in Gruppen, den Perspektiven (`UIPerspective`) verwaltet, die vom Nutzer angelegt werden können. Es ist immer genau eine Perspektive aktiv, deren Ansichten dargestellt werden. Wenn der Nutzer ein interaktives Submodul hinzufügt, wird die entsprechende Ansicht der aktuellen Perspektive hinzugefügt.

Eine Ansichtsinstanz zeigt eine Datenstruktur (`DataType`) von einem interaktiven Submodul an. Ein Ansichtstyp kann dabei mit genau einem `DataType` umgehen. Eine Listenansicht kann beispielsweise nur Listendatenstrukturen anzeigen. Wird die Art der Ansicht vom Nutzer geändert, so wird eine neue Ansicht erzeugt, die die alte ersetzt und das selbe interaktive Submodul erhält. Es wird dann eine neue Datenstruktur angefordert, da für die neue Darstellungsform möglicherweise mehr Informationen benötigt werden.

Für die drei `DataTypes` `DataList`, `DataTree` und `DataTable` gibt es jeweils eine Ansicht: `UICollection`, `UITreeView` und `UITableView`. Es spricht jedoch nichts dagegen, pro `DataType` auch mehrere Ansichten anzubieten. In diesem Fall werden dem Nutzer alle passenden Ansichten zur Auswahl angeboten.

Die Benutzungsoberfläche wird vom `UIManager` verwaltet. Hier wird das Hauptanwendungsfenster erzeugt und Möglichkeiten für die Darstellung von Dialogfenstern bereitgestellt. Zudem werden Perspektiven dort verwaltet. Es kann auch abgefragt werden, welche Ansichten für ein interaktives Submodul in Frage kommen, basierend auf den unterstützten `DataTypes`.

Wird ein neues Submodul hinzugefügt, ermittelt der `UIManager` auch, welche Ansicht dafür standardmäßig erzeugt werden soll. Dazu geben interaktive Submodule ihre unterstützten `DataTypes` nach Priorität sortiert an — der erste `DataType` wird dann ausgewählt. Abbildung A.2 zeigt, wie das Hinzufügen einer Ansicht zu einer Perspektive abläuft. Der Nutzer wählt ein interaktives Submodul aus und löst damit einen `EventHandler` aus. Dieser lässt vom `UIManager` eine passende View erzeugen und fügt diese der aktuellen Perspektive hinzu. Der `UIManager` erzeugt die View, indem er über die unterstützten `DataTypes` eine passende `UIViewFactory` ermittelt und zur Instanziierung verwendet.

Der `UIManager` ist auch am Eventbus angemeldet und kann Perspektiven über Änderungen informieren. So kann eine Aktualisierung der Ansichten nach Veränderungsevents erfolgen.

# Kapitel 7

## Implementierung

In diesem Kapitel werden interessante Implementierungsdetails und verwendete Bibliotheken gezeigt. Umgesetzt wurde die Applikation als JavaFX Anwendung mit Java 8.

### 7.1 Implementierungsdetails

Hier werden zunächst Details zu einigen Aspekten der Implementierung gezeigt.

#### 7.1.1 Filtereffizienz

Bei der Filterung wurden Maßnahmen zur Effizienzsteigerung umgesetzt. Es werden möglichst viele Einträge parallel gefiltert, durch Verwendung paralleler Streams (siehe Listing 7.1).

```
public List<DataItem> getFilteredList(Filter filter) {  
    if (filter != null) {  
        return getList().stream()  
            .parallel()  
            .filter(filter::filter)  
            .collect(Collectors.toList());  
    } else {  
        return getList();  
    }  
}
```

Listing 7.1: Filter werden parallel auf mehrere Elemente angewendet

Angewendet werden Filter auf die Datenstrukturen, die von den Modulen erzeugt werden. Die Datenstrukturen behalten dabei eine Kopie der originalen, ungefilterten Daten. So können diese bei einer Änderung des Filters einfach nochmals gefiltert werden, statt sie noch einmal neu aus dem Modell erzeugen zu müssen. Das sorgt für eine schnelle Filterung.

Außerdem ist der Und-Filter als short-circuiting Variante implementiert; er bricht ab, sobald das Ergebnis feststeht (siehe Listing 7.2). Dafür wird ausgenutzt, dass die Stream-Methode `anyMatch` zurückkehrt, sobald ein passendes Element gefunden wurde. Davon können vor allem langsame SPARQL Filter profitieren, wenn sie mit anderen Filtern kombiniert werden.

```
public boolean filter(DataItem dataItem) {
    return !filters.stream()

        // Alle Filter anwenden
        .map(filter -> filter.filter(dataItem))

        // Abbruch nach dem ersten 'false'
        .anyMatch(result -> !result);
}
```

Listing 7.2: Der Und-Filter ist durch `anyMatch` short-circuiting

### 7.1.2 Parameter-Annotation

Für Parameter in Modulaktionen und Exportsubmodulen wird das Visitor Pattern verwendet, um eine Benutzungsoberfläche mit den Parametern zu generieren. Dafür muss jedoch immer eine Visitmethode überschrieben werden, die letztlich immer das Gleiche macht: die Visitmethoden aller Parameter mit dem Visitor aufrufen.

Um das zu vermeiden, wird Reflexion eingesetzt. Die Basisklassen der Modulaktionen und Exportsubmodule implementieren eine Visitmethode (siehe Listing 7.4), die alle Felder des Typs `Parameter`, die mit der Annotation `@VisitParameter` annotiert sind, mit dem Visitor besucht. Dadurch reicht es, die Parameterfelder wie in Listing 7.3 zu annotieren und die Visitmethode muss nicht mehr überschrieben werden.

Die Annotation ermöglicht es, dieses automatische Verhalten gezielt für Parameter ein- und auszuschalten. Zudem ist dort eine Dokumentation hinterlegt, die erklärt, was genau dabei passiert.

```
@VisitParameter private StringParameter tagName;
@VisitParameter private StringParameter tagDetail;
```

Listing 7.3: Zwei annotierte Parameter in der TagModuleAction

```
public void visit(ParameterVisitor parameterVisitor) {
    for (Field field : getClass().getDeclaredFields()) {

        // Zugriff auf private Felder ermöglichen
        field.setAccessible(true);

        // nur passende Felder mit Annotation beachten
        if (Parameter.class.isAssignableFrom(field.getType())
            && field.isAnnotationPresent(VisitParameter.class)) {

            // casten und Visitor anwenden
            ((Parameter) field.get(this)).visit(parameterVisitor);
        }
    }
}
```

Listing 7.4: Visit Implementierung über Reflexion

### 7.1.3 Datenimport

Daten werden von Importsubmodulen in das Modell importiert, indem Befehle an eine Datenquelle gesendet und deren Ergebnis ausgewertet und in das Modell eingefügt wird. Listing 7.5 zeigt beispielhaft, wie der im `PackageModule` verwendete Befehl zum Importieren der Paketdaten mit `Dpkg` aussieht. Das Ergebnis der Paketabfrage wird so formatiert, dass die einzelnen Werte wieder isoliert und in das Modell eingefügt werden können. Für einen effizienteren Import führt das Datei- und Verzeichnisimportsубmodul mehrere solcher Befehle gleichzeitig aus.

```
dpkg-query -W -f='${Package}\t${Version}\t${Section}\t
                ${Essential}\t${Priority}\t${Depends}\n'
```

Listing 7.5: Befehl zum Ermitteln der Paketdaten



### 7.1.4 FXML Oberflächenstruktur

Die Struktur der Benutzungsoberfläche wird in mehreren FXML Dateien definiert, die geladen und mit Controllern verbunden werden. Die Darstellungsanpassung wird in einer CSS Datei vorgenommen.

### 7.1.5 Prozedurale Farbgenerierung

Farben für Module und Tags werden prozedural generiert. Dafür wird ein Hash des Namens generiert, zu dem diese Farbe gehören soll. Verschiedene Stellen der Ganzzahlrepräsentation des Hashes werden dann verwendet, um die Parameter einer Farbe im HSB-Farbraum zu setzen. So kann die generelle Helligkeit und Sättigung der generierten Farben gut angepasst werden. Durch Einsatz eines MD5-Hashes [25] ergibt sich eine gute Streuung bei den Farben für unterschiedliche Strings. Dennoch bekommt ein Wert immer die selbe Farbe zugewiesen.

## 7.2 Bibliotheken

Die Anwendung nutzt verschiedene über Maven [22] eingebundene Bibliotheken, die in diesem Abschnitt aufgelistet werden.

### 7.2.1 Boilerplate-Code vermeiden

Um Boilerplate-Code [30] (trivialen, repetitiven und uninteressanten Code) zu vermeiden, kommt hier die Library *Project Lombok* [23] zum Einsatz. Diese ermöglicht es, Annotationen zu setzen, um Getter, Setter, Konstruktoren, toString Methoden und anderen Boilerplate-Code zu generieren. Ein Beispiel dafür wird in Listing 7.6 gezeigt. Tag ist eine vollständige Datenklasse mit Gettern, toString-, equals- und hashCode-Methoden sowie einem Konstruktor, der alle als final deklarierten Felder initialisiert.

```
@Data public class Tag {  
    private final String name;  
    private final String detail;  
}
```

Listing 7.6: Die @Data Lombok Annotation in der Tag Klasse

### 7.2.2 Dependency Injection

*Gluon Ignite* [17] wird zusammen mit *Google Guice* [15] für Dependency Injection [13] (DI) verwendet. Damit können Abhängigkeiten, wie der *UIManager* oder der *ModuleManager* injiziert werden und für Interfaces (wie beispielsweise „Model“) konkrete Implementierungen eingesetzt werden. Ignite sorgt dabei für eine Abstraktion des DI-Frameworks. So werden keine Abhängigkeiten zu einem bestimmten DI-Framework aufgebaut.

### 7.2.3 Weitere Bibliotheken

Für die SSH-Verbindung und die Ausführung von Befehlen auf entfernten Rechnern wird die SSH-Bibliothek *jcabi-ssh* [18] verwendet. Diese Bibliothek ist schlank, einfach zu benutzen und kann verschiedene Authentifizierungsmethoden einsetzen. Importmodule können mehrere Befehle ausführen. Diese können parallel über mehrere SSH-Sessions gleichzeitig ausgeführt werden.

*Apache Jena* [19] wird als Grundlage für das Modell benutzt. Jena ist nicht thread-safe, daher werden *Read / Write Locks* eingesetzt, um den parallelen Zugriff auf das Modell zu organisieren.

*Google Guava* [14] liefert einige nützliche Hilfsklassen und -methoden. Hier wird vor allem der dort enthaltene Eventbus verwendet, der sehr einfach zu handhaben ist. Dieser wird mit Dependency Injection (siehe Abschnitt 7.2.2) verwaltet und an die Kommunikationsteilnehmer verteilt.

Für die Verarbeitung von YAML Objekten für die Erzeugung von Filtern kommt *SnakeYAML* [27] zum Einsatz. Für Eingabefelder mit Autocomplete und Icons und für PopOvers wird die JavaFX Steuerelementebibliothek *ControlsFX* [8] verwendet und die Icons, die die Benutzungsoberfläche übersichtlicher machen, stammen von *FontAwesome* [9].

## Kapitel 8

# Beispielverwendung

Hier soll anhand einer beispielhaften Verwendung der Anwendung gezeigt werden, wie die Möglichkeiten des Programms konkret genutzt werden können. Aus einem (dieser Arbeit beigelegten) Vagrantfile wird dazu mit Vagrant [29] eine virtuelle Maschine mit Ubuntu 12.04 LTS erzeugt und mit der Anwendung analysiert.

In diesem Szenario sind vergleichsweise wenige Dateien und Pakete vorhanden. Das Ziel ist es, wichtige installierte Pakete zu finden, um diese auf einem anderen System zu installieren (siehe Problembeschreibung in Abschnitt 2.2.3).

