technische universität
dortmund

Abschlussbericht der Projektgruppe 566

Situation-Aware Semantic Service Discovery

25. April 2013

Adem Dikmen
Andrej Dudenhefner
Gaetano Geck
Stefan Gleibs
Mike Gösker
Tilmann Heikamp
Mirjam Koch
Timucin Korkmaz
Kevin Kowalinski
Florian Kurpicz
Vera Tomskikh

Lehrstuhl 5 Programmiersysteme Prof. Dr. Bernhard Steffen
Fakultät Informatik
Technische Universität Dortmund
Otto-Hahn-Straße 14
44227 Dortmund
http://ls5-www.cs.tu-dortmund.de

Abschlussbericht der Projektgruppe 566
Thema: Situation-Aware Semantic Service Discovery

Eingereicht: 25. April 2013

Betreuer: Oliver Bauer, Stefan Naujokat

Lehrstuhl 5 Programmiersysteme Prof. Dr. Bernhard Steffen
Fakultät Informatik
Technische Universität Dortmund
Otto-Hahn-Straße 14
44227 Dortmund
http://ls5-www.cs.tu-dortmund.de

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Der Abschlussbericht zur Projektgruppe (kurz PG) Situation-Aware Semantic Service Discovery (SASSD) im Sommersemester 2012 und Wintersemester 2012/2013 gibt einen Überblick über den gesamten Verlauf des Projektes, verdeutlicht Entwicklungsentscheidungen, beschreibt das entstandene Softwareprodukt und hält Probleme und Erfolge während der Arbeit nach.

In der heutigen Wirtschaft und Forschung findet das Konzept der Services und Workflows zunehmend Anwendung. Repetitive Tätigkeiten werden durch die Wiederverwendbarkeit von Services, als kleinen Programmen oder Arbeitsschritten, sowohl schneller geplant als auch leichter durchgeführt und auch komplexere Aufgaben können durch die Zerlegung in einzelne Schritte leichter bewältigt werden. Wie so oft besteht allerdings bei der Verwendung von Services und Arbeitsablaufplänen nur dann ein großes zeitliches Einsparungspotential, wenn der zusätzliche Aufwand zur Planung nicht letztlich mehr Zeit kostet, als eingespart wurde. Dazu muss gewährleistet sein, dass auch bei einer großen Auswahl von Services schnell und sinnvoll aus diesen ausgewählt werden kann.

Unsere PG hat es sich an dieser Stelle zum Ziel gesetzt, den Auswahlprozess zu vereinfachen und zu verbessern, ohne mehr Wissen vom Anwender zu verlangen. Dazu soll ihm bei der Zusammensetzung der Workflows ein Empfehlungssystem an die Seite gestellt werden, das die Menge der zur Verfügung stehenden Services verwaltet und damit beherrschbar macht. Die Empfehlungen sollten aus Services bestehen, die aufgrund unterschiedlicher Kriterien mit hoher Wahrscheinlichkeit den gerade in Entwicklung befindlichen Workflow gut weiterführen. Ähnlich zu bekannten Empfehlungssystemen z. B. beim Einkauf übers Internet sollen dabei Services empfohlen werden, die ihrer Beschreibung nach zum aktuellen Workflow passen, häufig im ähnlichen Kontext bisheriger Workflows auftauchten, sehr beliebt sind oder von ähnlichen Nutzergruppen oft verwendet werden.

Sowohl um einen praxisbezogenen Einstieg in die Thematik zu bekommen, als auch, weil in diesem Bereich bereits nutzbare Datenbestände existieren, haben wir uns für den Bereich der Bioinformatik entschieden. Mehrere Organisationen und Gruppen haben Projekte initialisiert, in denen Services und Workflows aus diesem Bereich geteilt und verwaltet werden können.

Bioinformatiker untersuchen biologische und chemische Vorgänge häufig mit Hilfe von Computerprogrammen, indem sie beispielsweise DNA als Zeichenketten codieren und diese dann entsprechend der erkannten Transkriptionsregeln in Zeichenketten umwandeln, die mRNA codieren. Weitere mögliche Schritte wären die Umwandlung in Proteine oder die Untersuchung der Formveränderungen der RNA-Strings. Diese Prozesse werden so oft benötigt, dass sie bereits als Services zur Verfügung stehen und in Workflows miteinander verknüpfbar sind.
Im Jahr unserer gemeinsamen Arbeit haben wir uns zunächst mit verschiedenen Datenquellen beschäftigt und vertraut gemacht. Gleichzeitig haben wir nach geeigneten Möglichkeiten gesucht, ein Empfehlungssystem so zu implementieren, dass es nicht nur für Bioinformatiker sondern auch Anwender aus anderen Fachgebieten nutzbar ist. Wegen der erwähnten Verbreitung des Workflow-Konzepts haben wir stets möglichst große Flexibilität im Bezug auf die verwendeten Services angestrebt. Egal, in welchem Bereich die Services und Workflows verwendet werden, sollte das Empfehlungssystem geeignete Empfehlungen ausgeben. Verwaltung und Empfehlung von Services war die oberste Priorität des Projektes. Da Services mit unterschiedlichen Ausführungssemantiken aus unterschiedlichen Quellen betrachtet wurden, war die Ausführbarkeit entstehender Workflows in einer bestimmten Workflowmanagementumgebung kein Ziel der PG.
Im konkreten Fall standen uns Datenbanken zur Verfügung, in denen sich Services, Workflows, Ontologien und Nutzerstatistiken befanden. Wir mussten also einerseits Importmodule für diese konkreten Quellen entwickeln, aber andererseits auch ein zielführendes Metamodell auswählen oder entwickeln.
Wir haben verschiedene Algorithmen als Plug-Ins implementiert, die auf etablierten Verfahren basieren und von uns für Serivces und Workflows angepasst wurden. Gleichzeitig wurde das Empfehlungssystem, das die jeweiligen Empfehlungen einzelner Algorithmen gewichtet kombiniert, in eine Workflowmanagementumgebung integriert. Dabei war uns wichtig, dass Nutzer die verwendeten Datenquellen und Algorithmen mit ihrer jeweiligen Gewichtung mit Hilfe einer GUI anpassen können. Der Aspekt der Kompatibilität und Flexibilität ist vor allem bei Projekten wie diesem, das auf früherer Arbeit aufbaut und später wieder verwendet werden soll, besonders wichtig. Daher wurde ein besonders großer Wert auf den modularen Aufbau des Projektes gelegt und es wurden passende Technologien zum Entwurf und zur Implementierung des Projektes verwendet.
Teilweise erfolgreiche Tests unseres Empfehlungssystems bestärken uns in der Annahme, dass im Bereich der Workflows weitere Fortschritte möglich sind, die das Arbeiten mit diesen Prinzipien weiter erleichtern werden. Probleme und Verbesserungsmöglichkeiten, die sich während des Testens und der Evaluation herauskristallisiert haben, zeigen Perspektiven auf.

1.1 Aufbau des Berichts

Um spätere Entwicklungen verständlich darstellen zu können, werden in Kapitel 2 verschiedene Grundlagen erläutert. Angefangen mit den Grundbegriffen zum Um-

gang mit Services und Workflows, über die uns zur Verfügung stehenden Datenbanken bis hin zu theoretischen Grundlagen von Recommender-Systemen.
In Kapitel 3 werden der Entwurf und die Implementierung der Recommender-Systems beschrieben, wobei zunächst die Softwareauswahl besprochen wird. Danach folgen grundlegende Konzepte unserer Modellierung und die Beschreibung der Systemarchitektur. Abschließend werden in diesem Kapitel die entwickelten Importmodule und Algorithmen des von uns implementierten Recommender-Systems beschrieben.
Kapitel 4 befasst sich mit den Aufgaben zur Integration des Empfehlungssystems in eine Workflowumgebung, die im zweiten Semester der Projektgruppe zu bewältigen waren.
In Kapitel 5 folgen die Evaluation und der Ausblick, wobei in der Evaluation die Qualität des Produkts im Vordergrund steht und im Ausblick auch ein abschließendes Fazit gezogen wird.

2 Grundlagen

In diesem Kapitel werden alle Grundlagen vorgestellt, die notwendig sind, um die eigentliche Aufgabe und Arbeit zu verstehen. Zunächst sollen wichtige Grundbegriffe und -prinzipien für die Arbeit mit Workflows erläutert werden. Darüber hinaus werden die verwendeten Datenquellen vorgestellt, in denen sich Informationen zu Services und Workflows aus der Bioinformatik und Ontologien befinden. Während die Ontologien wichtige Struktur-Informationen liefern, bilden die aus den Services und Workflows gewonnenen Daten die Basis für die Empfehlungen des Recommender-Systems. Zuletzt werden die grundlegenden Konzepte von Empfehlungssystemen dargestellt; Zustandekommen und Evaluation von Empfehlungen werden beleuchtet.

2.1 Grundbegriffe

In diesem Abschnitt werden die wichtigsten Begriffe aus der Workflow-, Service- und Recommender-Praxis beschrieben. Hierzu ist anzumerken, dass statt des englischen Begriffs *Service* auch von einem *Dienst* gesprochen werden kann.

2.1.1 Service

Ein Service beschreibt zunächst einmal jede Art von „Vorgehen“. Auf formaler Ebene kann das die Bearbeitung einer Datenmenge als Eingabe durch einen Algorithmus sein, der eine neue Datenmenge als Ausgabe erzeugt [Erl07]. Zur Beschreibung von Geschäftsprozessen werden aber auch sehr informelle Vorgänge als Service betrachtet; beispielsweise kann der Einkauf in einer Bäckerei als Service mit dem Kundenwunsch als Eingabe gesehen werden, in dem die gewünschte Ware ausgehändigt und das Geld übergeben wird. Die Ausgabe eines solchen Services lässt sich leichter als neuer „Zustand“ des Systems verstehen, könnte aber entsprechend kodiert auch einfach ein Ausgabedatenstrom sein.

Die Idee hinter der Bezeichnung dieser Vorgänge als „Services“ ist abgesehen von der gewonnenen Übersicht die Wiederverwendbarkeit. So ein allgemeiner und wiederverwendbarer Vorgang ist beispielsweise der „Bezahlvorgang“. Dieser kommt in unterschiedlichen Geschäften vor, der Ablauf ist in der Regel immer gleich. Die detaillierte Modellierung von einzelnen Aspekten wie der Rückgabe von Wechselgeld oder ähnlichem muss dann nur einmal erfolgen und kann immer wieder benutzt werden. Um diese Wiederverwendbarkeit zu ermöglichen, werden spezialisierte Vorgänge wie die Barzahlung mit Wechselgeld oder die EC-/Kreditkartenzahlung mit PIN-Eingabe oder Unterschrift in Services beschrieben. Eine weitere Aufteilung in kleinere Bestandteile ist normalerweise nicht sinnvoll.
Im Rahmen der Projektgruppe sind Services von besonderer Bedeutung, wie sie in Datenbanken wie BioCatalogue [Bioa] verwendet und im Kontext der Bioinformatik eingesetzt werden. Für diese gilt: Ein *Service* ist ein autonomes, plattformunabhängiges Softwaremodul, welches sich im Idealfall durch folgende Eigenschaften auszeichnet [Bel08]:

Eigenständigkeit Der Service ist hoch modular und kann unabhängig angewendet werden.

Verteilte Komponente Der Service ist über das Netzwerk erreichbar und über einen Namen, der sich von der absoluten Netzwerkadresse unterscheidet, ansprechbar.

Öffentliche Schnittstelle Nutzer der Services müssen nur die Schnittstelle kennen und sich nicht um genaue Details der Implementierung kümmern.

Interoperabilität Servicenutzer und -anbieter können verschiedene Implementierungssprachen und –plattformen verwenden.

Auffindbarkeit Ein spezieller Datenverzeichnisservice erlaubt es, dass sich Services registrieren und so von Nutzern gefunden werden können.

Dynamische Einbindbarkeit Zum Zeitpunkt der Workflow-Erstellung muss die Implementierung eines Services noch nicht unbedingt fertiggestellt oder für den Nutzer verfügbar sein. Der Service wird bei der Workflow-Ausführung in Echtzeit lokalisiert und eingebunden.

Ein großer Vorteil von Services in der Anwendung besteht darin, dass ein Nutzer die Funktionsweise und Implementierung nicht kennen oder verstehen muss, um den Service einsetzen zu können. Über eine natürlichsprachliche Beschreibung ist die Gesamtfunktion des Services für ihn ersichtlich.

2.1.2 Workflow

Ein *Workflow* ist eine Verkettung von mehreren Services, welche die oben aufgezählten Eigenschaften von Services besitzen. Dabei ist es Voraussetzung, dass Start-

und Endzustände existieren. Die Verknüpfung der Services kann linear sein, oder auch Verzweigungen und Schleifen enthalten. Services sind hoch modular und können unabhängig verwendet werden. Über ihre öffentliche Schnittstelle können sie flexibel angesteuert werden. Ein Nutzer eines solchen Services muss die Eingabedaten und die Aufgabe des Services verstehen, nicht jedoch dessen Implementierung. Insbesondere kann er die Informationen darüber, welche Eingaben ein Service erwartet und welche Ausgaben er liefert, nutzen, um mehrere Services adäquat zu verbinden. Die Services können so hintereinander ausgeführt werden, um größere Aufgaben zu lösen, und werden als Workflow zusammengefasst. Hierzu sind auch die Eigenschaften der Auffindbarkeit (um die Komponenten des Workflows festlegen zu können) und der Erreichbarkeit über das Netzwerk (Verteilte Komponente) von großem Nutzen [Bad08].
Workflows können prinzipiell in Form von Subworkflows als Komponenten in anderer Workflows dienen, weil sie genau wie Services über Eingabe- und Ausgabedaten verfügen. Die Voraussetzung dafür ist eine hierarchische Modellierung von Workflows. Auf diese Weise wird das Konzept der Wiederverwendbarkeit von Services weiter ausgedehnt und auf ihre Zusammenstellung übertragen.
Ein Beispiel hierfür sind Vorgänge, die in einem Versandunternehmen immer wieder vorkommen, aber auf unterschiedliche Produkte angewendet werden können. Während ein Service das Verpacken eines konkreten Produkts (beispielsweise eines Kleidungsstücks oder eines Handys) beschreibt, könnte ein Workflow aus den Komponenten Verpackung, Frankierung und Adressierung erstellt werden. Dieser könnte dann allgemein die Vorbereitung eines Pakets beschreiben. Ein übergeordneter Workflow könnte die gesamte Bestellung eines Kunden beschreiben, in der die Paketvorbereitung auf die konkrete Bestellung folgt und parallel zu Buchungsprozessen vorgenommen wird. Durch den Aufbau von Workflow aus Start- und Endzustand, Services und deren Verknüpfungen lassen sich Workflows als gerichtete Graphen modellieren. Eine Verzweigung kann durch mehrere Kanten; eine Schleife als Kreis dargestellt werden. Daher bietet sich für das Bearbeiten von Workflows ein graphischer Editor an, welcher es dem Nutzer erlaubt Services zu kombinieren. Dabei liefert die Ausgabe eines Services in einfachen Fällen oft bereits die Eingabe des nachfolgenden Services. Auch ist es denkbar, dass eine Ausgabe durch einen weiteren Service zunächst konvertiert oder an mehreren Kanten zur Verfügung gestellt werden muss, so dass sie wieder als Eingabe dienen kann.

2.1.3 Workflowmanagementumgebungen

Workflowmanagementumgebungen sind Programme wie die Taverna Workbench [Tav], mit denen Workflows verwaltet und erstellt werden können. In der einfachsten Ausprägung ergibt sich dadurch die Möglichkeit, Workflows aus Services zu erstellen (bzw. zu definieren) und zu verwalten; die Ausführung der Services bzw. Workflows kann auf einem anderen Computer oder durch ein anderes Programm erfolgen, denkbar wäre die Bearbeitung auf einem Arbeitsplatzrechner und die Ausführung auf einem leistungsstarken Server.

Das von der Vorgänger-Projektgruppe entwickelte Tool „Eclipse4Bio“ [Ecla] ist jedoch nicht nur eine Management- sondern gleichzeitig auch eine Ausführungsumgebung. Mit diesen Programmen können Services aus entsprechenden Bibliotheken abgerufen werden und über ihre Bezeichnungen (oder Annotationen) ausgewählt und einem Workflow hinzugefügt werden. In der entsprechenden Ausführungsumgebung können Workflows oder Services passende Eingabe-Daten in Form von Parametern erhalten und gestartet werden. Die Verknüpfung der beiden Funktionen hat offensichtliche Vorteile, denn nicht nur die Entwicklung von Workflows wird dadurch erleichtert, sondern auch das Testen. Eine Einarbeitung von Veränderungen in einen Workflow ist jederzeit möglich, Veränderungen der Ausgabe können sofort nachvollzogen werden. Insbesondere, weil Workflows sehr viele verschiedene Prozesse bzw. als Service bezeichnete Vorgänge enthalten können, hat das Trennen der beiden Funktionen den Vorteil, dass eine Workflowmanagementumgebung nicht abhängig von einem bestimmten Typ der Services ist und in ihr auch „natürlichsprachliche“ Services miteinander kombiniert werden können ohne etwaige Verarbeitungsprobleme auf der Ebene der Ausführung behandeln zu müssen.

2.2 Datenquellen

Die Typen verwendbarer Empfehlungsverfahren werden insbesondere durch die zur Verfügung stehenden Daten bestimmt. Abgesehen von Informationen zu den zu empfehlenden Entitäten (den Services) können weitere Daten, die Zusammenhänge zwischen diesen Entitäten beschreiben, ermittelt und mit dem System verwaltet werden. Folgende Abschnitte geben ein Überblick über die zu diesem Zweck verwendeten Internetportale und Programmbibliotheken, eingeteilt nach den bereitgestellten Informationen für einzelne Services, Workflows und Ontologie-Beschreibungen.

2.2.1 Servicedatenbanken

Für den Import von Service-Beschreibungen stehen die umfangreiche Programmbibliothek des EMBOSS-Projektes wie auch das Internetportal BioCatalogue zur Verfügung.

EMBOSS

EMBOSS [Emb] steht für *European Molecular Biology Open Software Suite* und ist ein Software-Paket für den Fachbereich der Molekularbiologie. Es enthält neben zahlreichen bestehenden Anwendungen (in Form von Services) auch umfangreiche Bibliotheken zur Entwicklung und Veröffentlichung von neuer Software als Open Source. Die bereits vorhandenen Dienste können eine Vielzahl von unterschiedlichen Dateiformaten verarbeiten und sind in verschiedene Anwendungsgruppen kategorisiert. Das EMBOSS-Projekt existiert bereits seit dem Jahr 2000 und der Release der aktuellen Version 6.4.0 wurde im Juli 2011 verkündet.

BioCatalogue

Die Plattform BioCatalogue [Bioa] ist ein Verzeichnis für Web-Services aus dem Bereich der Bioinformatik. Angemeldete Nutzer können hier nicht nur eigene Services anbieten, sondern auch die Beschreibung von anderen Services durch Hinzufügen von Tags und Annotationen verbessern. Begonnen wurde das Projekt im Juni 2008 und seitdem wurden mehr als 2200 Services aufgenommen. Es gibt über 600 registrierte Nutzer, die neben den bereits genannten Aktionen auch noch Services als ihre persönlichen Favoriten auswählen können. Entwickler können die Inhalte der Plattform nicht nur über die Webseite, sondern auch über sogenannte REST-APIs (*Representational State Transfer*, [Fie12]) abfragen.
Es hat den Anschein, als würde im Jahr 2012 von den Verantwortlichen nicht mehr weiter daran gearbeitet. Aufgrund der Menge von bereits vorhandenen Informatio-

nen erscheint es dennoch als eine geeignete Datenquelle für das von der Projektgruppe zu entwickelnde Recommender-System.

2.2.2 Workflowdatenbank

Neben Services sollen auch bereits bestehende Workflows gesammelt werden, da dies wichtige Informationen darüber liefern kann, auf welche Art und Weise die Services in der Vergangenheit verwendet wurden. Es erscheint sinnvoll, Plattformen auszuwählen, die mit den betrachteten Service-Verzeichnissen in Verbindungen stehen.
Es kann allerdings vorkommen, dass die extrahierten Workflows zum Teil Services verwenden, die nicht in der verwendeten Service-Datenbanken verzeichnet sind. Dennoch liefern die restlichen Workflow-Bestandteile (wie zum Beispiel die zu oder von solchen Services verlaufenden Kanten) wichtige Daten und Informationen, die wir für die Empfehlungen verwenden können.

MyExperiment

MyExperiment [Mye] ist ein Partnerprojekt von BioCatalogue und bietet ein Verzeichnis für vollständige Workflows in unterschiedlichen Formaten. Insgesamt wurden bereits mehr als 1.800 Workflows in die Plattform eingepflegt, die zum Großteil in den anwendungsspezifischen Formaten von Taverna 2, Taverna 1 und RapidMiner spezifiziert sind. Eine Workflow-Beschreibungsseite auf der Webseite enthält üblicherweise ein Diagramm mit der grafischen Darstellung des Workflows, so dass ein menschlicher Besucher den Ablauf des Workflows auf den ersten Blick leicht nachvollziehen kann. Weiterhin existieren eine textuelle Beschreibung, Lizenzinformationen und auch eine übersichtliche Auflistung der einzelnen Workflow-Komponenten. Genau wie bei BioCatalogue können registrierte Nutzer zudem die von ihnen bevorzugten Workflows als Favoriten markieren oder mit Annotationen versehen. Auf MyExperiment können sich Nutzer außerdem zu Gruppen zusammenschließen, um ihre Zusammenarbeit organisatorisch zu vereinfachen.
Bei der Suche nach Workflows kann nach diversen Parametern (etwa nach Dateityp, Tags, Nutzern und Lizenzen) gefiltert werden. Für jeden Workflow ist auf der Webseite auch angegeben, wie oft er bereits angesehen und heruntergeladen wurde. Es lassen sich zudem neue Versionen eines bestehenden Workflows veröffentlichen, ohne einen komplett neuen Eintrag dafür anlegen zu müssen. Neben Workflows können auf MyExperiment auch Dateien anderer Art und sogenannte „Packs“ hochgeladen werden. Packs können zum Beispiel Zusammenstellungen von mehreren Workflows sein, die für eine bestimmte Anwendung gemeinsam sinnvoll sind. Darüber hinaus gibt es auf der Webseite einen Reiter mit dem Namen *Services*, welcher auf Service-Beschreibungen von BioCatalogue verweist. Auch MyExperiment verfügt über eine REST-API, die den Download der vorhandenen Daten ermöglicht.

2.2.3 Ontologiedatenbank

Neben den Service- und Workflowdaten können weitere Informationen in die Datenbank aufgenommen werden: Ontologie-Konzepte, welche im Vergleich zu den anderen Daten stärker auf eine (explizite) Darstellung der semantischen Eigenschaften der Services abzielen.
Eine Ontologie stellt eine Kozeptualisierung eines Gebietes dar [KI10], also eine Beschreibung eines komplexen Bereichs (in diesem Fall der Bioinformatik) durch mehrere Konzepte, die verschiedene Objekte des jeweiligen Bereichs repräsentieren. Zwischen diesen Objekten bestehen Beziehungen, die ebenfalls in der Ontologie enthalten sind. Eine Ontologie kann als gerichteter Graph dargestellt werden, wobei Konzepte die Knoten und Beziehungen die Kanten des Graphens sind. Entsprechende Repräsentationen von Ontologien ermöglichen einem Rechner, Zusammenhänge zu verarbeiten, welche andernfalls nur für einen Menschen ersichtlich wären, der über ausreichende Kenntnisse in dem entsprechenden Themengebiet verfügt.

EDAM

EDAM [Eda] steht für *EMBRACE Data and Methods* und ist eine Ontologie für den Bereich der Bioinformatik, die Verbindungen zwischen diversen Konzepten formalisiert darstellt. EDAM steht in direktem Zusammenhang zu EMBOSS, so dass jeder EMBOSS-Service mit entsprechenden EDAM-Annotationen versehen ist. Aus diesem Grund ist es sehr sinnvoll, die EDAM-Informationen bei Verwendung von EMBOSS-Services für Empfehlungen zu berücksichtigen.

BioPortal

BioPortal [Biob] ist eine Plattform, auf der mehrere Ontologien aus dem Bereich der Biologie und Bio-Informatik bereitgestellt werden. Es kann zum Beispiel nach einzelnen Begriffen in mehreren Ontologien gleichzeitig gesucht werden. In die Plattform integriert ist auch eine Art Empfehlungssystem, das zu einem bestimmten Thema die am ehesten passende Ontologie benennt, sowie die Möglichkeit der automatischen Annotation von Text mit entsprechend relevanten Begriffen. Insgesamt kann das Portal eine große Menge an Informationen liefern, auch wenn es nicht direkt mit den anderen genannten Projekten in Verbindung steht.

2.3 Empfehlungssysteme

Um die theoretischen Hintergründe von Empfehlungssystemen zu beschreiben, werden zunächst allgemeine Eigenschaften von Empfehlungssystemen vorgestellt; dann folgen verschiedene Arten der Empfehlungsgenerierung mit ihren jeweiligen Vor- und Nachteilen
Darüber hinaus werden auch Prinzipien der (schwierigen) Evaluation von Empfehlungssystemen vorgestellt, die später in der konkreten Evaluation unseres Projekts Anwendung finden.

2.3.1 Grundlagen

Ein *Empfehlungssystem* (auch *Recommender-System* genannt) ist „ein System, dass einem Benutzer in einem gegebenen Kontext aus einer gegebenen Menge an Entitäten aktiv eine Teilmenge ‚nützlicher' Elemente empfiehlt"[Kla09].

Entität

Der Begriff der „Entität" aus der Definition bezeichnet hier die Menge aller Möglichkeiten für den User, Dinge auszuwählen oder etwas auszuführen. Je nach System können also beispielsweise die Waren, in einem Shop, die zur Verfügung stehenden Entitäten sein oder Handlungen in einem Tätigkeitsablauf. Die Entitäten beschreiben also einen Teil des Kontextes, da aus der Änderung der zur Verfügung stehenden Entitäten genau wie auch aus Änderung des Profils eine neue Empfehlung zustande kommen kann[Kla09].

Kontextarten

Der in der obigen Definition verwendete Begriff Kontext kann in folgende vier Arten aufgeteilt werden[1]

Situationskontext: *Wo befindet sich der Nutzer?*
Dazu gehören beispielsweise Geodaten über seinen Standort.