Dazu werden zunächst im Importscreen Daten vom Testrechner importiert (Abbildungen 8.1 und 8.2). Der Hauptbildschirm (Abbildung 8.3) zeigt die Daten dann an. Um die Pakete besser betrachten zu können wird eine neue Perspektive „Pakete“ mit einem Paketmodul angelegt. Das Modul wird größer gezogen und der Ansichtstyp von Liste auf Tabelle umgestellt (Abbildung 8.5).

Die Pakettabelle wird nach den Begriffen „required“ und „important“ gefiltert und die gefundenen Pakete mit dem Tag „wichtig“ markiert (Abbildungen 8.6 und 8.7). Die mit diesem Tag markierten Pakete werden dann in eine für den Paketmanager Dpkg lesbare Datei exportiert (Abbildungen 8.8 und 8.9, Listing 8.1). Die Aktion „Zugehörige Dateien mit Tag versehen“ (Abbildung 8.6) wird auf alle wichtigen Pakete angewendet.

Es wird zurück zur Übersichtsperspektive gewechselt (Abbildung 8.10). Das Ressourcenmodul wird durch das Modul „Getaggte Ressourcen“ ersetzt. Der Tag „wichtig“ wird aus der Ansicht herausgefiltert. Diese enthält nun nur noch zu wichtigen Paketen gehörende Dateien. Auf diese wird die Aktion „Text kopieren“ angewendet (Abbildung 8.11). Die Pfade aller Dateien von wichtigen Paketen befinden sich nun in der Systemzwischenablage und können zum Abgleich der Pfade auf dem neuen System genutzt werden.

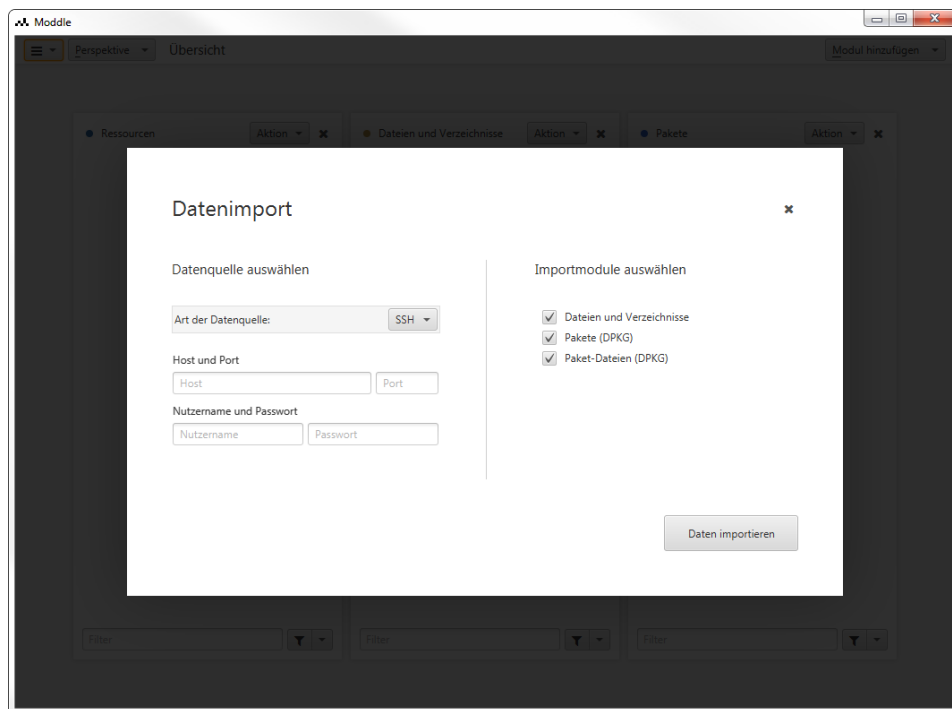


Abbildung 8.1: Der Importbildschirm

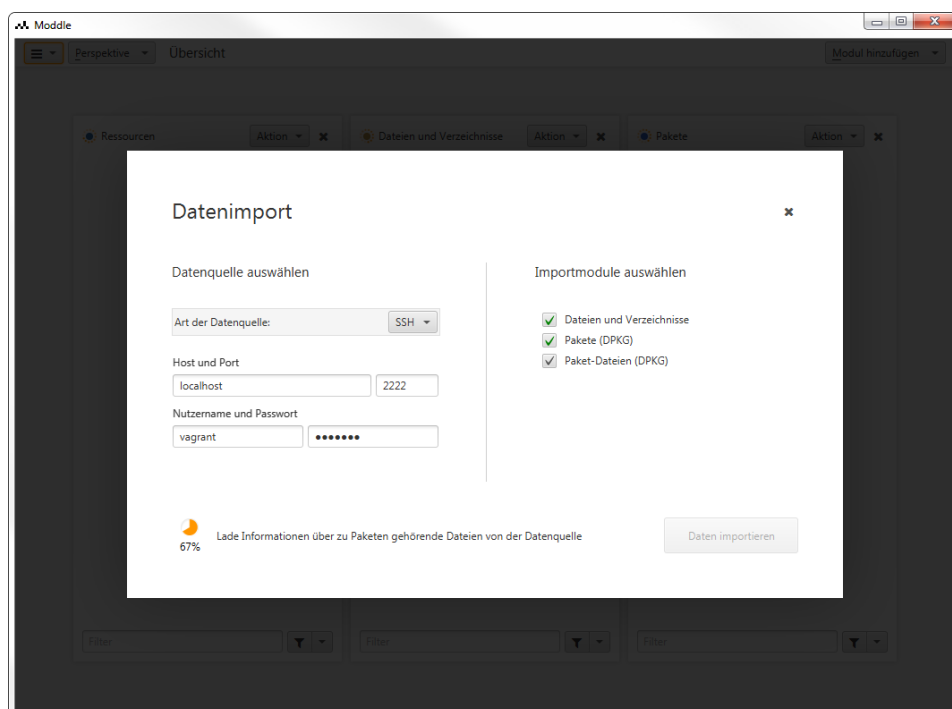


Abbildung 8.2: Der Importbildschirm während dem Importvorgang

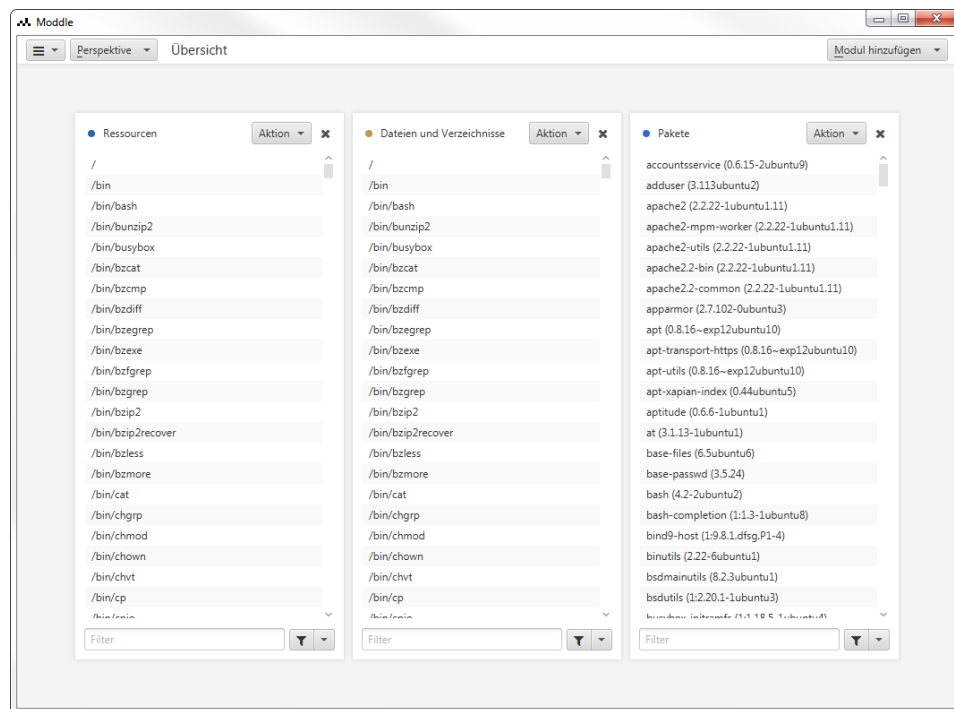


Abbildung 8.3: Der Hauptbildschirm

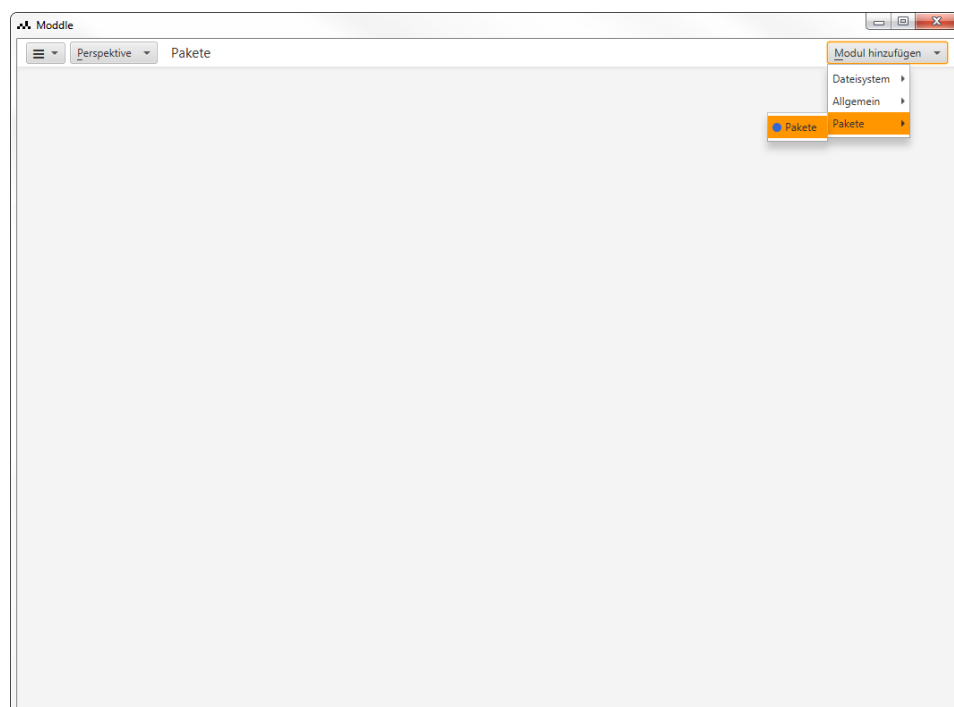
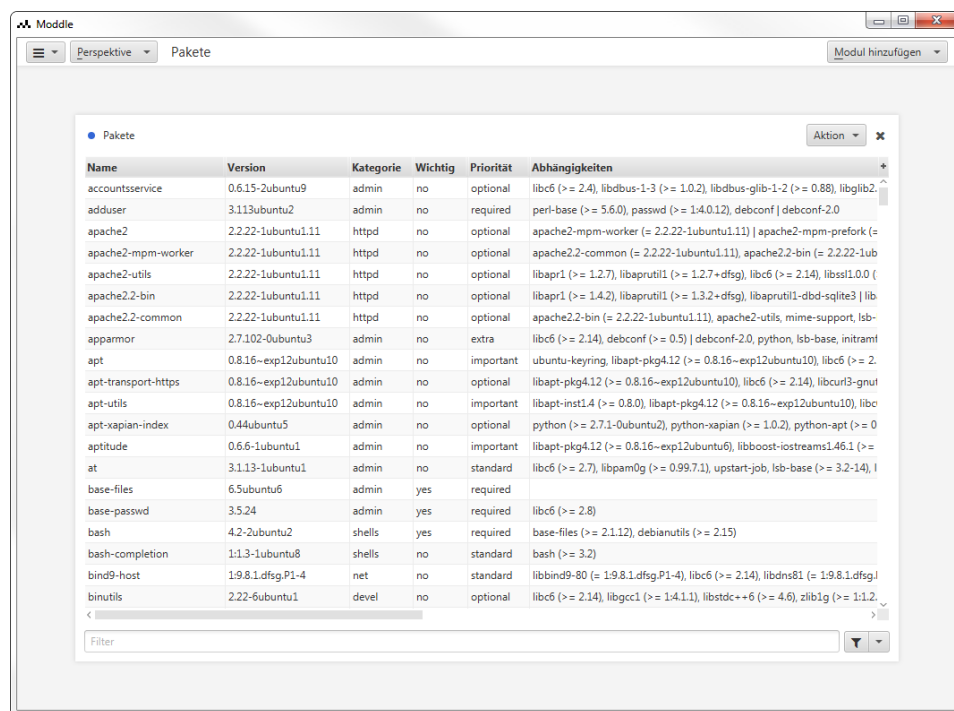
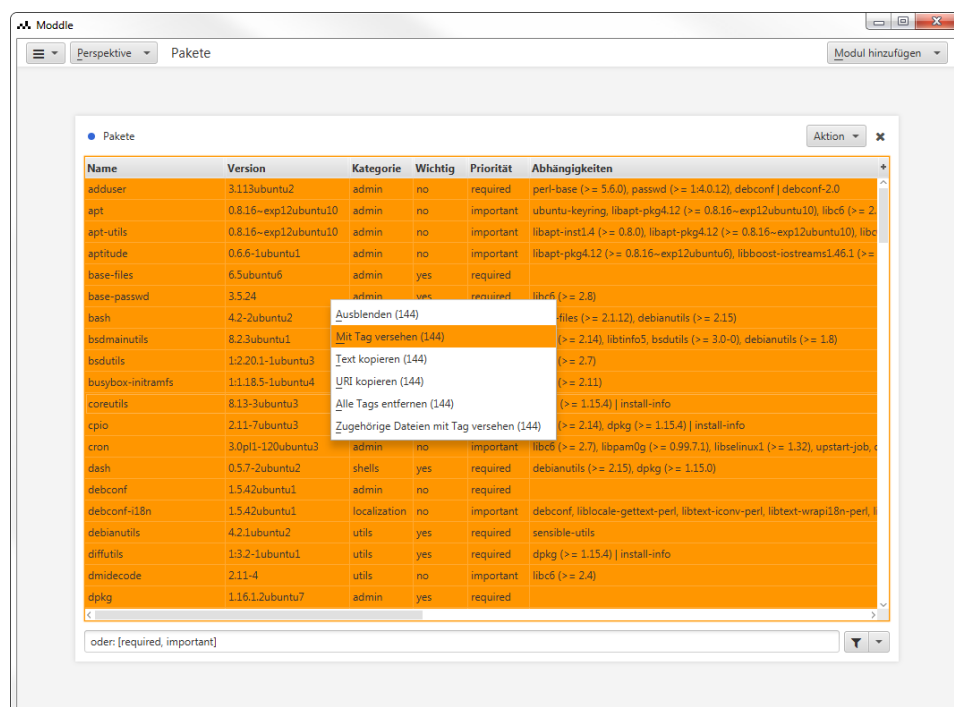