Nutzerkontext: *Was tut der Nutzer oder will er tun?*
Hierin inbegriffen sind etwa die aktuell von ihm verwendeten Programme oder Webseiten.

[1] vgl. http://smartweb.dfki.de

Diskurskontext : *Was tat der Nutzer bisher?*
Hier wird die Ausführungshistorie des Nutzers ausgewertet, zum Beispiel der Inhalt vergangener Suchanfragen.

Ontologiekontext: *Worum geht es?*
Hier wird analysiert, welche Zusammenhänge es gibt und mit welcher Intention der Nutzer bisher gehandelt zu haben scheint. Hierunter könnte etwa die Suche nach einem Geschenk für den Partner fallen.

Situation

„Die Situation S konstituiert sich aus den Rahmenparametern der realen Welt (Datum, Uhrzeit, Geoinformationen, verwendetes Endgerät des Benutzers, gerade angezeigter Text im Browser des Benutzers etc.)“ [Kla09]. Allgemeine Informationen wie der Standort eines Nutzers können beispielsweise bei Einkäufen relevant sein, da manche Anbieter nicht in jedes Land versenden. Ein weiteres Beispiel ist auch, dass z.B. manche Inhalte, in bestimmten Gegenden, nur zu bestimmten Uhrzeiten verfügbar sein sollen. Spezifische Informationen zu der vom User verwendeten Technik können sowohl für die Erstellung des Benutzerprofils wichtig sein, als auch für die Einordnung des Nutzers im System. Ein anderer einfacher Fall wäre Werbung oder das Angebot zur Weiterleitung noch während der Nutzer eine Anfrage eingibt.

Benutzerprofil

Bevor das Recommender-System Empfehlungen ausgeben kann, ist es, wie in der Definition angedeutet, wichtig, dass Informationen über das Nutzerverhalten vorliegen. Das Benutzerprofil kann aus *impliziten* und *expliziten* Informationen bestehen [Kla09]. Das System verarbeitet dann das aus beiden Informationsarten zusammengesetzte Benutzerprofil, um Empfehlungen zu generieren.

- ***Implizite Informationen*** Implizite Informationen sammelt das Recommender-System im Diskurskontext; es analyisiert beispielsweise die Ausführungshistorie des Nutzers. Die zuvor aufgerufenen Seiten, typische Verhaltensweisen oder ähnliches werden zur Datengewinnung zu Verhaltensmustern zusammengefasst, die das Empfehlungssystem für daran orientierte Empfehlungen nutzen kann.

- ***Explizite Informationen*** Bei der expliziten Informationsgewinnung werden Nutzern direkte Informationen abverlangt. Hier kommen zum Beispiel Formulare mit Fragen zu Geschlecht oder Alter zum Einsatz. Für bestimmte Nutzergruppen typische Eigenschaften oder mit diesen verknüpfte Informationen, können vom Recommender-System verwendet werden, um Empfehlungen für einen speziellen Nutzer zu generieren. Ein typisches Beispiel wäre die Empfehlung von Kosmetik für Nutzer, die angegeben haben, dass sie weiblich sind.

Mathematische Definition eines Empfehlungssystems

Zusätzlich zur natürlichsprachlichen Definition und Klärung der grundlegenden Begrifflichkeiten soll nun eine mathematische Definition eines Empfehlungssystems erfolgen. Diese Grundlage ist wesentlich, um die einzelnen Berechnungen des Empfehlungssystems definieren und verstehen zu können. Mithilfe dieser allgemeinen mathematischen Berechnungen findet das System die für spezifische Nutzer relevanten Empfehlungen.
Folgende Formalisierung stellt den Zusammenhang zwischen Empfehlung und Nutzerkontext mathematisch dar[Kla09]:

$$\max_T(Nutzwert(B, K, T)) \text{ mit } K = (P, M, S)$$

Hierbei steht B für den Benutzer, K für den Kontext und T für die Menge der empfohlenen Elemente. Der Kontext K ist als ein Tripel dargestellt, bestehend aus Benutzerprofil P, der Entitätsmenge M und der Situation S.
Insgesamt soll das Empfehlungssystem diejenigen Elemente T empfehlen, für die der Nutzwert im gegebenen Kontext und für den aktuellen Benutzer maximal ist.

2.3.2 Empfehlungen

Auf die vorangegangenen Grundlagen aufbauend wird in diesem Kapitel näher erläutert, wie genau ein Empfehlungssystem arbeitet und welche Empfehlungsarten es gibt.

Grober Ablauf einer Empfehlung

Folgende Abbildung[TH01] gibt einen Überblick darüber, wie der Prozess einer Empfehlung eines Empfehlungssystems, definiert durch Terveen und Hill [TH01], abläuft.

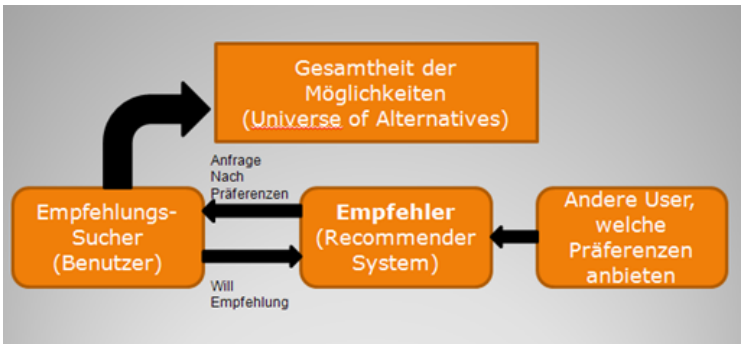


Abbildung 2.1: Prozess einer Empfehlung

Wie in der Abbildung zu erkennen ist, steht der Benutzer grundsätzlich mit seinen Wünschen vor der Gesamtheit seiner Entscheidungsmöglichkeiten (dem „Universe of Alternatives“). Er erhält Empfehlungen vom Empfehlungssystem, das wiederum im Fall des kollaborativen Filterns auf Informationen durch andere Nutzer zurückgreift.

Im Allgemeinen sind diese Empfehlungen weiter dadurch zu unterscheiden, dass sie dem Nutzer nach dessen Aufforderung oder unaufgefordert zukommen. Darüber hinaus gibt es die Unterscheidung nach Empfehlungen, die sich nur aus den Daten des momentanen Nutzers generieren und solchen, die dazu Daten von anderen Nutzern des Systems verwerten und auf statistische Erfahrungen zurückgreifen.
Besonders gut ist eine Empfehlung dann, wenn sie in beiden Fällen vor allem die Präferenzen des aktuellen Benutzers berücksichtigt [Emp].

Arten einer Empfehlung

In diesem Abschnitt werden die verschiedenen Kategorisierungen einer Empfehlung allgemein erläutert. Die nachfolgende Abbildung 2.2 visualisiert die verschiedenen, in der Praxis Anwendung findenden Empfehlungsarten.

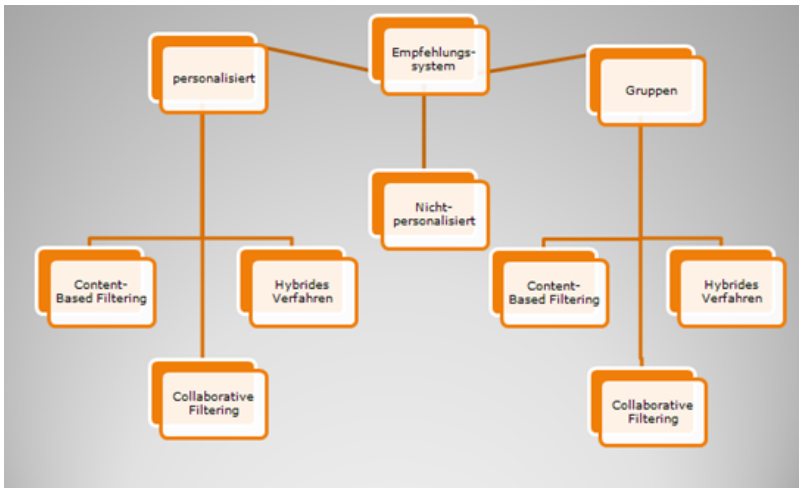


Abbildung 2.2: Arten einer Empfehlung

Wie in Abbildung 2.2 zu erkennen ist, ist es möglich, Empfehlungssysteme in drei Kategorien einzuteilen: Die personalisierten, die nicht-personalisierten und die Gruppen-Empfehlungen[Sto09].

Personalisierte Empfehlungen. Dagegen kommen bei personalisierten Empfehlungen komplexere Strukturen zum Einsatz. Am Anfang des Empfehlungsprozesses müssen Informationen über den Nutzer gesammelt werden. Die implizit oder explizit gesammelten Daten werden dann ausgewertet und es wird ein Benutzerprofil angelegt. Die Speicherung kann beispielsweise aufgrund eines vorangehenden Logins des Nutzers erfolgen, sodass das System ihn immer wiedererkennen kann.

Nicht-personalisierte Empfehlungen. Nicht-personalisierte Empfehlungen werden basierend auf sehr einfachen Methoden abgegeben. Jeder Nutzer erhält, unabhängig davon, welche Eigenschaften er aufweist, die gleichen Empfehlungen. Für den Nutzer ist es eine einfache Empfehlungsart, da ihm kaum Informationen abverlangt werden und das Empfehlungssystem, aufgrund bestimmter ermittelbarer Werte, wie zum Beispiel Geoinformationen, eine Empfehlung ausspricht.

Gruppen-Empfehlungen. Im Regelfall ist ein Empfehlungssystem für die Empfehlung an einzelne Personen zuständig. Jedoch gibt es in der Praxis auch Beispiele, in denen Gruppen von Menschen Adressaten von Empfehlungen sind. Der Empfehlungsablauf ist der gleiche wie bei personalisierten Empfehlungen, jedoch werden hier zusätzlich Informationen über die Gruppe gesammelt und der Nutzen, in Anlehnung an die Formel aus Kapitel 2.3.1, soll nicht nur für einzelne Personen, sondern für die Gruppe maximiert werden.

Empfehlungssystemtypen

Jedes Empfehlungssystem benötigt eine oder mehrere Informationsquellen. Diese Informationsquelle kann eine Gruppe von Nutzern sein, spezifisches Wissen über den Inhalt der vorzuschlagenden Elemente, eine Ontologie und ähnliches mehr. Daher gibt es verschiedene Ansätze, wie Empfehlungssysteme realisiert werden können.
Zu den wichtigsten Ansätzen gehören das kollaborative, das inhaltsbasierte und das wissensbasierte Filtern. Natürlich lassen sich auch verschiedene Ansätze kombinieren und somit hybride Filtermechanismen erzeugen.

Inhaltsbasiertes Filtern: Hierbei werden erst statische Eigenschaften (Inhalt) jeder Möglichkeit analysiert. Bei Texten kann es etwa die Häufigkeit der Vorkommen bestimmter Schlüsselworte sein, bei Bildern Farben und Formen. Nach der Eigenschaftsanalyse können einem Benutzer genau die Möglichkeiten empfohlen werden, deren statische Eigenschaften am besten zum aktuellen Kontext (z.B. dem Benutzerprofil) passen.

Der Vorteil dieser Methode ist, dass keine weiteren Menschen als Informationsquelle benötigt werden. Ein Nachteil allerdings besteht darin, dass zunächst eine Eigenschaftsanalyse, d. h. eine Inhaltsbeschreibung, stattgefunden haben muss.

Kollaboratives Filtern: Hierbei wird das Langzeitverhalten einer Menge von Benutzern analysiert, die aufgrund von bestimmten Eigenschaften miteinander zu Gruppen zusammengefasst werden. Diese Eigenschaften können zeitlich unabhängig sein und auf einem ähnlichen Verhalten basieren, oder zeitabhängig und beispielsweise die gleichzeitige Nutzung berücksichtigen. Werden Nutzer aufgrund der gleichzeitigen Nutzung eines Systems berücksichtigt, während der sie auch direkt oder indirekt interagieren, spricht man auch von einer ”Community”, um die aktive Zusammengehörigkeit einer eine Menge von Services parallel nutzenden Gruppe von Nutzern darzustellen. Es ist sinnvoll anzunehmen, dass Benutzer mit ähnlichen Interessen auch ähnliche Möglichkeiten wahrnehmen. Daher kann es sich als praktisch erweisen, einem Benutzer in einem Kontext gerade die Möglichkeiten zu empfehlen, die von anderen Benutzern im ähnlichen Kontext wahrgenommen wurden.

Der Vorteil einer solchen Herangehensweise ist, dass keine Information über den tatsächlichen Inhalt der Vorschläge vorliegen muss[MS12]. Der offensichtliche Nachteil dieser Herangehensweise besteht in den dafür notwendigen Datenmengen, die über eine genügend große Gruppe von Benutzern zur Verfügung

stehen müssen. Im Allgemeinen werden je mehr Daten benötigt, desto spezieller die Empfehlung ausfallen soll.

Wissensbasiertes Filtern: Hierbei geht man einen Schritt weiter und bezieht Expertenwissen/Ontologien mit ein. Somit können z.B. Entscheidungsbäume oder Bayes-Netze aufgebaut werden, um die Relevanz der Möglichkeiten in beliebigem Kontext zu inferieren.
Der Vorteil hierbei ist, dass im Regelfall eine gewisse Qualität durch das hinein geflossene Expertenwissen gesichert wird.
Der Nachteil ist die aufwendige Aufbereitung des nötigen Expertenwissens und die damit verbundene Trägheit bei der Reaktion auf Veränderungen.

Hybrides Filtern: Hierbei können alle Aspekte von inhaltsbasiertem Filtern, kollaborativem Filtern und wissensbasiertem Filtern einfließen. Die Ergebnisse der einzelnen Ansätze werden dann sinnvoll kombiniert, um zu einer Gesamtempfehlung zu gelangen.

2.3.3 Inhaltsbasiertes Filtern

Inhaltsbasiertes Filtern (engl. Content-Based Filtering) macht einen wesentlichen Teil der üblicherweise verwendeten Algorithmen zur Empfehlungsgenerierung aus [Jan+10]. In diesem Abschnitt werden die Grundlagen des inhaltsbasierten Empfehlen vorgestellt, ebenso wie einige Algorithmen, die in diesem Bereich Anwendung finden.

Inhalt

Um den Inhalt (die statischen Eigenschaften) jeder Möglichkeit zu analysieren, muss der Begriff des Inhalts formalisiert werden. Im Folgenden wird die Eigenschaftsanalyse am Beispiel von Texten betrachtet.
Diese Betrachtung ist sinnvoll, da zum einen Beschreibungen von Services textuell vorliegen und zum anderen die vorgestellten Methoden sich auf andere Eigenschaften von Services (z.B. Tags) übertragen lassen. Es gibt verschiedene Ansätze, den Inhalt eines Textes für eine algorithmische Auswertung zugänglich zu machen:

- Schlagwortanalyse:
  Schlagworte sind Worte, die den Text charakterisieren, aber nicht in ihm vorkommen müssen. Damit ist es sehr schwierig, Schlagworte algorithmisch aus einem gegebenen Text zu extrahieren. Folglich wird die Schlagwortanalyse meistens von Menschen durchgeführt und es existieren Standards für das Erstellen eines Schlagwortkatalogs [Rsw]. Das Ergebnis der Schlagwortanalyse eines Textes ist eine Liste seiner Schlagworte. Aufgrund der Schwierigkeit einer Schlagwortanalyse wird diese im Kontext von Services im Rahmen der PG nicht durchgeführt.

- Schlüsselwortanalyse:
  Schlüsselworte sind Worte, die den Text charakterisieren und gleichzeitig in ihm vorkommen. Das macht ihre algorithmische Bestimmbarkeit im Gegensatz zu Schlagworten wesentlich einfacher. Im nachfolgenden Abschnitt wird das TF-IDF-Verfahren vorgestellt, mit dessen Hilfe die Schlüsselwortanalyse durchgeführt werden kann. Durch die Einschränkung auf Schlüsselworte ist die Qualität, d.h. die Aussagekraft der Ergebnisse, der Schlüsselwortanalyse gegenüber der Schlagwortanalyse meist geringer. Gerade bei kurzen Texten ist diese Einschränkung schwerwiegend. Analog zur Schlagwortanalyse ist das Ergebnis der Schlüsselwortanalyse eines Textes eine Liste seiner Schlüsselworte. Eine Schlüsselwortanalyse lässt sich auf die Beschreibungen von Services anwenden.

- N-Gramm-Analyse:
  Ein N-Gramm ist eine Folge von N Zeichen. Die Zerlegung eines Textes in N-Gramme führt demnach zu einer Liste von Zeichenketten der Länge N. Um das Anfangs- und Endsegment nicht zu vernachlässigen, wird der Text mit Leerzeichen aufgefüllt. Zum Beispiel führt die Zerlegung des Wortes „Pentagondodekaeder“ in N-Gramme mit $N = 3$ (Trigramme) zu der Liste [␣P, ␣Pe, Pen, ent, nta, tag, ago, gon, ond, ndo, dod, ode, dek, eka, kae, aed, ede, der, er␣, r␣]. Das Ergebnis einer N-Gramm-Analyse ist die Häufigkeitsverteilung der N-Gramme. Eine N-Gramm-Analyse lässt sich ebenfalls auf die Beschreibungen von Services anwenden, jedoch wird dieser Ansatz zugunsten der Schlüsselwortanalyse im Rahmen der PG nicht verfolgt.

Die Unterschiede der Schlagwort- und Schlüsselwortanalyse lassen sich am folgenden Beispiel verdeutlichen. Man betrachte die beiden Sätze „Vertrauen ist gut, Kontrolle ist besser“ und „Das Vorurteil ist von der Wahrheit weiter entfernt als die Unkenntnis“. Die Schlüsselwortanalyse könnte zum ersten Satz die Liste [Vertrauen, Kontrolle] und zum zweiten [Vorurteil, Wahrheit, Unkenntnis] liefern. Damit wären diese beiden Sätze kaum ähnlich. Bei der Schlagwortanalyse wäre hingegen bedeutsam, dass beide Sätze übersetzte Zitate von Wladimir I. Lenin sind. Damit könnten die Ergebnisse der Schlagwortanalyse [Vertrauen, Kontrolle, Zitat, Lenin] und [Vorurteil, Wahrheit, Unkenntnis, Zitat, Lenin] sein und damit einander sehr ähnlich. Die Ähnlichkeit von zwei Elementen lässt sich also nicht immer ausschließlich über ihren textuellen Inhalt bestimmten, sondern es müssen teilweise auch Meta-Eigenschaften mit berücksichtigt werden. In einem solchen Fall wäre die Qualität der Schlagwortanalyse deutlich besser.

Der Begriff des Inhalts lässt sich noch weiter formalisieren.
Seien dazu $\mathcal{I}$ die Menge aller einem Benutzer zur Verfügung stehenden Möglichkeiten (Items) (z.B. Texte) und $\mathcal{F} = \{f_1, \ldots, f_l\}$ die Menge aller möglichen Eigenschaften (Features) der Elemente aus $\mathcal{I}$.
Der Inhalt jedes Elements $i \in \mathcal{I}$ lässt sich als Vektor $v \in \mathbb{Q}^{\mathcal{F}}$ darstellen. Da $\mathcal{F}$ in der Regel eine große Kardinalität hat, sind die den Inhalt charakterisierenden Vektoren meistens spärlich besetzt. Die Darstellung als Vektor erlaubt eine natürli-

che Beschreibung der Distanz zweier Inhalte, die z.B. durch die euklidische Distanz der entsprechenden Vektoren beschrieben werden kann. Die tatsächlichen Werte der Komponenten von v können dabei je nach Ansatz unterschiedlich berechnet werden. Drei dieser Ansätze werden nun am Beispiel des Textes „Lernen, lernen und nochmals lernen.“ (Zitat von Wladimir I. Lenin) betrachtet.
Die Menge $\mathcal{F}$ aller Eigenschaften kann in diesem Fall die Menge aller möglichen Schlag- oder Schlüsselworte sein
$\mathcal{F} = \{Aal, \ldots, Bildung, \ldots, Lernen, \ldots, Zebra\}$.

- Der erste Ansatz schränkt den Vektorraum auf $\{0,1\}^{\mathcal{F}} \subset \mathbb{Q}^{\mathcal{F}}$ ein, indem jede Komponente des Eigenschaftsvektors v auf 1 gesetzt wird, falls das Wort im Text vorkommt, und anderenfalls auf 0. Diese Darstellung ist isomorph zu der Darstellung des Textes als die Menge (gleiche Worte fallen zusammen) der Worte, die in ihm vorkommen.

- Der zweite Ansatz schränkt den Vektorraum auf $\mathbb{N}^{\mathcal{F}} \subset \mathbb{Q}^{\mathcal{F}}$ ein, indem jede Komponente des Eigenschaftsvektors v auf die Anzahl der Vorkommen des entsprechenden Wortes gesetzt wird. Diese Darstellung ist isomorph zu der Darstellung des Textes als Multimenge (gleiche Worte fallen nicht zusammen) der Worte, die in ihm vorkommen.

- Der dritte Ansatz schreibt jeder Komponente ein Gewicht zu, das der Relevanz des entsprechenden Wortes entsprechen soll. Diese Darstellung ist vergleichbar mit sog. „Fuzzy-Mengen“, in welchen Elemente „ein bisschen“ vorkommen können.

Mögliche Eigenschaftsvektoren dieser Ansätze sind in der Abbildung 2.3 beschrieben.

$Worte$	$Eigenschaftsvektor$		
	$Ansatz1$	$Ansatz2$	$Ansatz3$
$Aal:$	$\begin{pmatrix} 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ \vdots \\ 0 \\ \vdots \\ 3 \\ \vdots \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ \vdots \\ 0.7 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$
$\vdots$			
$Bildung:$			
$\vdots$			
$Lernen:$			
$\vdots$			
$Zebra:$			

Abbildung 2.3: Eigenschaftsvektoren

Es ist erkennbar, dass der dritte Ansatz mindestens so viel Information beinhaltet, wie der zweite und der zweite wiederum mindestens so viel, wie der erste. Offensichtlich ist der Vorteil eines einfacheren Ansatzes bereits seine Einfachheit und die Möglichkeit einer kompakten Repräsentation bzw. schnelleren Auswertung.

Der Nachteil eines einfacheren Ansatzes hingegen ist der Mangel an Präzision und der damit verbundene Qualitätsverlust.

TF-IDF-Verfahren

Einen algorithmischen Ansatz, die Schlüsselwortanalyse durchzuführen, stellt das TF-IDF-Verfahren [Jan+10] dar. Dabei wird einem Schlüsselwort eine größere Relevanz zugeordnet, wenn es oft im Text vorkommt (TF: Term Frequency), und eine niedrigere, wenn es in vielen anderen Texten vorkommt (IDF: Inverse Document Frequency). Als Ausgangspunkt wird ein Korpus benötigt, der aus einer genügend großen Menge verschiedener Texte besteht. Für deutsche Texte kann z.B. das 'Deutsche Referenzkorpus' vom Institut für Deutsche Sprache [Ids] verwendet werden.

Algorithmus 1 *TF-IDF-Verfahren*
Eingabe: *Korpus* $C = \{T_1, \ldots, T_n\}$, *Text* T, *Wort* $w \in T$
Definiere:
$N(T, w)$ = *Anzahl der Vorkommen des Wortes* w *im Text* T
$I(C, w)$ = *Anzahl der Texte im Korpus* C *mit mindestens einem Vorkommen des Wortes* w

TF: Term Frequency
$TF(T, w) = \frac{N(T,w)}{\max\{N(T,k) | k \text{ ist Schlüsselwort}\}}$

IDF: Inverse Document Frequency
$IDF(C, w) = \log(\frac{|C|}{I(C,w)})$

R: Relevanz
$R(C, T, w) = TF(T, w) * IDF(C, w)$

Ausgabe: *Relevanz* $R(C, T, w)$

Die Zuordnung eines Textes T zu seinem Inhaltsvektor v erfolgt komponentenweise mit Hilfe des TF-IDF-Verfahrens.
Für ein Wort w ist $v_w = \begin{cases} R(C, T, w) & \text{falls } w \in T \\ 0 & \text{sonst} \end{cases}$

Das TF-IDF-Verfahren kann durch mehrere Techniken weiter verbessert werden ([Jan+10], [Kla09]):

- Stemming:
  Vor dem Einsatz des TF-IDF-Verfahrens können sämtliche Worte auf ihre

Wortstämme reduziert werden. Dadurch fallen semantisch gleiche bzw. ähnliche Worte zusammen und die Dimension der Inhaltsvektoren sinkt. Die Qualität der Analyse steigt.

- Stoppwörter:
  Vor dem Einsatz des TF-IDF-Verfahrens können sämtliche Worte wie „und, auch, der, ...“ aus den Texten entfernt werden, da sie i.d.R. kaum Schlag- oder Schlüsselworte ausmachen. Dadurch sinkt die Dimension der Inhaltsvektoren. Die Qualität der Analyse bleibt erhalten.

- Charakteristika-Auswahl:
  Vor dem Einsatz des TF-IDF-Verfahrens wird eine feste Auswahl an Schlüsselworten getroffen und nur diese werden im Verfahren berücksichtigt. Dadurch sinkt die Dimension der Inhaltsvektoren dramatisch, aber es kann passieren, dass durch eine unglückliche Vorauswahl relevante Schlüsselworte nicht berücksichtigt werden und damit die Qualität der Inhaltsvektoren sinkt. Eine Möglichkeit, eine automatische *Charakteristika-Auswahl* vorzunehmen, ist es, Worte mit einer zu großen oder zu geringen *Term Frequency* nicht in der Analyse zu berücksichtigen. Falls die Information über die Relevanz der Texte einer Trainingsmenge vorliegt, lässt sich mit Hilfe des χ^2 Tests für Unabhängigkeit eine Menge der Schlüsselworte bestimmen, sodass das Vorkommen oder Fehlen der entsprechenden Schlüsselworte im Zusammenhang mit der Relevanz der Texte steht [Jan+10].

- Thesauri:
  Thesauri sind Sammlungen von Begriffen mit einer Beschreibung ihrer Beziehungen zueinander. Sie können dazu benutzt werden, um z.B. Synonyme zu finden. Dadurch können semantisch ähnliche Worte zu einem virtuellen Wort zusammengefasst und die Dimension der Inhaltsvektoren gesenkt werden.