Abbildung 8.4: Eine neue Perspektive, zu der ein Paketmodul hinzugefügt wird



| Name                | Version              | Kategorie | Wichtig | Priorität | Abhängigkeiten   |
|---------------------|----------------------|-----------|---------|-----------|--|
| accountsservice     | 0.6.15-2ubuntu9      | admin     | no      | optional  | libc6 (>= 2.4), libdbus-1-3 (>= 1.0.2), libdbus-glib-1-2 (>= 0.88), libglib2.0-0 (>= 2.30.0), libidn2-0 (>= 2.0.0-4), libnss-systemd (>= 2.35-1), libpam0g (>= 1.1.8-3), libsystemd0 (>= 232-3), libudev1 (>= 232-3), libxml2 (>= 2.9.10)  |
| adduser             | 3.113ubuntu2         | admin     | no      | required  | perl-base (>= 5.6.0), passwd (>= 1:4.0.12), debconf   debconf-2.0  |
| apache2             | 2.2.22-1ubuntu1.11   | httpd     | no      | optional  | apache2-mpm-worker (= 2.2.22-1ubuntu1.11)   apache2-mpm-prefork (= 2.2.22-1ubuntu1.11)   |
| apache2-mpm-worker  | 2.2.22-1ubuntu1.11   | httpd     | no      | optional  | apache2-2-common (= 2.2.22-1ubuntu1.11), apache2-2-bin (= 2.2.22-1ubuntu1.11)  |
| apache2-utils       | 2.2.22-1ubuntu1.11   | httpd     | no      | optional  | libapr1 (>= 1.2.7), libaprutil1 (>= 1.2.7+dfsg), libc6 (>= 2.14), libssl1.0.0 (>= 1.0.1.1)   |
| apache2-2-bin       | 2.2.22-1ubuntu1.11   | httpd     | no      | optional  | libapr1 (>= 1.2.7), libaprutil1 (>= 1.2.7+dfsg), libaprutil1-dbd-sqlite3   libaprutil1-sqlite3   |
| apache2-2-common    | 2.2.22-1ubuntu1.11   | httpd     | no      | optional  | apache2-2-bin (= 2.2.22-1ubuntu1.11), apache2-utils, mime-support, lib-ssl1.0.0 (>= 1.0.1.1)   |
| apparmor            | 2.7.102-0ubuntu3     | admin     | no      | extra     | libc6 (>= 2.14), debconf (>= 0.5)   debconf-2.0, python, lib-base, initramfs-tools, libapparmor1   |
| apt                 | 0.8.16-exp12ubuntu10 | admin     | no      | important | ubuntu-keyring, libapt-pkg4.12 (>= 0.8.16-exp12ubuntu10), libc6 (>= 2.14), libgpg-error0 (>= 1.15-2), libassuan2 (>= 2.0.5-2), libksba8 (>= 1.3.5-2), libldap2 (>= 2.4.42-2), libbz2-1.0 (>= 1.0.6-9), libcurl3-gnutls (>= 7.29.0-2ubuntu2), libexpat1 (>= 2.0.1-6), libffi7 (>= 3.0.4-2), libgnutls30 (>= 3.4.14-2ubuntu2), libidn2-0 (>= 2.0.0-4), libjansson4 (>= 2.10-3), libjson-c5 (>= 0.12-1), libk5crypto3 (>= 1.19-3), libkeyutils1 (>= 1.3-3), libkrb5-3 (>= 1.12-2), libldap-common (>= 2.4.42-2), libldap2-dev (>= 2.4.42-2), libnettle6 (>= 3.4.1-3), libp11-kit0 (>= 0.23.7-3), libtasn1-6 (>= 4.13-3), libunistring2 (>= 0.9.3-9), libxml2 (>= 2.9.10), libzstd1 (>= 0.15.2+dfsg-2) |
| apt-transport-https | 0.8.16-exp12ubuntu10 | admin     | no      | optional  | libapt-pkg4.12 (>= 0.8.16-exp12ubuntu10), libc6 (>= 2.14), libcurl3-gnutls (>= 7.29.0-2ubuntu2)  |
| apt-utils           | 0.8.16-exp12ubuntu10 | admin     | no      | important | libapt-inst1.4 (>= 0.8.0), libapt-pkg4.12 (>= 0.8.16-exp12ubuntu10), libc6 (>= 2.14), libgpg-error0 (>= 1.15-2), libksba8 (>= 1.3.5-2), libldap2 (>= 2.4.42-2), libbz2-1.0 (>= 1.0.6-9), libcurl3-gnutls (>= 7.29.0-2ubuntu2), libexpat1 (>= 2.0.1-6), libffi7 (>= 3.0.4-2), libgnutls30 (>= 3.4.14-2ubuntu2), libidn2-0 (>= 2.0.0-4), libjansson4 (>= 2.10-3), libjson-c5 (>= 0.12-1), libk5crypto3 (>= 1.19-3), libkeyutils1 (>= 1.3-3), libkrb5-3 (>= 1.12-2), libldap-common (>= 2.4.42-2), libldap2-dev (>= 2.4.42-2), libnettle6 (>= 3.4.1-3), libp11-kit0 (>= 0.23.7-3), libtasn1-6 (>= 4.13-3), libunistring2 (>= 0.9.3-9), libxml2 (>= 2.9.10), libzstd1 (>= 0.15.2+dfsg-2)               |
| apt-xapian-index    | 0.44ubuntu5          | admin     | no      | optional  | python (>= 2.7.1-0ubuntu2), python-xapian (>= 1.0.2), python-apt (>= 0.8.16-exp12ubuntu10)   |
| aptitude            | 0.6.6-1ubuntu1       | admin     | no      | important | libapt-pkg4.12 (>= 0.8.16-exp12ubuntu10), libboost-iostreams1.46.1 (>= 1.46.1-2), libc6 (>= 2.14), libgpg-error0 (>= 1.15-2), libksba8 (>= 1.3.5-2), libldap2 (>= 2.4.42-2), libbz2-1.0 (>= 1.0.6-9), libcurl3-gnutls (>= 7.29.0-2ubuntu2), libexpat1 (>= 2.0.1-6), libffi7 (>= 3.0.4-2), libgnutls30 (>= 3.4.14-2ubuntu2), libidn2-0 (>= 2.0.0-4), libjansson4 (>= 2.10-3), libjson-c5 (>= 0.12-1), libk5crypto3 (>= 1.19-3), libkeyutils1 (>= 1.3-3), libkrb5-3 (>= 1.12-2), libldap-common (>= 2.4.42-2), libldap2-dev (>= 2.4.42-2), libnettle6 (>= 3.4.1-3), libp11-kit0 (>= 0.23.7-3), libtasn1-6 (>= 4.13-3), libunistring2 (>= 0.9.3-9), libxml2 (>= 2.9.10), libzstd1 (>= 0.15.2+dfsg-2)  |
| at                  | 3.1.13-1ubuntu1      | admin     | no      | standard  | libc6 (>= 2.7), libpam0g (>= 0.99.7.1), upstart-job, lib-base (>= 3.2-14), libat1 (>= 3.1.13-1)  |
| base-files          | 6.5ubuntu6           | admin     | yes     | required  | libc6 (>= 2.8)   |
| base-passwd         | 3.5.24               | admin     | yes     | required  | libc6 (>= 2.8)   |
| bash                | 4.2-2ubuntu2         | shells    | yes     | required  | base-files (>= 2.1.12), debianutils (>= 2.15)  |
| bash-completion     | 1.13-1ubuntu8        | shells    | no      | standard  | bash (>= 3.2)  |
| bind9-host          | 1:9.8.1.dfsg.P1-4    | net       | no      | standard  | libbind9-80 (>= 1:9.8.1.dfsg.P1-4), libc6 (>= 2.14), libdns81 (>= 1:9.8.1.dfsg.P1-4), libidn2-0 (>= 2.0.0-4), libjansson4 (>= 2.10-3), libjson-c5 (>= 0.12-1), libk5crypto3 (>= 1.19-3), libkeyutils1 (>= 1.3-3), libkrb5-3 (>= 1.12-2), libldap-common (>= 2.4.42-2), libldap2-dev (>= 2.4.42-2), libnettle6 (>= 3.4.1-3), libp11-kit0 (>= 0.23.7-3), libtasn1-6 (>= 4.13-3), libunistring2 (>= 0.9.3-9), libxml2 (>= 2.9.10), libzstd1 (>= 0.15.2+dfsg-2)  |
| binutils            | 2.22-6ubuntu1        | devel     | no      | optional  | libc6 (>= 2.14), libgcc1 (>= 1:4.1.1), libstdc++6 (>= 4.6), zlib1g (>= 1:1.2.11)   |