- Phrasen:
  Manchmal ist es sinnvoll, nicht einzelne Worte, sondern Phrasen zu betrachten. So sind die Worte „Europäische“ und „Union“ einzeln nicht dasselbe wie ihre Verbindung zu „Europäische Union“. Deshalb kann es hilfreich sein, für bestimmte Phrasen virtuelle Worte einzuführen.

- WSD (Word Sense Disambiguation, Unterscheidung nach Semantik des Wortes):
  Manche Wörter wie „Bank“ haben mehrere Bedeutungen. Um die Polysemie (Mehrdeutigkeit) aufzulösen, kann es ausreichen, die Umgebung des entsprechenden Wortes zu betrachten. Falls ein Wort an zwei Stellen verschiedene Bedeutungen hat, sollte dies der Inhaltsvektor wiedergeben. So können z.B. virtuelle Worte für die jeweilige Bedeutung eingefügt und gezählt werden. Die Dimension der Inhaltsvektoren steigt dabei, allerdings steigt somit auch die Qualität der Analyse.

- Kollokationen:
  Bestimmte Worte kommen signifikant oft gemeinsam vor. Diese Tatsache wird

als Kollokation bezeichnet und erlaubt eine algorithmische Gruppierung von Worten. Kollokationen erlauben, ähnlich wie Synonyme und Assoziationen, einerseits Worte zusammenzufassen und andererseits Polysemie aufzulösen.

- NER (Named Entity Recognition (Eigennamenerkennung)):
  Wenn im Text Entitäten (Personen oder Firmen mit Eigennamen) vorkommen, die durch einen Überbegriff zusammengefasst werden können, dann ist es sinnvoll diesen Überbegriff als ein virtuelles Wort einzuführen und diese Entitäten damit zu erfassen. Beispielsweise könnte der Name „Richard Stallman“ dem Überbegriff „Free Software Foundation“ zugeordnet werden. Damit würde Ähnlichkeit zu anderen Texten aufgebaut werden, die diesen Namen zwar nicht nennen, aber die damit verbundenen Projekte.

Ähnlichkeit

Ausgehend von der Formalisierung des Begriffes *Inhalt* wird nun auch der Begriff *Ähnlichkeit* formalisiert.

Für eine Menge S heißt die Abbildung $sim\colon S \times S \to \mathbb{R}$ *Ähnlichkeitsmaß*[Jan+10], falls sie folgende Axiome erfüllt:

- Nichtnegativität: $\forall i, j \in S\colon 0 \leq sim(i, j)$
- Symmetrie: $\forall i, j \in S\colon sim(i, j) = sim(j, i)$
- Beschränktheit: $\forall i, j \in S\colon sim(i, j) \leq sim(i, i) = 1$

Im Bereich der Empfehlungssysteme werden meistens nicht alle dieser Axiome benötigt/verlangt.

Im Folgenden werden Ähnlichkeitsmaße, die durch diverse sogenannte Koeffizienten beschrieben werden, auf dem Eigenschaftsraum betrachtet. In der Literatur scheint kein „bestes“ Ähnlichkeitsmaß zu existieren und je nach Einsatzgebiet eignen sich die verschiedenen Maße unterschiedlich gut zur Beschreibung der Ähnlichkeit. Bezeichne weiterhin $\mathcal{I}$ die Menge aller einem Benutzer zur Verfügung stehenden Möglichkeiten (Items bzw. Entitäten), $\mathcal{F} = \{f_1, \ldots, f_l\}$ die Menge aller möglichen Eigenschaften (Features) der Elemente aus $\mathcal{I}$, und dementsprechend $\mathbb{Q}^{\mathcal{F}}$ den Eigenschaftsraum der Möglichkeiten.

Cosinus. Eine Möglichkeit ist es, den Eigenschaftsraum als Vektorraum aufzufassen und damit über den Winkel zwischen zwei charakterisierenden Vektoren eine Aussage über ihre Ähnlichkeit zu treffen [Kla09]. Je kleiner der Winkel, desto ähnlicher sind die Vektoren und damit die dazugehörigen Items.

Definition 1 *Cosinus-Similarity*
$\forall v, w \in \mathbb{Q}^{\mathcal{F}}$
$sim_{\cos}(v, w) = \frac{v \cdot w}{|v| * |w|} = \frac{\sum\limits_{i \in \mathcal{F}} v_i * w_i}{\sqrt{\sum\limits_{i \in \mathcal{F}} v_i^2} * \sqrt{\sum\limits_{i \in \mathcal{F}} w_i^2}}$

$\forall v, w \in \mathcal{P}(\mathcal{F})$
$sim_{\cos}(v, w) = \frac{|v \cap w|}{\sqrt{|v| * |w|}}$

Es existieren noch weitere Koeffizienten [Kla09] (Pearson Korrelationskoeffizient, Dice-Koeffizient, Overlap-Koeffizient, Jaccard-Koeffizient). Im Rahmen der PG wird nur die Cosinus-Similarity betrachtet.

Mutual Information

In der Wahrscheinlichkeitstheorie existiert eine Formalisierung für den Zusammenhang zweier Zufallsgrößen. Diese wird als gegenseitige Information (*Mutual Information* [KI10]) bezeichnet und kann dazu verwendet werden, Aussagen über Ähnlichkeit zu treffen. Ein häufiges gemeinsames Vorkommen von zwei Merkmalen deutet darauf hin, dass sie sich auf gemeinsame Sachverhalte beziehen und daher einen ähnlichen Kontext anzeigen. Deshalb liegt es nahe, eine semantische Verbindung zwischen diesen Merkmalen zu vermuten.

Sei:
$p(i)$ = Wahrscheinlichkeit des Vorkommens des Merkmals $i \in \mathcal{F}$
$p(i, j)$ = Wahrscheinlichkeit des gemeinsamen Vorkommens der Merkmale $i, j \in \mathcal{F}$

Definition 2 *Mutual Information*
$MI(i, j) = \begin{cases} ld(\frac{p(i,j)}{p(i)*p(j)}), & \textit{wenn } i \neq j \\ 1, & \textit{sonst} \end{cases}$
ld (logarithmus dualis) ist dabei der Logarithmus zur Basis 2.

Es gilt:
i, j unabhängig $\Rightarrow p(i, j) = p(i) * p(j) \Rightarrow MI(i, j) = 0$
i, j kookkurrent $\Rightarrow p(i, j) > p(i) * p(j) \Rightarrow MI(i, j) > 0$
Je größer also der Wert der *Mutual Information* ist, desto kookkurrenter sind zwei

Merkmale. Deshalb liegt bei einem größeren Wert der *Mutual Information* die Vermutung nahe, dass zwischen den betrachteten Merkmalen eine semantische Verbindung besteht.

Definition 3 *Mutual Information Koeffizient*
$\forall v, w \in \mathbb{Q}^{\mathcal{F}}$
$sim_{mi}(v, w) = \sum_{i,j \in \mathcal{F}} MI(v_i, w_j)$

Nearest Neighbours

Um eine Möglichkeit zu bewerten, ist es sinnvoll zu berücksichtigen, ob ähnliche Möglichkeiten bereits bewertet wurden. Dazu werden *nächste Nachbarn* und ihre Bewertungen betrachtet.

Definition 4 *Nearest Neighbours*
Sei $v \in \mathbb{Q}^{\mathcal{F}}$, $k \in \mathbb{N}$,
eine Menge $N = \{i_1, \dots i_k\} \subseteq \mathcal{I}$ *heißt k-Nearest-Neighbours von v, falls*
$\forall x \in (\mathcal{I} \setminus N), \forall y \in N \colon d(v, y) \leq d(v, x)$, *wobei d der euklidische Abstand ist.*

Es ist auch möglich, die Menge der k-Nearest-Neighbours als Empfehlungsgrundlage zu nutzen. Falls sich das Nutzerprofil als $v \in \mathbb{Q}^{\mathcal{F}}$ repräsentieren lässt, wäre es sinnvoll erst die k-Nearest-Neighbours zu bestimmen, um dann aus ihnen Elemente zu empfehlen.
Darüber hinaus kann an Stelle des euklidischen Abstands auch ein beliebiges Ähnlichkeitsmaß verwendet werden. Dabei gilt es zu beachten, dass im Gegensatz zu der euklidischen Distanz, ein größerer Wert des Ähnlichkeitsmaßes gleichzeitig eine größere Nähe bedeutet.

Relevance Feedback (Rocchio)

Falls der Benutzer eine Möglichkeit hat (explizit oder implizit) die ihm vorgeschlagenen Elemente zu bewerten, lässt sich die Empfehlung iterativ verfeinern. Ein Beispiel dafür ist der *Relevance Feedback Algorithmus (Rocchio)* [Jan+10]. Sobald die Information vorliegt, welche der Empfehlungen der Benutzer als relevant (D^+) und welche als irrelevant (D^-) ansieht, lässt sich im Eigenschaftsraum jeweils ein Prototyp dieser Mengen bestimmen (Zentroide d^+ (von D^+) und d^- (von D^-)). Anschließend kann die Empfehlungsgrundlage im Eigenschaftsraum gewichtet von d^- weg und zu

d^+ hin verschoben werden, in der Hoffnung auf diese Weise mehr relevante Elemente zu finden.

Dabei spielt die Gewichtung der Verschiebung eine große Rolle und lässt sich an drei Parametern festlegen:

- α beschreibt, wie stark die vorangegangene Suche zu berücksichtigen ist.
- β beschreibt, wie stark die positiv bewerteten Elemente zu berücksichtigen sind.
- γ beschreibt, wie stark die negativ bewerteten Elemente zu berücksichtigen sind.

In der Literatur werden die Werte $\alpha = 1$, $\beta = 2$, $\gamma = 0.25$ empfohlen [Jan+10].

Algorithmus 2 *Relevance Feedback (Rocchio)*
Eingabe: *Benutzerprofil.*

1. *Setze* $Q :=$ *Benutzerprofil* $\in \mathbb{Q}^{\mathcal{F}}$.
2. *Bestimme Empfehlungen auf der Grundlage des Punktes* Q *(z.B. durch Nearest Neighbours oder Klassifizierung).*
3. *Lasse den Benutzer beliebig viele Empfehlungen bewerten (relevant/ irrelevant).*
4. *Unterteile bewertete Elemente in Mengen* D^+ *(relevant) und* D^- *(irrelevant).*
5. *Bestimme Zentroide* d^+ *(von* D^+*) und* d^- *(von* D^-*).*
6. *Setze* $Q := \alpha * Q + \beta * d^+ - \gamma * d^-$.
7. *Falls es Änderungen gab, gehe zu 2.*
8. ***Ausgabe:*** *Aktuelle Empfehlungen auf der Grundlage des Punktes* Q.

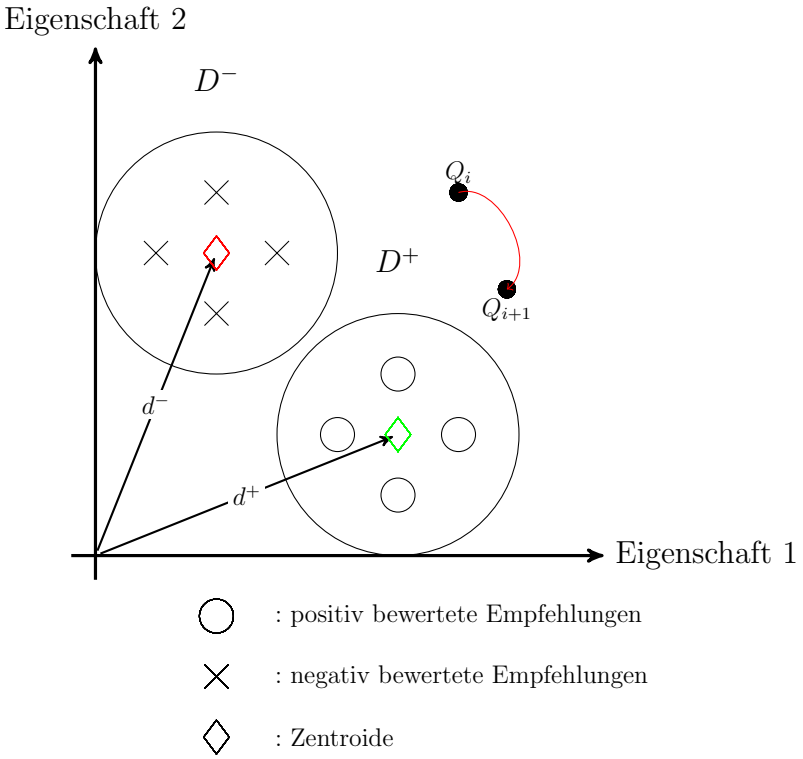


Abbildung 2.4: Veranschaulichung einer Iteration des Relevance Feedback Algorithmus.

Der Punkt Q_i repräsentiert den Punkt Q vor Schritt 6 und der Punkt Q_{i+1} repräsentiert den Punkt Q nach Schritt 6. Die Bezeichnungen d^+, D^+, d^-, D^- repräsentieren die entsprechenden Größen in Schritten 4 und 5.

2.3.4 Kollaboratives Filtern

Das gruppenbasierte Empfehlen kommt im Gegensatz zum inhaltsbasierten Filtern ohne genaue Kenntnisse der beschriebenen oder empfohlenen Entitäten aus, beruht dafür aber gänzlich auf Informationen über die Nutzung der Entitäten durch Anwender oder Verbraucher. Pars pro Toto stellen wir das Association RuleMining vor.