Abbildung 8.5: Die Paket-Perspektive mit einer Tabellendarstellung der Pakete



| Name              | Version              | Kategorie    | Wichtig | Priorität | Abhängigkeiten   |
|-------------------|----------------------|--------------|---------|-----------|--|
| adduser           | 3.113ubuntu2         | admin        | no      | required  | perl-base (>= 5.6.0), passwd (>= 1:4.0.12), debconf   debconf-2.0  |
| apt               | 0.8.16-exp12ubuntu10 | admin        | no      | important | ubuntu-keyring, libapt-pkg4.12 (>= 0.8.16-exp12ubuntu10), libc6 (>= 2.14), libgpg-error0 (>= 1.15-2), libassuan2 (>= 2.0.5-2), libksba8 (>= 1.3.5-2), libldap2 (>= 2.4.42-2), libbz2-1.0 (>= 1.0.6-9), libcurl3-gnutls (>= 7.29.0-2ubuntu2), libexpat1 (>= 2.0.1-6), libffi7 (>= 3.0.4-2), libgnutls30 (>= 3.4.14-2ubuntu2), libidn2-0 (>= 2.0.0-4), libjansson4 (>= 2.10-3), libjson-c5 (>= 0.12-1), libk5crypto3 (>= 1.19-3), libkeyutils1 (>= 1.3-3), libkrb5-3 (>= 1.12-2), libldap-common (>= 2.4.42-2), libldap2-dev (>= 2.4.42-2), libnettle6 (>= 3.4.1-3), libp11-kit0 (>= 0.23.7-3), libtasn1-6 (>= 4.13-3), libunistring2 (>= 0.9.3-9), libxml2 (>= 2.9.10), libzstd1 (>= 0.15.2+dfsg-2) |
| apt-utils         | 0.8.16-exp12ubuntu10 | admin        | no      | important | libapt-inst1.4 (>= 0.8.0), libapt-pkg4.12 (>= 0.8.16-exp12ubuntu10), libc6 (>= 2.14), libgpg-error0 (>= 1.15-2), libksba8 (>= 1.3.5-2), libldap2 (>= 2.4.42-2), libbz2-1.0 (>= 1.0.6-9), libcurl3-gnutls (>= 7.29.0-2ubuntu2), libexpat1 (>= 2.0.1-6), libffi7 (>= 3.0.4-2), libgnutls30 (>= 3.4.14-2ubuntu2), libidn2-0 (>= 2.0.0-4), libjansson4 (>= 2.10-3), libjson-c5 (>= 0.12-1), libk5crypto3 (>= 1.19-3), libkeyutils1 (>= 1.3-3), libkrb5-3 (>= 1.12-2), libldap-common (>= 2.4.42-2), libldap2-dev (>= 2.4.42-2), libnettle6 (>= 3.4.1-3), libp11-kit0 (>= 0.23.7-3), libtasn1-6 (>= 4.13-3), libunistring2 (>= 0.9.3-9), libxml2 (>= 2.9.10), libzstd1 (>= 0.15.2+dfsg-2)               |
| aptitude          | 0.6.6-1ubuntu1       | admin        | no      | important | libapt-pkg4.12 (>= 0.8.16-exp12ubuntu10), libboost-iostreams1.46.1 (>= 1.46.1-2), libc6 (>= 2.14), libgpg-error0 (>= 1.15-2), libksba8 (>= 1.3.5-2), libldap2 (>= 2.4.42-2), libbz2-1.0 (>= 1.0.6-9), libcurl3-gnutls (>= 7.29.0-2ubuntu2), libexpat1 (>= 2.0.1-6), libffi7 (>= 3.0.4-2), libgnutls30 (>= 3.4.14-2ubuntu2), libidn2-0 (>= 2.0.0-4), libjansson4 (>= 2.10-3), libjson-c5 (>= 0.12-1), libk5crypto3 (>= 1.19-3), libkeyutils1 (>= 1.3-3), libkrb5-3 (>= 1.12-2), libldap-common (>= 2.4.42-2), libldap2-dev (>= 2.4.42-2), libnettle6 (>= 3.4.1-3), libp11-kit0 (>= 0.23.7-3), libtasn1-6 (>= 4.13-3), libunistring2 (>= 0.9.3-9), libxml2 (>= 2.9.10), libzstd1 (>= 0.15.2+dfsg-2)  |
| base-files        | 6.5ubuntu6           | admin        | yes     | required  | libc6 (>= 2.8)   |
| base-passwd       | 3.5.24               | admin        | yes     | required  | libc6 (>= 2.8)   |
| bash              | 4.2-2ubuntu2         | shells       | yes     | required  | base-files (>= 2.1.12), debianutils (>= 2.15)  |
| bsdmainutils      | 8.2.3ubuntu1         | utils        | yes     | required  | libc6 (>= 2.14), libinfo5, bsdutils (>= 3.0.0), debianutils (>= 1.8)   |
| bsdutils          | 1:2.20.1-1ubuntu3    | utils        | yes     | required  | libc6 (>= 2.7)   |
| busybox-initramfs | 1:1.18.5-1ubuntu4    | utils        | yes     | required  | libc6 (>= 2.11)  |
| coreutils         | 8.13-3ubuntu3        | utils        | yes     | required  | libc6 (>= 1.15.4)   install-info   |
| cpio              | 2.11-7ubuntu3        | utils        | yes     | required  | libc6 (>= 2.14), dpkg (>= 1.15.4)   install-info   |
| cron              | 3.0pl1-120ubuntu3    | admin        | no      | important | libc6 (>= 2.7), libpam0g (>= 0.99.7.1), libselinux1 (>= 1.32), upstart-job, cron   |
| dash              | 0.5.7-2ubuntu2       | shells       | yes     | required  | debianutils (>= 2.15), dpkg (>= 1.15.0)  |
| debconf           | 1.5.42ubuntu1        | admin        | no      | required  | libc6 (>= 2.14), dpkg (>= 1.15.4)   install-info   |
| debconf-i18n      | 1.5.42ubuntu1        | localization | no      | important | debconf, liblocale-gettext-perl, libtext-iconv-perl, libtext-wrapi18n-perl, libperl5.28  |
| debianutils       | 4.2.1ubuntu2         | utils        | yes     | required  | sensible-utils   |
| diffutils         | 1:3.2-1ubuntu1       | utils        | yes     | required  | dpkg (>= 1.15.4)   install-info  |
| dmidecode         | 2.11-4               | utils        | no      | important | libc6 (>= 2.4)   |
| dpkg              | 1.16.1.2ubuntu7      | admin        | yes     | required  | libc6 (>= 2.4)   |

Abbildung 8.6: Gefilterte Pakettabelle mit einer ausgewählten Aktion

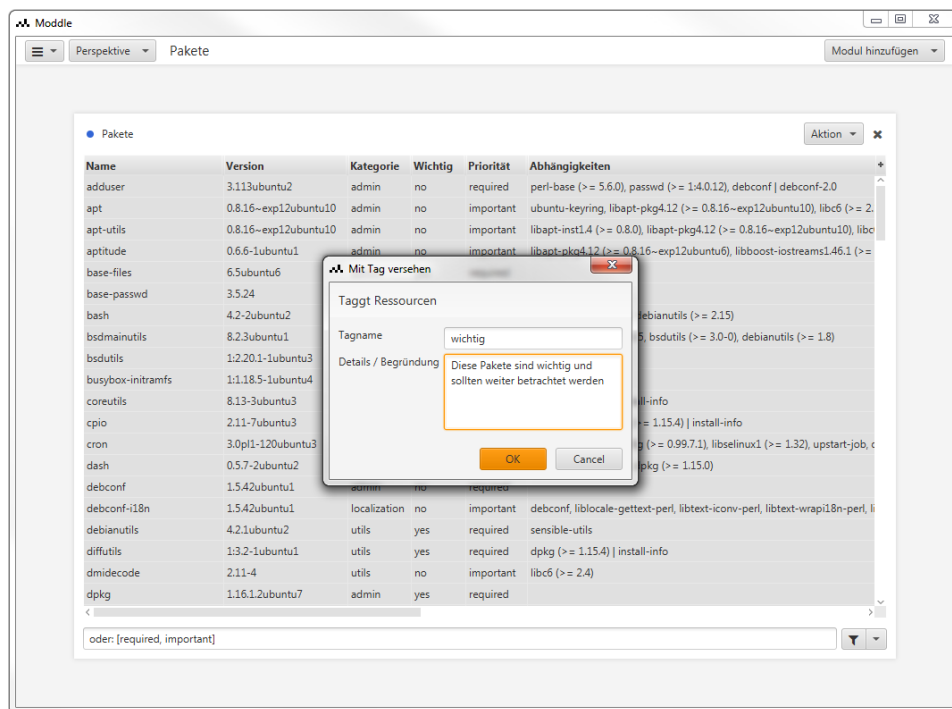


Abbildung 8.7: Hinzufügen eines Tags zu den ausgewählten Paketen

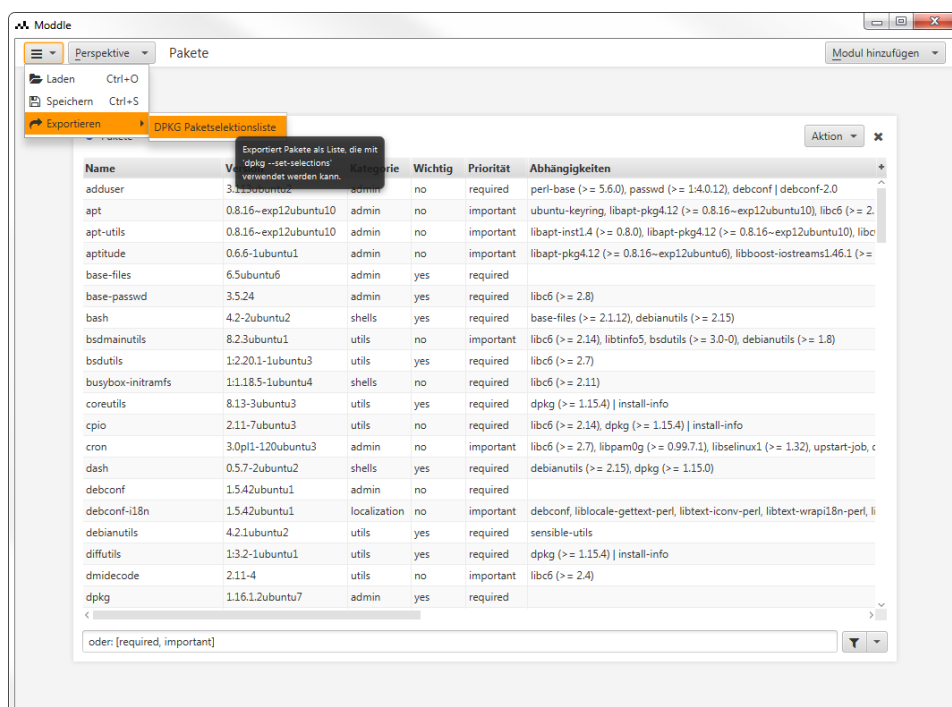
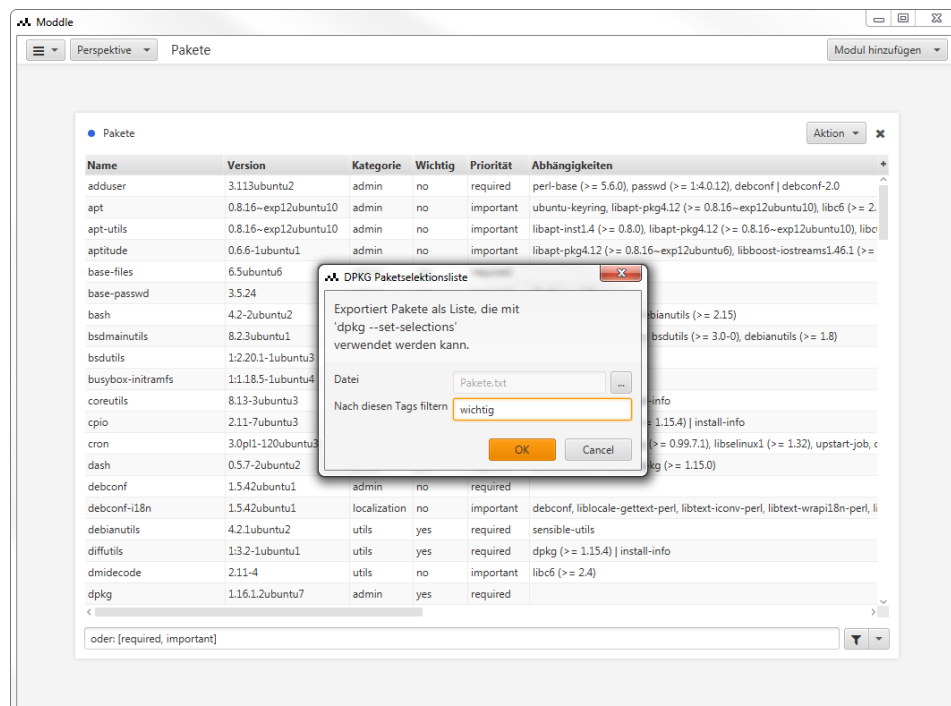