Association Rule Mining

Untersucht man das Einkaufsverhalten von Konsumenten, so fällt auf, dass die Ereignisse „Kauf von Artikel i“ und „Kauf von Artikel j“ keine voneinander stochastisch unabhängigen Ereignisse sind. Es ist viel mehr so, dass bestimmte Warenkörbe besonders oft erworben werden. Das Ziel des *Association Rule Mining* ist es, aus einer Menge von gekauften Artikeln bestimmte Regeln abzuleiten, welche mit einer gewissen Wahrscheinlichkeit gelten. Ein Beispiel für eine solche Regel ist „Kauft jemand Katzenfutter, so kauft er mit 70% Wahrscheinlichkeit auch Katzenstreu“.

Im Folgenden werden solche Regeln formalisiert.

Definition 5 *Entität, Transaktion, Assoziationsregel*
Eine Entität *ist je nach Kontext ein gekaufter Gegenstand, ein Service, etc.*

Eine Transaktion *ist eine Menge von* Entitäten*, die im Kontext der Einkaufsverhaltensanalyse zusammen gekauft wurden.*

Seien X, Y Mengen von Entitäten. Dann bedeutet die Assoziationsregel *$X \to Y$, dass falls Entitäten der Menge X in einer Transaktion enthalten sind, es auch wahrscheinlich ist, dass Entitäten der Menge Y in ihr enthalten sind.*

Im Folgenden werden Maße vorgestellt, die zur Berechnung von Empfehlungen benutzt werden:

Definition 6 *Support, Confidence*
Sei D eine Menge von Transaktionen*, X eine Menge von* Entitäten *und $X \to Y$ eine* Assoziationsregel*.*

$$Support\ (X) = \frac{|\{T \in D | X \subseteq T\}|}{|D|}$$

$$Support\ (X \to Y) = Support\ (X \cup Y)$$

$$Confidence\ (X \to Y) = \frac{Support(X \to Y)}{Support(X)}$$

Der Support misst den Anteil der *Transaktionen* der Datenbasis, in denen alle Artikel in X und Y vorkommen. Dieses Maß misst also nicht die stärke der Implikation, sondern ihre Relevanz bezüglich der betrachteten Datenbasis. Indem Wert auf ein gewisses Maß an Support gelegt wird (*minimaler Support*), werden irrelevante Schlussregeln ausgeschlossen, die nicht oft genug vertreten sind.
Die *Confidence* ist analog zu einer bedingten Wahrscheinlichkeit definiert und misst die Stärke der Schlussregel. Umgangssprachlich bedeutet *Confidence*, wie wahrscheinlich der Kauf von allen Artikeln aus X und Y ist, wenn sichergestellt ist, dass alle Artikel aus X in der *Transaktion* vorkommen. Regeln mit einer zu geringen *Confidence* (*minimale Confidence*) werden ebenfalls als irrelevant angesehen.
Um aus einer vorhandenen Datenbasis *Assoziationsregeln* mit einem *minimalen Support* und einer *minimalen Confidence* effizient zu extrahieren, kann der *Apriori-Algorithmus* [KI10] angewandt werden.

Algorithmus 3 *Apriori-Algorithmus*
Eingabe: Datenbasis D von Transaktionen*, Menge I aller* Entitäten*, minimaler Support minSup, minimale Confidence minConf*

1. *Setze* $L_1 := \{\{x\} \mid x \in I, \mathit{Support}(\{x\}) \geq \mathit{minSup}\}$, $k := 2$
2. *Solange* $L_{k-1} \neq \emptyset$
3. *Setze* $C_k := \{X \cup x \mid X \in L_{k-1}, x \in (\bigcup L_{k-1}) \setminus X\}$
4. *Setze* $C_k^{sub} :=$
   $\{X \in C_k \mid \forall Y \subseteq X : |Y| = k-1 \Rightarrow Y \in L_{k-1}\}$
5. *Setze* $L_k := \{X \in C_k^{sub} \mid \mathit{Support}(X) \geq \mathit{minSup}\}$
6. *Setze* $k := k + 1$
7. *Setze* $L := \bigcup_k L_k$
8. *Ausgabe:* Assoziationsregeln
   $\{X \to Y \mid X \cup Y \in L, \mathit{Confidence}(X \to Y) \geq \mathit{minConf}\}$.

Ein konkreter Recommender kann beispielsweise so vorgehen, dass er für einen konkreten Nutzer, der bereits die Menge X gekauft hat, über alle einelementigen Mengen Y iteriert und dabei alle Regeln der Form $X \to Y$ beibehält, deren *Support* einen vorher definierten *minimalen Support* überschreitet. Anschließend die ausgewählten Regeln absteigend nach *Confidence* sortiert und die in je einem Y enthaltenen Artikel in dieser Reihenfolge vorschlägt.
Association Rule Mining zeichnet sich durch eine valide Grundidee aus und ist im Bereich der Filmempfehlungen auch praktisch erprobt.

2.3.5 Evaluation von Empfehlungssystemen

Die Evaluation von Empfehlungssystemen ist in der Praxis schwierig. Einerseits ist die tatsächliche Relevanz von empfohlenen Elementen ein eher theoretischer und meist subjektiver Wert. Andererseits ist es meist noch schwieriger, die Menge der tatsächlich relevanten Elemente überhaupt erst zu bestimmen. Falls Schätzungen dieser Größen gemacht werden können, bietet das Gebiet des *Information Retrieval* einige Grundbegriffe, um die Güte eines Empfehlungssystems zu beschrieben [Jan+10].

Definition 7 *Precision, Recall*
Sei:
Elements = Menge aller Elemente
Relevant = Menge aller relevanten Elemente (subjektiv)
Recommended = Menge aller vorgeschlagenen Elemente

Definiere:
Precision $p = \frac{|Relevant \cap Recommended|}{|Recommended|}$
Recall $r = \frac{|Relevant \cap Recommended|}{|Relevant|}$

Am besten lassen sich diese Definitionen anhand eines Mengendiagramms veranschaulichen. Precision beschreibt, wie sich die Menge „gut“ zu der Menge *Recommended* verhält, und damit wie klein der Bereich „zu viel“ ist. Recall beschreibt, wie sich die Menge „gut“ zu der Menge *Relevant* verhält, und damit wie klein der Bereich „zu wenig“ ist.

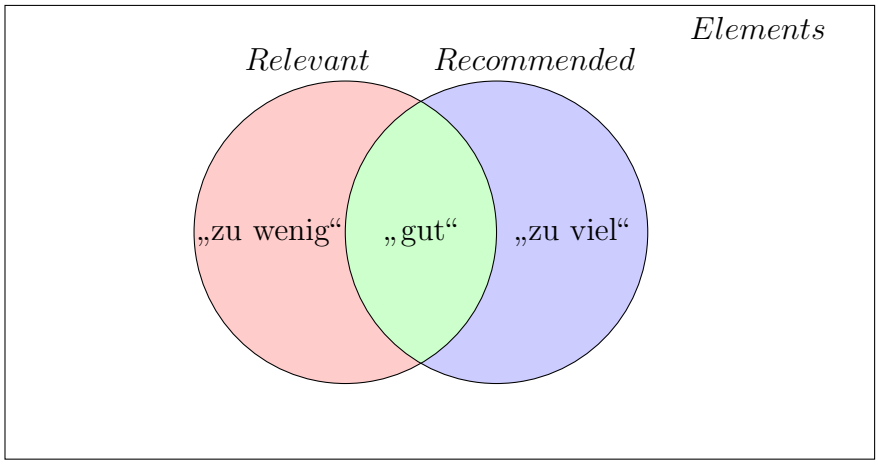


Abbildung 2.5: Mengendiagramm für Precision, Recall

Precision wird als ein Maß für die Güte der Empfehlung angesehen. Je näher Precision an 1 ist, desto geringer ist der Anteil der irrelevanten Empfehlungen.
Recall wird als ein Maß für die Gründlichkeit der Suche angesehen. Je näher Recall an 1 ist, desto mehr relevante Elemente werden tatsächlich vorgeschlagen.
Es ist wichtig beide Maße gleichzeitig zu betrachten, denn es ist in der Regel einfach, einen der beiden Werte auf Kosten des anderen zu maximieren. Wenn z.B. alle Elemente vorgeschlagen werden, werden selbstverständlich auch zugleich alle relevanten Elemente vorgeschlagen und obwohl $Recall = 1$ gilt, ist so eine Empfehlung unbrauchbar. Hingegen, wenn ein offensichtlich relevantes Element existiert und nur dieses eine Element vorgeschlagen wird, gilt zwar $Precision = 1$, aber wenn es noch weitere relevante(re) Elemente gibt, ist so eine Empfehlung meistens auch kaum brauchbar.

3 Entwurf und Implementierung

Nachdem nun sämtliche theoretischen Grundlagen geklärt sind, wenden wir uns der Praxis zu. In diesem Kapitel soll vorgestellt werden, wie der Entwurf und die Implementierung unseres Systems abgelaufen sind.

3.1 Grundlegende Konzepte

Gegenstand dieses Abschnitts ist die technische Umsetzung einiger wichtiger Entitäten, die in Kapitel 2 vorgestellt wurden. Dazu gehört insbesondere der Aspekt der einheitlichen Repräsentation und Verwaltung der aus den diversen Datenquellen (vgl. Abschnitt 2.2) importierten Service-, Workflow- und Ontologiebeschreibungen. Das in diesem Zuge entwickelte Metamodell wird mitsamt der Entscheidungen, die zu seiner Konzeption führten, ebenso erläutert wie die Darstellung und Berechnung von Empfehlungen.

3.1.1 Metamodell

Die Vielfalt der Quellen, mit denen das Empfehlungssystem umgehen muss, stellte eine Herausforderung bei der Modellierung eines einheitlichen Modells dar. Denn die verschiedenen Datenquellen haben alle ihre eigenen (zum Teil proprietären, zum Teil nur unzureichend oder gar nicht dokumentierten) Datenmodelle, sodass eine einheitliche Verarbeitung nicht ohne Weiteres möglich ist.
Die Vielfalt der Quellen aber ist ein unverzichtbarer Vorteil, denn je mehr Daten die Algorithmen zur Verfügung haben, umso qualitativ hochwertigere Empfehlungen können sie abgeben.

Anforderungen an das Metamodell

Die wichtigste Aufgabe des Metamodells ist es somit, die vielen verschiedenen Datenquellen und ihre Formate in einem Modell zu vereinen, so dass möglichst keine

Daten verloren gehen. Nur solche Daten, die für die von uns gewählten Empfehlungs-Algorithmen relevant sind, müssen jedoch unbedingt übernommen werden, während sonstige Daten zum Behufe besserer Übersicht auch vernachlässigt werden können. Aus den als relevant ermittelten Daten wird eine Datenstruktur erstellt, die von sämtlichen Algorithmen verwendet werden kann.
Zur Erstellung dieses Datenmodells haben wir uns zu Gunsten möglichst vieler verfügbarer Informationen dazu entschieden, auch solche Eigenschaften zu integrieren, die nur aus einzelnen Quellen zu gewinnen waren. In Kauf nehmend, dass beispielsweise einige Services gewisse Eigenschaften in keiner Ausprägung aufweisen, haben wir uns für diese Variante entschieden. Der entscheidende Vorteil liegt darin, dass Informationen genutzt werden können, statt sie zu verwerfen; insbesondere, da die Schnittmenge der aus jeder Quelle vorhandenen Daten nicht ausreichend groß war.

Metamodell des Lehrstuhls für Programmiersysteme

Für ein aktuelles Forschungsprojekt hatte der Lehrstuhl 5 der hiesigen Fakultät bereits ein Metamodell zur Modellierung von Services fertiggestellt. Ferner sollte dieses Modell sich auch zur Darstellung, zum Entwurf und letztlich zur Ausführung von Workflows eignen.
Um eventuell die Benutzeroberfläche dieses Projektes nutzen zu können oder sogar das zu entwickelnde Recommender-System in das Projekt integrieren zu können, wurde geprüft, inwiefern dieses bereits bestehende Metamodell im Kontext der oben skizzierten Aufgabenstellung nutzbar ist. Die Anwendbarkeit des Metamodells hätte zur Folge, dass ein bereits hinreichend geprüftes Modell genutzt werden könnte und natürlich die leichtere Integration in eine GUI auf Grund des gleichen Metamodells.
Das Modell des Lehrstuhls enthielt bereits alle funktionalen Strukturen um Services abzubilden. Es separierte die Schnittstellen zu und von einem Service in ihre semantischen Verzweigungen (Branches) und deren syntaktischen Enden (Ein- und Ausgabe). Des Weiteren hatte das Modell bereits Strukturen, um eine gesamte Bibliothek an Services zu verwalten und auch taxonomische Beziehungen zwischen den Services festzuhalten. Auch war es möglich, aus mehreren Services einen übergeordneten Service bzw. einen Workflow zu konstruieren.
Diese vielfältigen Möglichkeiten ergaben sich aus der Tatsache, dass dieses Modell für allgemeine Services ausgelegt war und daher zu viel Overhead für die Daten aus den Bioinformatik-Portalen bot. Außerdem wurde auch viel Wert auf die mögliche Ausführung von Workflows gelegt, was für unser Projekt jedoch nicht von Relevanz ist. Gleichzeitig ging das Modell aufgrund seiner Ausrichtung auch nicht auf die speziellen Daten der von uns betrachteten Portale ein, so dass diese nicht effizient genutzt werden konnten. In den Besprechungen kristallisierte sich der Konsens heraus, dass es aufwendiger ist dieses Datenmodell umzugestalten, als ein neues zu entwickeln.