Abbildung 8.8: Exporteintrag im Menü

Abbildung 8.9: Export wichtiger Pakete in die Datei *Pakete.txt*

```
# Pakete: 144
adduser install
apt install
apt-utils install
aptitude install
base-files install
base-passwd install
bash install
bsdmainutils install
bsdutils install
busybox-initramfs install
coreutils install
cpio install
cron install
...
```

Listing 8.1: Ausschnitt aus der exportierten Datei *Pakete.txt*



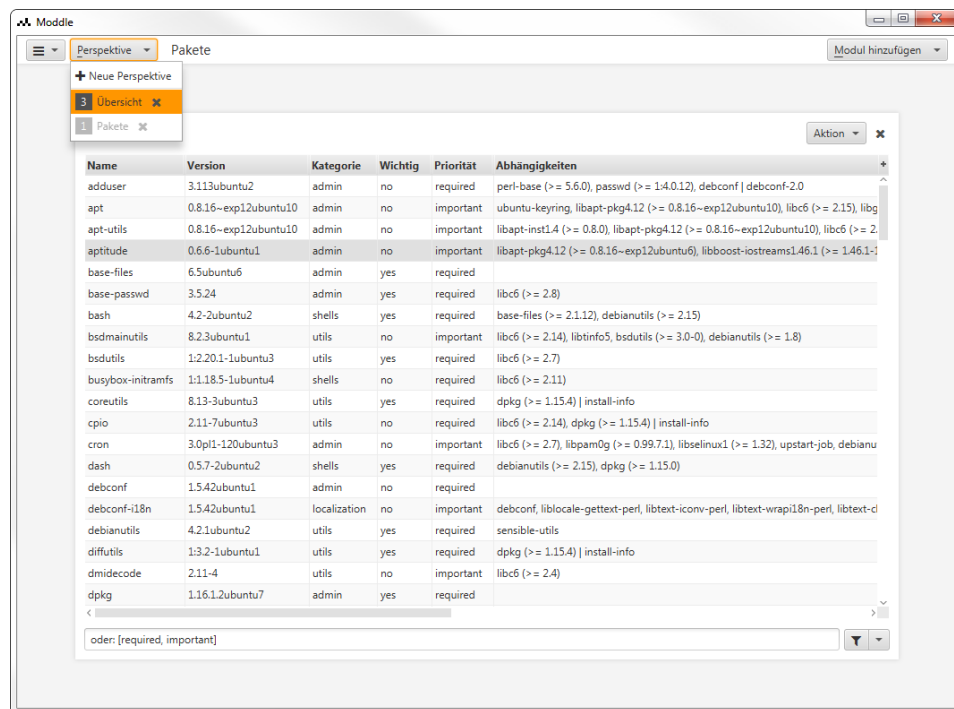


Abbildung 8.10: Wechsel der Perspektive zurück zur Übersicht

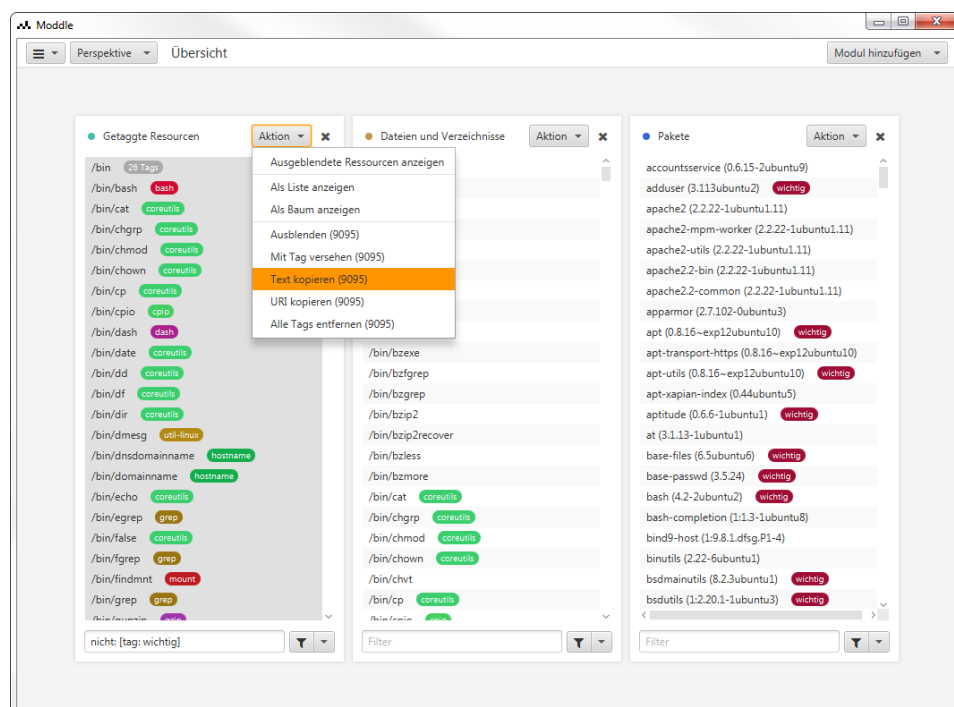


Abbildung 8.11: Kopieren der Pfade aller getaggtten Ressourcen

## Kapitel 9

# Zusammenfassung und Ausblick

Auf Rechnersystemen kann sich eine große Anzahl verschiedenster Softwarekomponenten befinden. Um verschiedene dieser Aspekte zusammen betrachten und auswerten zu können, ist es hilfreich, Informationen und Interaktionen unter einer Oberfläche zu vereinen und Werkzeuge zum Verknüpfen der Daten bereitzustellen. Die in dieser Arbeit entwickelte Anwendung stellt eine Plattform für genau diesen Zweck dar.

Es wurden einige Aspekte des Problemumfeldes gezeigt und festgestellt, dass eine interaktive Analyse geeignet ist, um die Komplexität von Softwarekonfigurationen auch ein verständlicheres Niveau zu bringen. Der Analysevorgang wurde betrachtet und Anforderungen an die Anwendung daraus abgeleitet. RDF wurde als Standard für das Modell gewählt, um eine gute Erweiterbarkeit und Interoperabilität der Anwendung zu erreichen und ein flexibles Datenmodell verwenden zu können.

Anhand von Beispielszenarien wurde gezeigt, welche Möglichkeiten diese Plattform für konkrete Fälle bietet. Die Stärke der Anwendung liegt jedoch in der Modularität, die eine Anpassung an beliebige weitere Fälle ermöglicht. Die Anwendungsarchitektur bietet dabei einige Möglichkeiten zur Erweiterung.

So können Module mit Importsubmodulen, interaktiven Submodulen, Exportsubmodulen, Modulaktionen und Filtern auch zur compilierten Anwendung nachträglich hinzugefügt werden. Darüber hinaus können auch neue Datenquellen (beispielsweise der lokale Rechner oder SSH mit Key), Datenstrukturen, Ansichten und ModuleLoader eingebaut werden. Interessant wäre dabei eine Graph Datenstruktur mit entsprechender Ansicht. Damit könnten Abhängigkeiten und netzwerkartige Daten gut visualisiert werden.

Auch für eine Austauschbarkeit ist gesorgt. Das Modell, das DI Framework, der FilterBuilder (also effektiv die Syntax für Filterabfragen) und der StringToColorCon-

verter (die Berechnung der Tag-Farben) können einfach durch andere Implementierungen ausgetauscht werden.

Für eine zukünftige Entwicklung wären viele kleine Ergänzungen denkbar. Ein Speichern und Laden von benutzerdefinierten Perspektiven würde es ermöglichen, Perspektiven mehrfach zu benutzen oder zu teilen. In einer Detailansicht für Ressourcen könnten Module Informationen zu einzelnen Ressourcen gesammelt anzeigen. Performance-Optimierungen beim Generieren von Baumstrukturen oder dem Aktualisieren von Ansichten würden die Arbeitsgeschwindigkeit erhöhen. Weitere Möglichkeiten zur Interaktion mit verschiedenen Modulen, wie Drag and Drop zwischen Ansichten oder eine globale Selektion können die Nutzererfahrung verbessern und die Arbeit mit der Anwendung effizienter machen.

Am wichtigsten jedoch ist das Hinzufügen weiterer Module, um die Anwendung durch weitere Analysefähigkeiten zu ergänzen. Alle in Abschnitt 2.2 beschriebenen Probleme lassen sich mit entsprechenden Modulen lösen. Für die Dokumentation können über Filter wichtige Elemente gefunden und über Tags mit zusätzlichen zu dokumentierenden Informationen verknüpft werden. Ein Dokumentationsmodul könnte diese Informationen dann nutzen, um eine aussagekräftige Dokumentation zu generieren.

Der Vergleich von Softwarekonfigurationen kann durch den Einsatz des Standards RDF vorgenommen werden, indem mehrere Modelle erstellt und mit RDF-Vergleichswerkzeugen verglichen werden. Eine direkte Integration einer Vergleichsfunktionalität in Form eines Exportmoduls, das Abweichungen oder Gemeinsamkeiten in Form einer Dokumentation exportiert ist ebenfalls denkbar.

Und die Portierung kann unterstützt werden, indem wichtige, beizubehaltende Daten mit Tags versehen werden. Ein entsprechendes Exportmodul kann diese Tags dann auswerten und entsprechende Provisionierungsskripte generieren. Das Paketmodul unterstützt bereits das Exportieren von Paketlisten.

Sinnvoll wären hier auch Module für Nutzerkonten und -einstellungen sowie für automatisch ausgeführte Aufgaben und verwendete Netzwerkports. Zudem könnten Module Dateien (zum Beispiel Konfigurationsdateien) analysieren, auswerten oder validieren.

Diese Arbeit zeigt, wie durch geschickte Modularisierung und Erweiterbarkeit eine Plattform geschaffen werden kann, die für die unterschiedlichsten Einsatzszenarien für die Analyse von Softwarekonfigurationen ein hilfreiches Werkzeug ist.

## Anhang A

# UML Diagramme

Auf den folgenden Seiten befinden sich zusätzliche UML Diagramme, die zum Verständnis einiger Strukturen hilfreich sind.

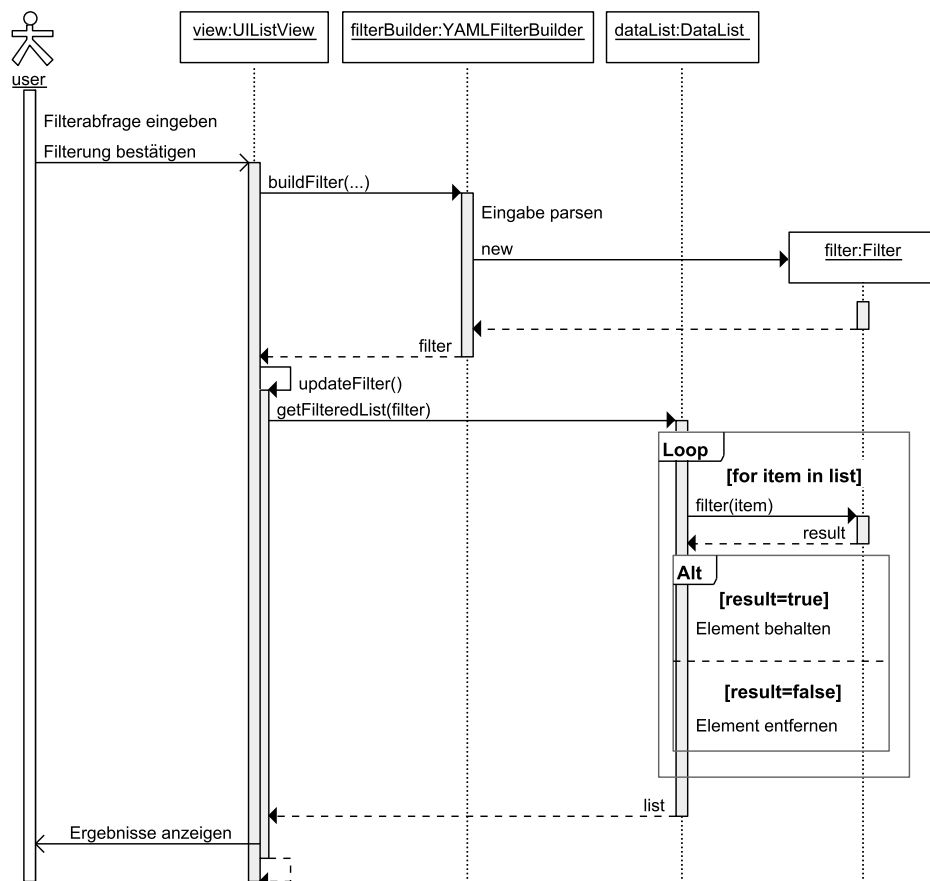


Abbildung A.1: Filterung einer Listenansicht

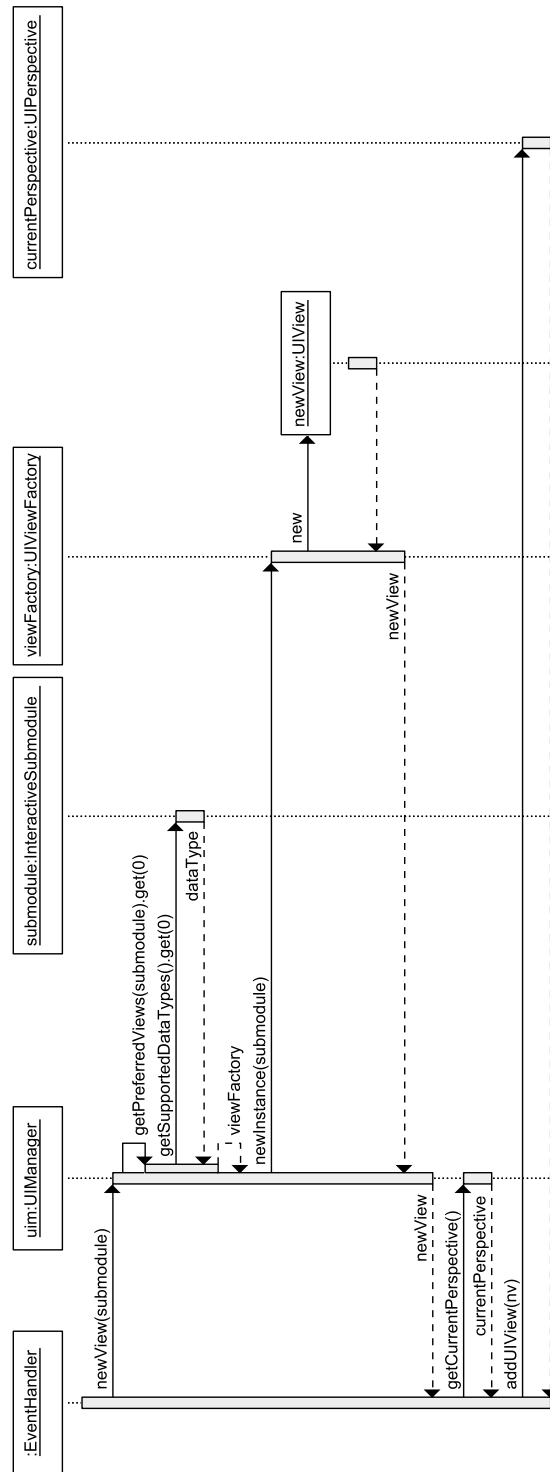


Abbildung A.2: Ablauf beim Hinzufügen eines interaktiven Submoduls

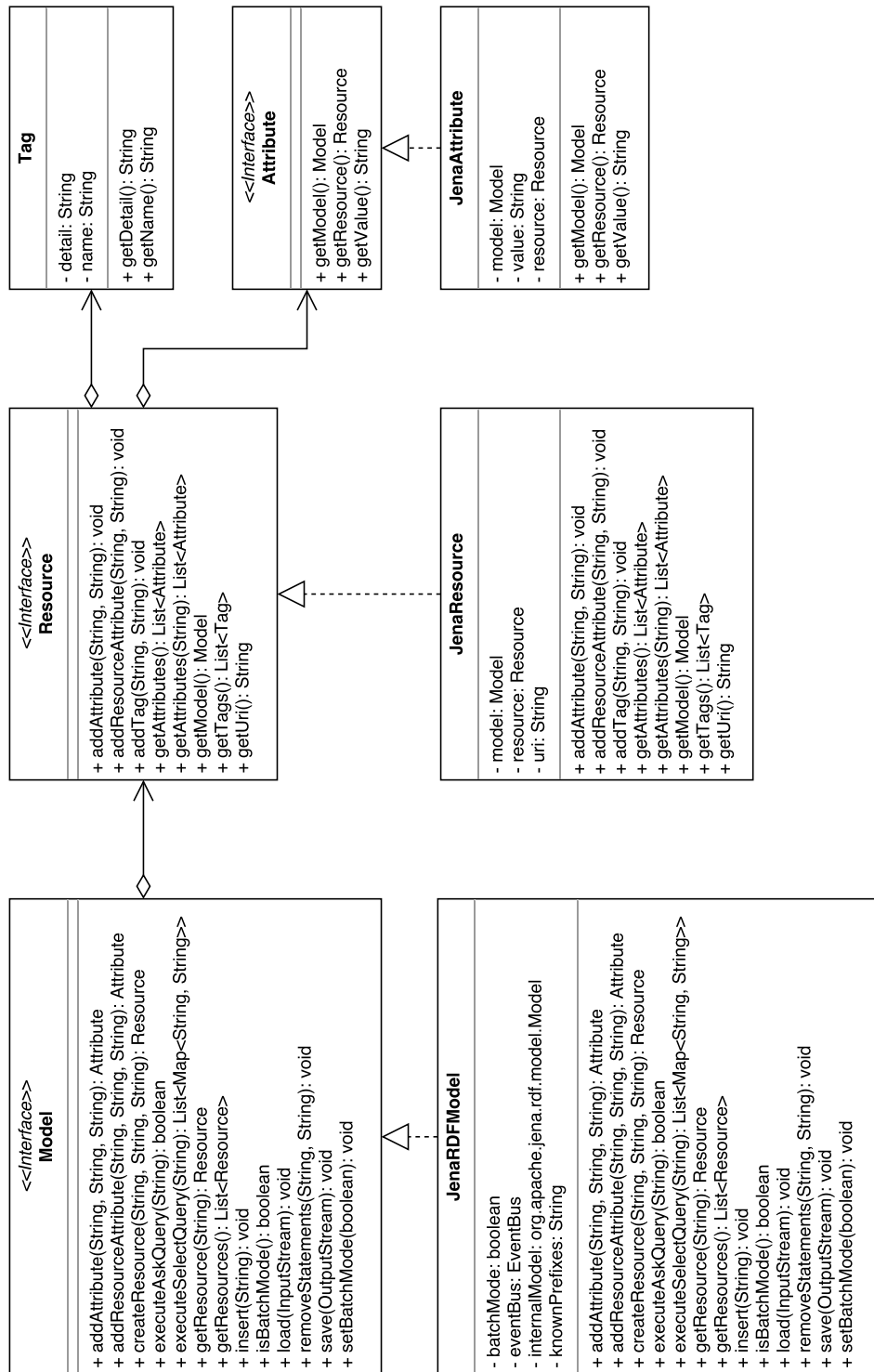


Abbildung A.3: UML Diagramm des Modells

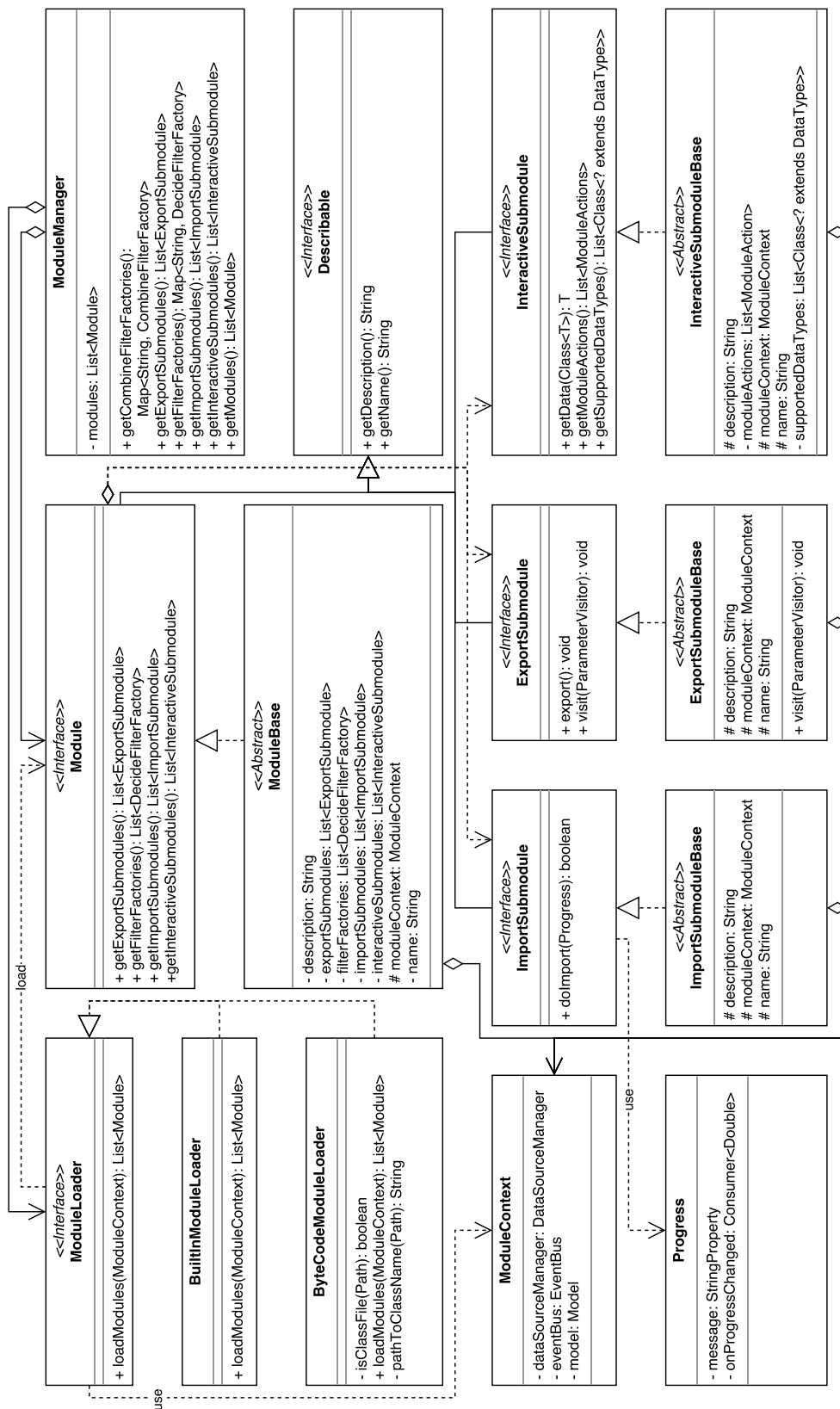


Abbildung A.4: UML Diagramm der Module

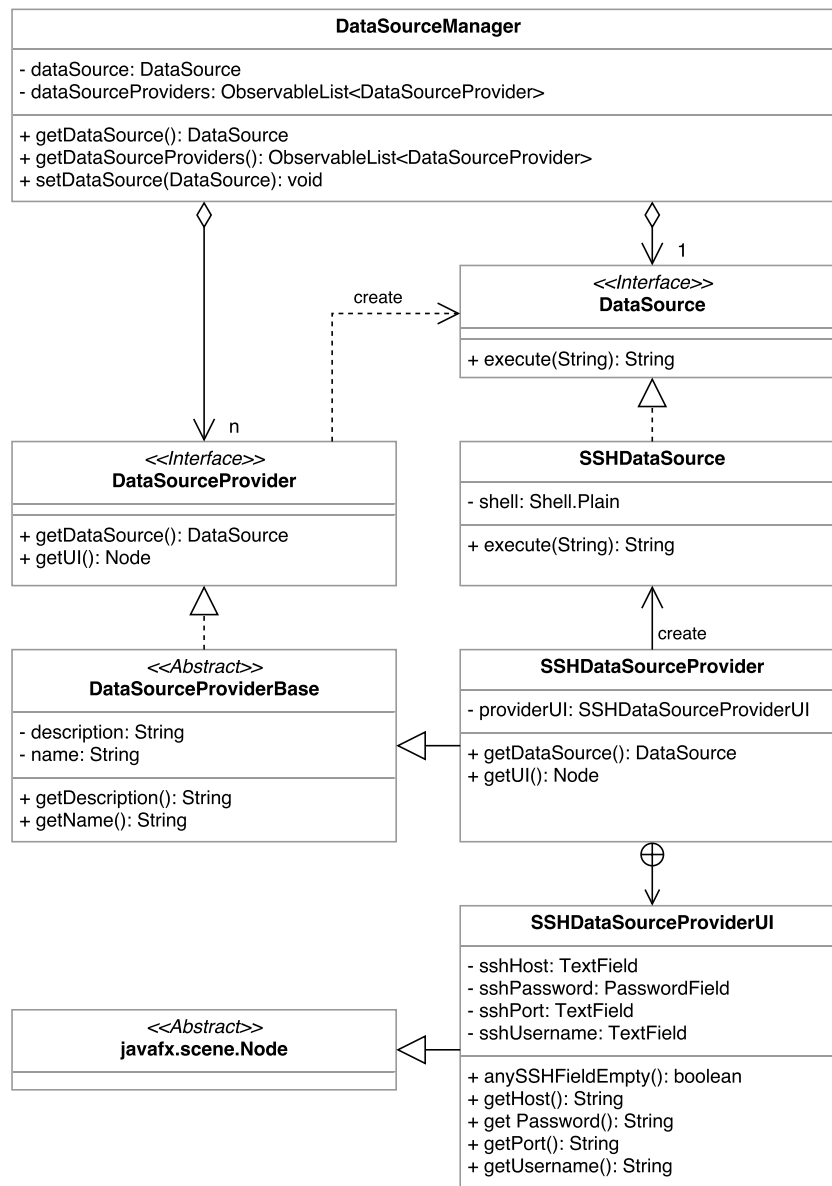


Abbildung A.5: UML Diagramm der Datenquellen



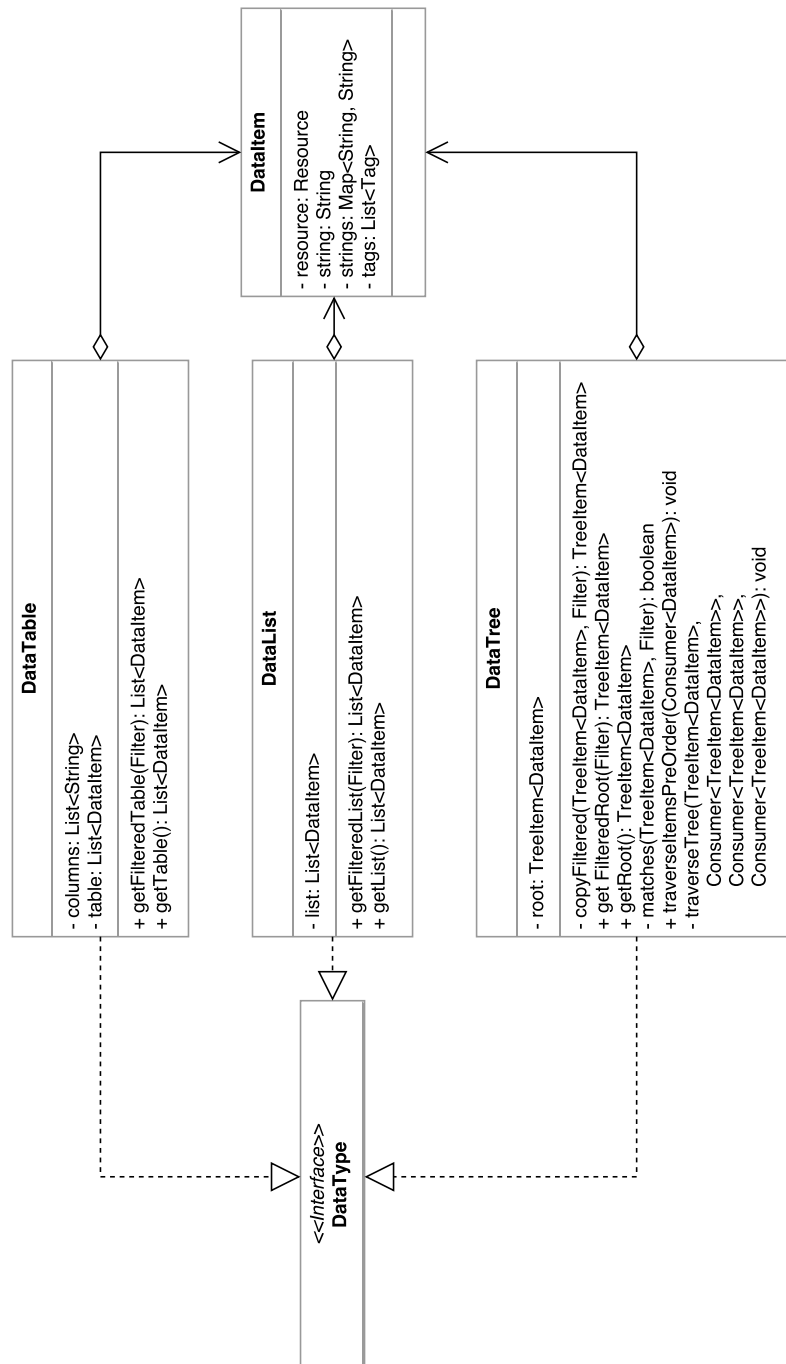


Abbildung A.6: UML Diagramm der Datenstrukturen

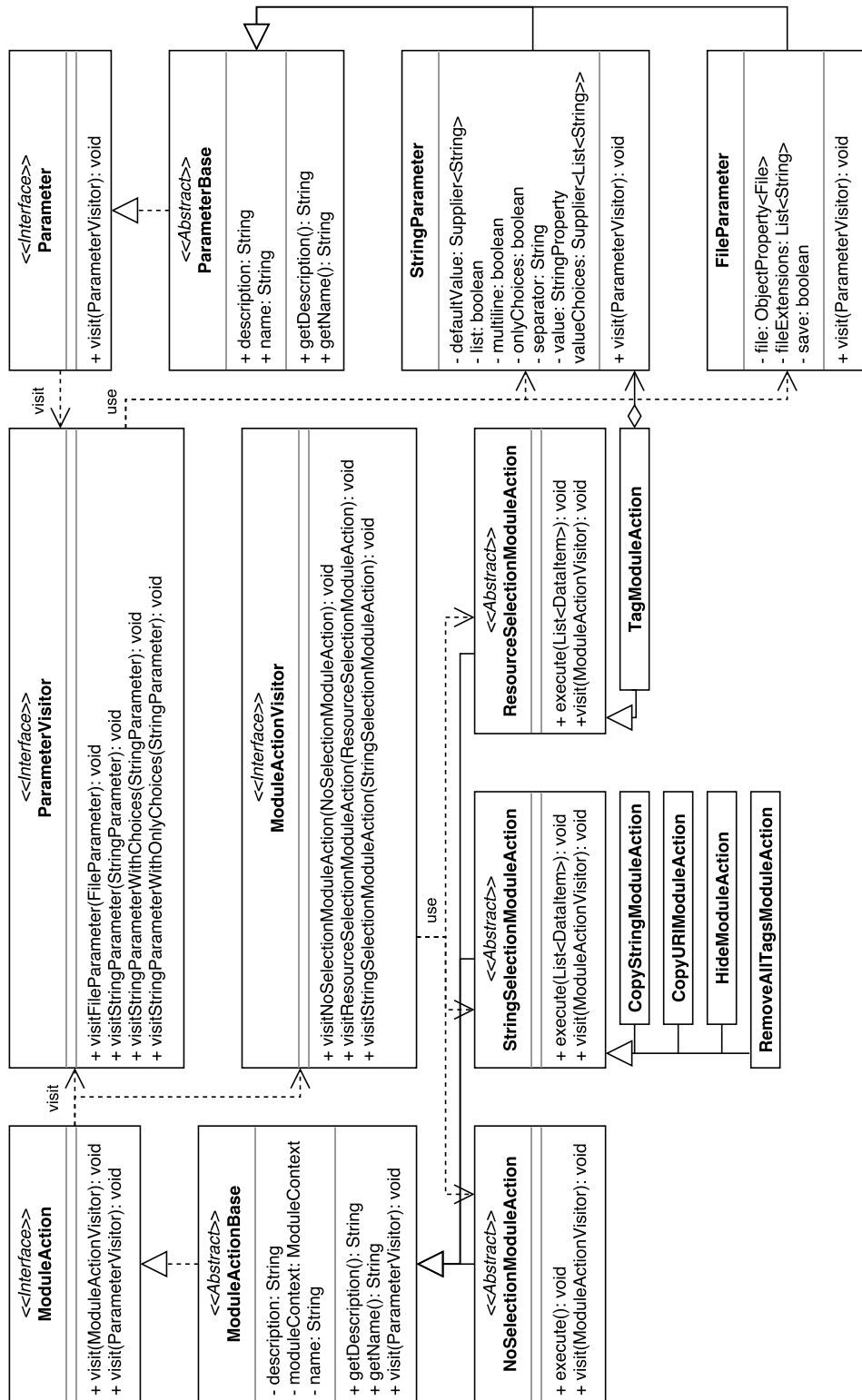


Abbildung A.7: UML Diagramm der Modulaktionen

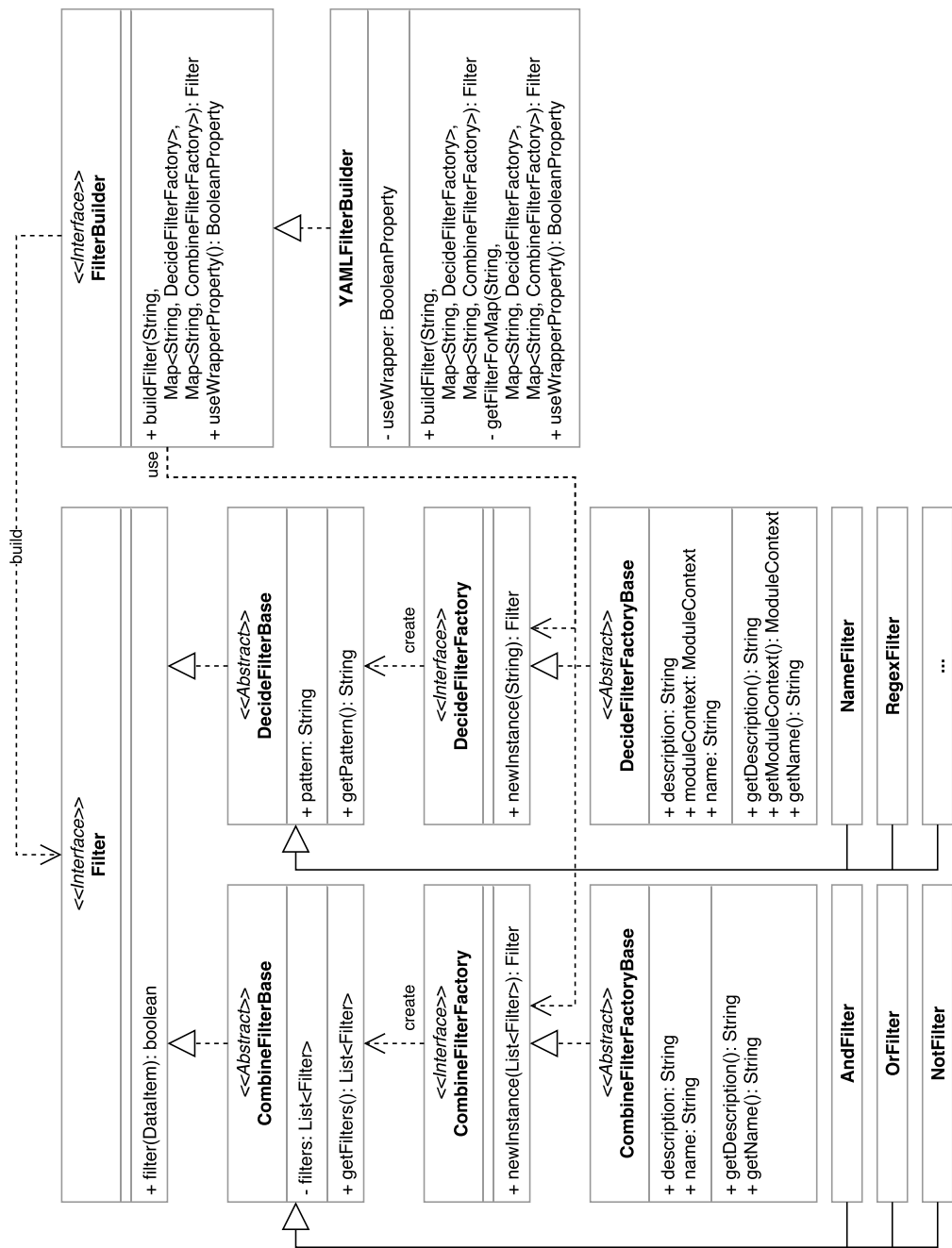


Abbildung A.8: UML Diagramm der Filter

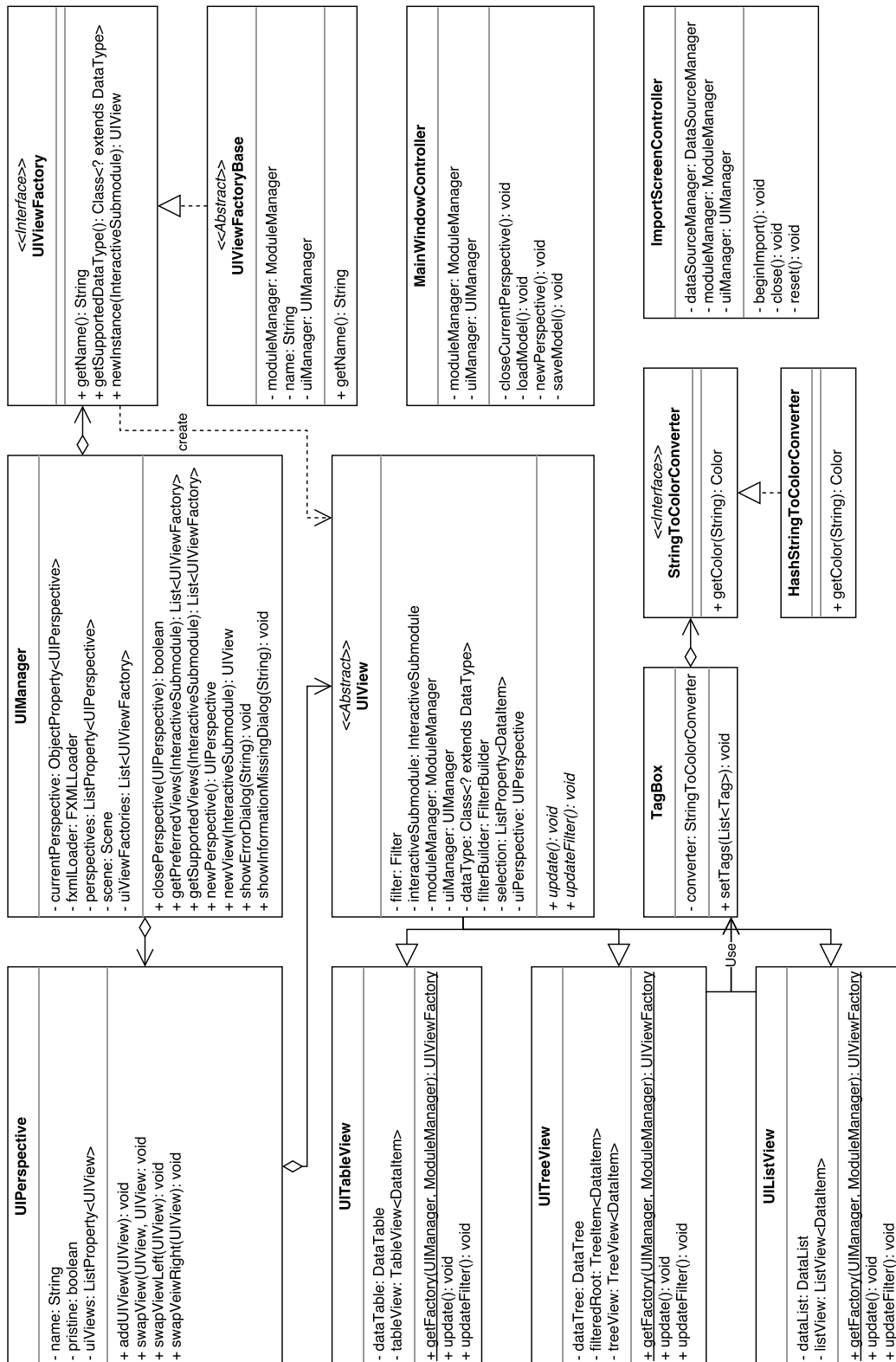


Abbildung A.9: UML Diagramm der Benutzungsoberfläche

## Literaturverzeichnis

- [1] COOPER, ALAN, ROBERT REIMANN und DAVID CRONIN: *About Face*. mitp, Heidelberg, München, Landsberg, Frechen, Hamburg, 2010.
- [2] DAHM, MARKUS: *Grundlagen der Mensch-Computer-Interaktion*. Pearson Studium, München, Boston, 2006.
- [3] GAMMA, ERICH, RICHARD HELM, RALPH JOHNSON und JOHN VLISSIDES: *Design Patterns: Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software*. mitp, Frechen, 2015.
- [4] HITZLER, PASCAL, MARKUS KRÖTSCH, SEBASTIAN RUDOLPH und YORK SURE: *Semantic Web: Grundlagen*. Springer Berlin, Berlin, Heidelberg, 2008.
- [5] RAYMOND, ERIC S.: *The art of UNIX programming*. Addison-Wesley, Boston, 2004.