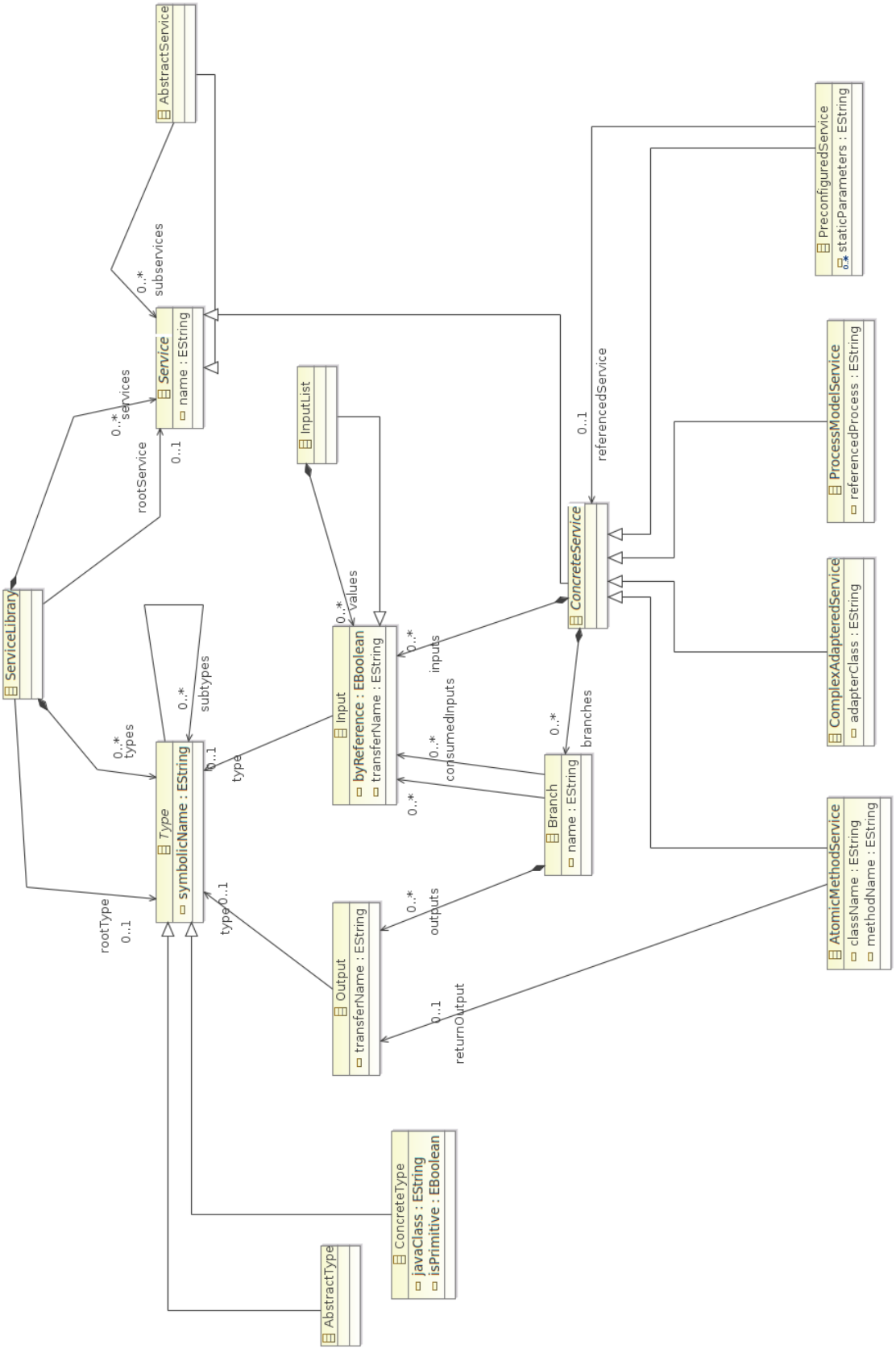


Abbildung 3.1: Metamodell des Lehrstuhls

Metamodell

Das erste Metamodell (siehe Abbildung 3.3) war nur ein rudimentärer Entwurf. Hier wurden lediglich die Informationen aus den Datenquellen zusammengetragen und in Relation zueinander gesetzt. Das Metamodell des Lehrstuhls (siehe Abbildung 3.1) diente hier zwar als lose Diskussionsgrundlage, wurde aber verworfen, da es sich auf Aspekte der Ausführung von Workflows konzentrierte. Der Zusätzliche overhead, der mit zu verwalten wäre und auch der Aufwand um dieses Modell den Bedürfnissen der Zielsetzung anzupassen hätten einen zu schlechten Kosten-Nutzen-Faktor.
Zum Zeitpunkt des ersten Entwurfs war auch noch nicht geklärt, wie die zusammengetragenen Daten persistent gespeichert werden sollten. Daher zählte ein Entity-Relationship-Modell (siehe Abbildung 3.2), welches lediglich alle nötigen Informationen darstellte, zu den Vorlagen des ersten Entwurfs.
Das Konzept des Metamodells setzte sich aber durch, denn es diente nicht nur zu der Modellierung der Datenstruktur, durch den in EMF enthaltenen Code-Generator, der im Abschnitt 3.2.2 noch ausführlicher erklärt wird, kann ein Java-Modell direkt aus dem Metamodell erzeugt werden.
Während der Implementierung von einzelnen Komponenten, wie den Importmodulen wurden jedoche kleinere Schwächen in der Modellierung aufgedeckt, welche während der Planung noch nicht in dieser Form ersichtlich waren. So wurde das Metamodell in den Besprechungen immer wieder thematisiert. Dabei wurden diese Schwachstellen aufgegriffen, in der Gruppe diskutiert und analysiert und bei Bedarf angepasst. Durch diese qualitative Verbesserung des Metamodells entstand das „finale Metamodell“.

Finales Metamodell

Das finale Metamodell (siehe Abbildung 3.4) wurde von der PG als Alternative zum vom Lehrstuhl vorgeschlagenen Modell entwickelt. Bei unserer Modellierung lag das Augenmerk eher auf der effizienten Nutzung der Datenstruktur und nicht auf ihrer Allgemeingültigkeit und berücksichtigt den Import von Workflows und Services aus verschiedenen Quellen.
Im Folgenden werden einzelne Elemente des Modells näher erläutert:

Service Jedem Service wird eine ID zugeordnet, sodass der Name nicht alleiniger Identifizierer sein muss (in den Daten kommen ähnlich oder gleich benannte Services häufig vor). Darüber hinaus enthält der Service selbst Beschreibung und eventuelle Kommentare, während seine Parameter als wesentlich für das Empfehlungssystem relevante Komponenten ausgelagert sind.

ServiceResourceDescription Dieser Teil des Modells definiert die Quelle, aus welcher der Service stammt.

ServiceLibrary Diese Struktur definiert die organisatorische Einheit, welche die Gesamtheit aller Services verwaltet.