## Online-Quellen

- [6] BECKETT, DAVID, TIM BERNERS-LEE, ERIC PRUD'HOMMEAUX und GAVIN CAROTHERS: *RDF 1.1 Turtle*. <https://www.w3.org/TR/2014/REC-turtle-20140225/>. letzter Zugriff: 30. Sep. 2016.
- [7] *Chef*. <https://github.com/chef/chef>. letzter Zugriff: 05. Okt. 2016.
- [8] *ControlsFX*. <http://fxexperience.com/controlsfx/>. letzter Zugriff: 01. Okt. 2016.
- [9] *FontAwesome*. <http://fontawesome.io/>. letzter Zugriff: 01. Okt. 2016.
- [10] *Dpkg*. <https://wiki.debian.org/Teams/Dpkg>. letzter Zugriff: 09. Okt. 2016.
- [11] *file-hierarchy(7) Linux User's Manual*. [http://man7.org/linux/man-pages/man7/file-hierarchy.7.html#NODE\\_TYPES](http://man7.org/linux/man-pages/man7/file-hierarchy.7.html#NODE_TYPES). letzter Zugriff: 13. Okt. 2016.
- [12] *Fluent Editor*. <http://www.cognitum.eu/semantics/FluentEditor/>. letzter Zugriff: 04. Okt. 2016.
- [13] FOWLER, MARTIN: *Inversion of Control Containers and the Dependency Injection pattern*. <http://martinfowler.com/articles/injection.html>. letzter Zugriff: 11. Okt. 2016.

- [14] *Guava*. <https://github.com/google/guava>. letzter Zugriff: 29. Sep. 2016.
- [15] *Guice*. <https://github.com/google/guice>. letzter Zugriff: 29. Sep. 2016.
- [16] HAYES, PATRICK J. und PETER F. PATEL-SCHNEIDER: *RDF 1.1 Semantics*. <https://www.w3.org/TR/2014/REC-rdf11-mt-20140225/>. letzter Zugriff: 19. Aug. 2016.
- [17] *Ignite*. <http://gluonhq.com/labs/ignite/>. letzter Zugriff: 29. Sep. 2016.
- [18] *jcabi-ssh*. <http://ssh.jcabi.com/>. letzter Zugriff: 29. Sep. 2016.
- [19] *Jena*. <https://jena.apache.org/>. letzter Zugriff: 29. Sep. 2016.
- [20] *Jenkins*. <https://jenkins.io/index.html>. letzter Zugriff: 09. Okt. 2016.
- [21] LSB WORKGROUP, THE LINUX FOUNDATION: *Filesystem Hierarchy Standard*. [http://refspecs.linuxfoundation.org/FHS\\_3.0/fhs/index.html](http://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html), 2015. letzter Zugriff: 13. Okt. 2016.
- [22] *Maven*. <https://maven.apache.org/>. letzter Zugriff: 13. Okt. 2016.
- [23] *Project Lombok*. <https://projectlombok.org/>. letzter Zugriff: 29. Sep. 2016.
- [24] *Puppet*. <https://puppet.com/>. letzter Zugriff: 05. Okt. 2016.
- [25] RIVEST, RONALD L.: *The MD5 Message-Digest Algorithm*. <https://tools.ietf.org/html/rfc1321>, 1992. letzter Zugriff: 12. Okt. 2016.
- [26] *RPM*. <http://rpm.org/>. letzter Zugriff: 10. Okt. 2016.
- [27] *SnakeYAML*. <https://bitbucket.org/asomov/snakeyaml>. letzter Zugriff: 01. Okt. 2016.
- [28] THE W3C SPARQL WORKING GROUP: *SPARQL 1.1 Overview*. <https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/>. letzter Zugriff: 30. Sep. 2016.
- [29] *Vagrant*. <https://www.vagrantup.com/>. letzter Zugriff: 11. Okt. 2016.
- [30] WIKIPEDIA: *Boilerplate* — *Wikipedia, Die freie Enzyklopädie*. <https://de.wikipedia.org/w/index.php?title=Boilerplate&oldid=158358195>, 2016. letzter Zugriff: 12. Okt. 2016.
- [31] *YAML*. <http://yaml.org/>. letzter Zugriff: 01. Okt. 2016.