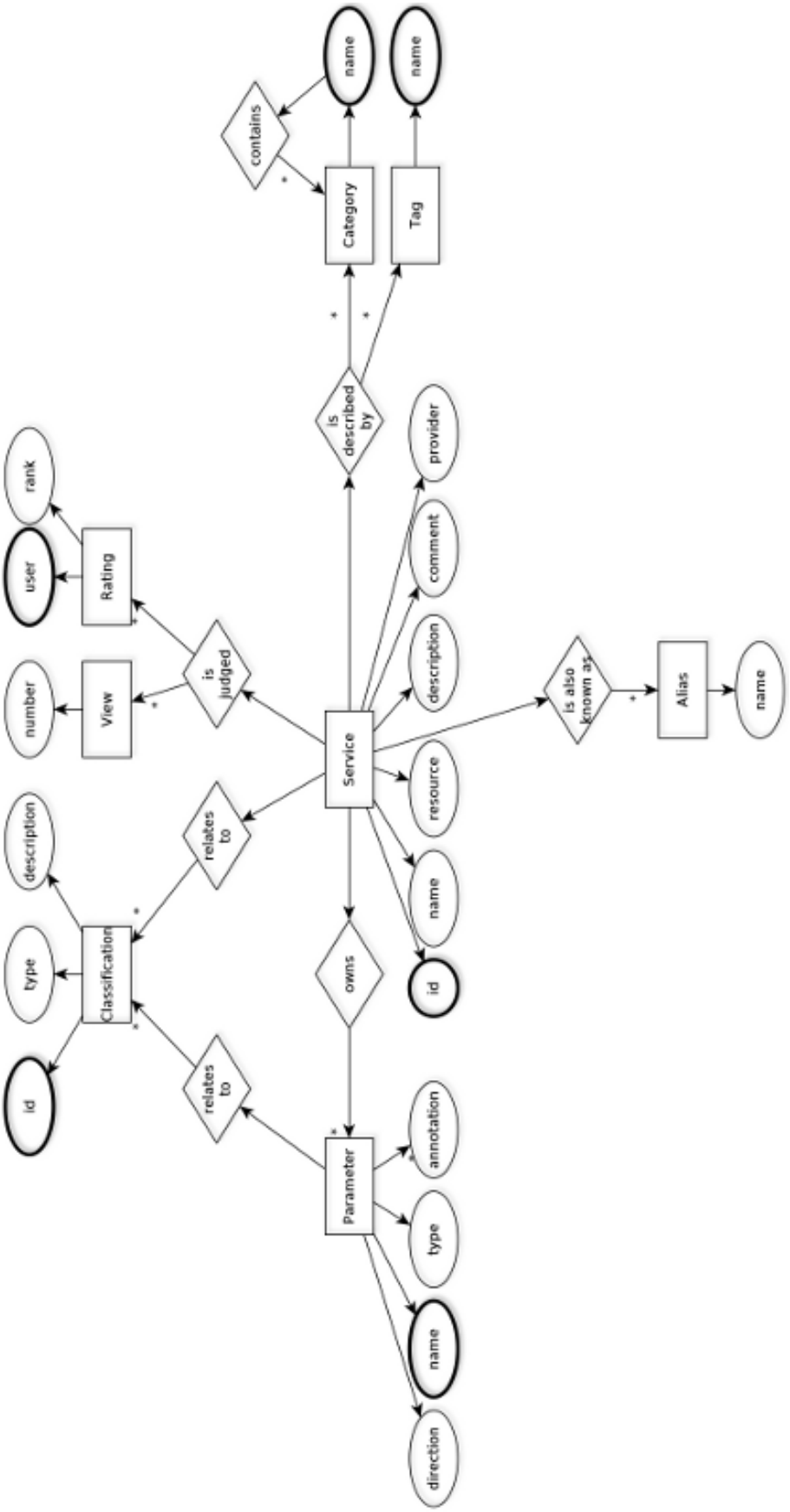


Abbildung 3.2: ER-Modell für die Informationen aus den Datenquellen

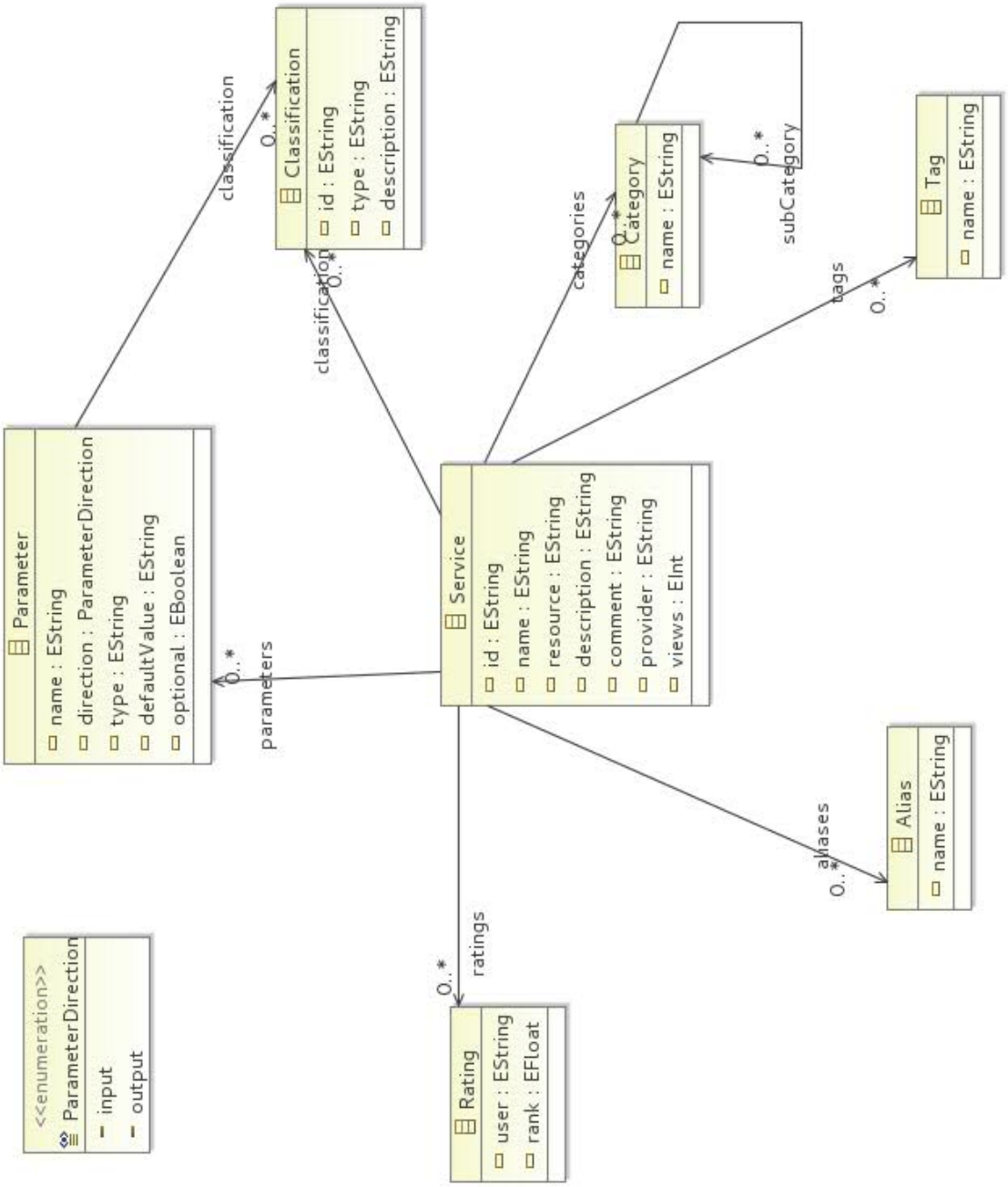


Abbildung 3.3: Entwurf des ersten PG-Metamodells

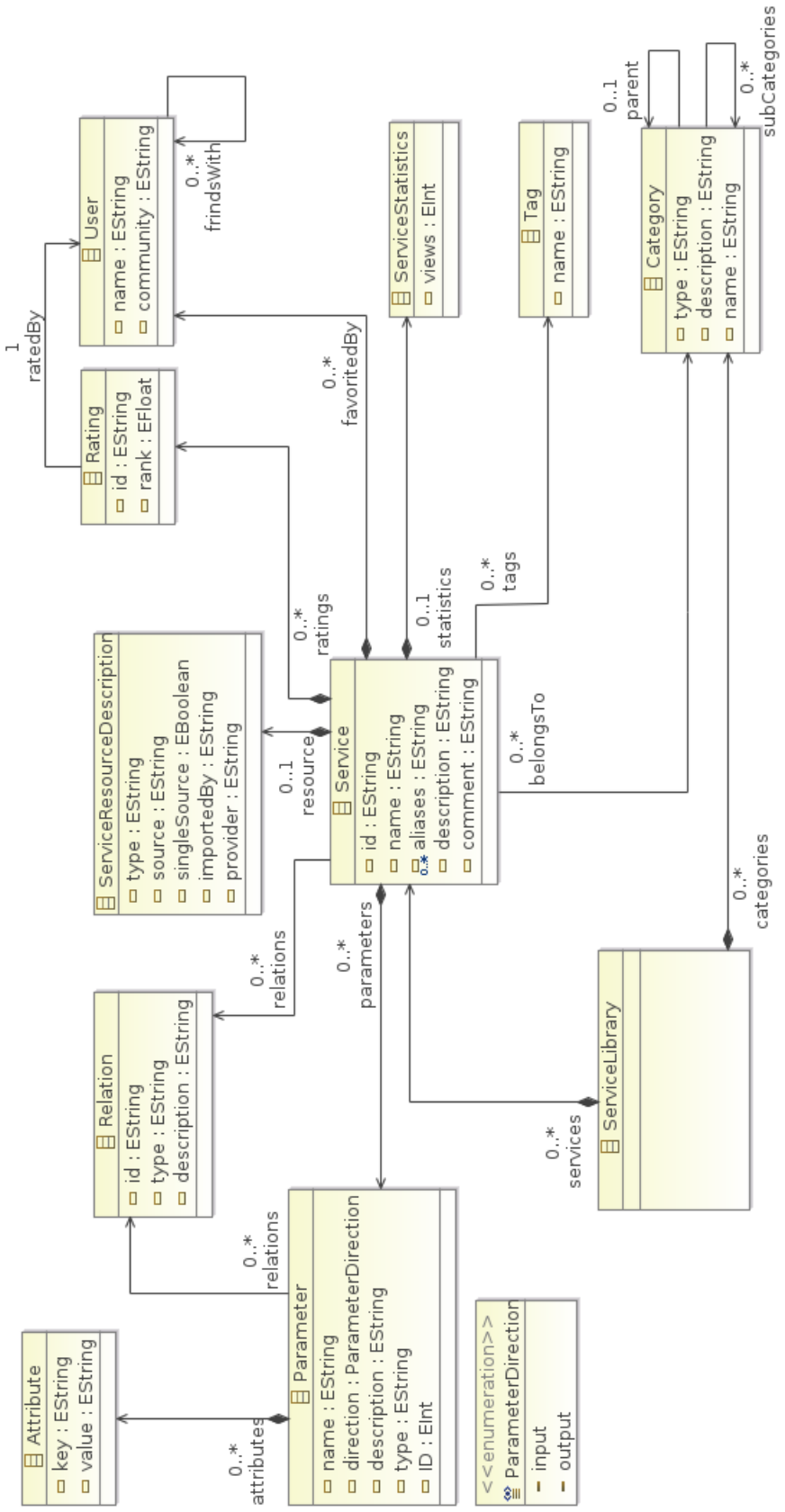


Abbildung 3.4: PG Metamodell, finale Version

Parameter Parameter sind nicht nur von entscheidender Bedeutung, um die Methoden des syntaktischen Filterns (vgl. Abschnitt 3.4.3) anzuwenden. Da sich Anzahl und Typ der Parameter unterscheiden und es in vielen Fällen ähnlich benannte Dienste gibt, dienen sie auch zur Identifizierung von gleichen Diensten, die unter verschiedenen Namen in den Quellen vorhanden sind.

Attribute Diese Eigenschaft dient der Verwaltung weiterer Merkmale eines Parameters. So könnte ein Eingabeparameter auf diese Weise etwa als optional markiert oder mit einem Standardwert versehen werden.

Relation Die Klasse `Relation` dient der Repräsentation von Zuordnungen einzelner Parameter und Services zu Kategorien der EDAM-Ontologie, wie sie bspw. in den ACD-Dateien der EMBOSS-Bibliothek angegeben sind.

Category Mithilfe von Ontologien können Services Kategorien zugeordnet werden, die wiederum miteinander in Beziehung stehen. Durch diese Zuordnung können Zusammengehörigkeiten von Services beschrieben werden.

Tag Ein Tag ist eine spezielle Art der Annotation, die in einigen der Portale sowohl an Services als auch an Workflows vergeben werden kann. Tags helfen bei der Gruppierung von Services und lassen eine einfache Zuordnung zu Gruppen zu.

ServiceStatistics Die Klasse `ServiceStatistics` umfasst im aktuellen Modell lediglich ein Attribut `views`, welches die Anzahl der Nutzer, welche die Service-Beschreibung auf einer Plattform betrachtet haben, speichern kann. Dieses Attribut wurde in eine eigene Klasse ausgelagert, da es in der Entwurfsphase für möglich erachtet wurde, dass andere Quellen weitere vergleichbare Informationen beitragen würden. Dies hat sich zwar nicht bewahrheitet, aber dennoch erscheint die separierende Modellierung vorteilhafter, da sie Erweiterungen vereinfacht.

User In einem späteren Stadium, schon in der Implementierungsphase, wurde das Modell um die Klassen `Rating` und `User` ergänzt. Diese dienen dazu, Informationen über (explizite) Nutzerwertungen eines Dienstes aufzunehmen. Die Nutzer werden dabei anhand ihres Namens innerhalb der Grenzen einer Quellplattform (*Community*) identifiziert.

Rating Unter Rating verstehen wir die Bewertung eines Services, falls die aus der entsprechenden Datenquelle ausgelesen werden konnte.

3.1.2 Empfehlungen

Aufgabe eines Empfehlungssystems ist das Erzeugen von Empfehlungen. In diesem Fall soll eine Menge von Services empfohlen werden. Den Erläuterungen zum Metamodell gemäß genügt es, anstelle einer vollständigen Service-Beschreibung lediglich die ID eines Services anzugeben.

Bei der Betrachtung der vom System erzeugten Empfehlungen muss zwischen zwei Typen unterschieden werden. Es existiert eine *interne* Repräsentation, die genaue Maßzahlen umfasst, und eine reduzierte, aus der internen Darstellung abgeleitete *externe* Repräsentation.

Interne Repräsentation

Interne Repräsentationen werden von den einzelnen aktiven Algorithmen[1] erzeugt und anschließend zu einer Repräsentation der Gesamtempfehlung zusammengesetzt. Zu diesem Behufe umfassen sie einige Informationen, welche für die an den Nutzer weitergeleitete Empfehlung unerheblich sind und ebendeshalb nicht Teil der externen Repräsentation sind.

Formale Entsprechung. Formal handelt es sich bei den internen Repräsentationen um partielle Abbildungen R aus der Menge der Service-IDs $\mathcal{S}$ in die Menge der nicht-negativen reellen Zahlen:

$$R : \mathcal{S} \rightharpoonup \mathbb{R}_0^+.$$

Dabei soll der Wert $R(s)$ für einen Service $s \in \mathcal{S}$ den Grad seiner vermuteten Relevanz darstellen (je höher der Wert, desto nützlicher der Service). Services, für welche die Abbildung undefiniert ist oder die auf den Wert 0 abgebildet werden, werden dabei als irrelevant interpretiert. Im Grunde ist es also unnötig, die Abbildung für einen derartigen Service zu definieren. Dennoch wurde die Möglichkeit der Definition (Abbildung auf den Nullwert) zugelassen, um den Entwicklern von Algorithmen andernfalls womöglich notwendige Fallunterscheidungen zu ersparen.
Ein wesentlicher Vorteil dieser Darstellung ergibt sich aus der Möglichkeit mehrere Empfehlungen über geeignet definierte Operatoren miteinander zu einer einzigen Empfehlung zu kombinieren. Komponentenweise lassen sich zum Beispiel eine *skalare Multiplikation* und eine *Addition* definieren.

Skalare Multiplikation. Für eine beliebige nicht-negative Zahl f und eine Empfehlung R ist das Produkt $R' := f \cdot R$ definiert als Abbildung $R' : \mathcal{S} \rightharpoonup \mathbb{R}_0^+$ mit

$$R'(s) := \begin{cases} f \cdot R(s) & \text{, falls } R(s) \neq \bot \\ \bot & \text{, sonst} \end{cases}$$

für alle Service-IDs $s \in \mathcal{S}$ (wobei $R(s) = \bot$ bedeute, dass die Abbildung R für das Argument s undefiniert ist).

Addition. Als zweistellige Verknüpfung, die dem Assoziativ- und Kommutativgesetz genügt, dient der Additionsoperator dazu, mehrere Empfehlungen in eine zu kombinieren. Auch er lässt sich komponentenweise definieren. Seien dazu R_1 und R_2

[1] Wie in Abschnitt 4.2.1 erläutert wird, kann der Nutzer festlegen, welche Algorithmen für eine Empfehlung berücksichtigt werden.

beliebige Empfehlungen, dann ergibt sich $R := R_1 + R_2$ als Abbildung $R : \mathcal{S} \rightharpoonup \mathbb{R}_0^+$ mit

$$R(s) := \begin{cases} R_1(s) + R_2(s) & \text{, falls } R_1(s) \neq \perp \text{ und } R_2(s) \neq \perp \\ R_1(s) & \text{, falls } R_1(s) \neq \perp \text{ und } R_2(s) = \perp \\ R_2(s) & \text{, falls } R_1(s) = \perp \text{ und } R_2(s) \neq \perp \\ \perp & \text{, sonst} \end{cases}$$

für alle Service-IDs $s \in \mathcal{S}$.

Die Wohldefiniertheit, insbesondere die Nicht-Negativität des einer Service-ID zugeordneten Wertes, ist in beiden Fällen offensichtlich.

Technische Umsetzung. Implementiert ist die Abbildung in der Klasse `Recommendation` (im Paket `algorithms`), welche die Verwaltung der Abbildung auf kanonische Weise an ein Objekt des Typs `TreeMap<String,Double>` aus der Java API delegiert. Dabei wird der Funktionsumfang auf die notwendigen Methoden eingeschränkt und insbesondere beim Hinzufügen eines Schlüssel-Wert-Paares eine Überprüfung des Wertes vorgenommen – der nicht negativ sein darf.
Zugleich stellt die Klasse statische Methoden bereit, welche die beiden oben beschriebenen Operatoren auf naheliegende Weise (durch Iteration über die Definitionsbereiche der beteiligten Operanden) implementieren, sowie eine Methode, die ausgehend von allen positiv bewerteten (relevanten) Service-IDs eine externe Repräsentation erzeugt.

Berechnung der Gesamtrepräsentation

Wie bereits erwähnt, erwartet das Empfehlungssystem von den einzelnen Algorithmen eine Empfehlung in Form einer internen Repräsentation. Die eben beschriebenen Verknüpfungen können von den Algorithmen zwar auch genutzt werden, wurden aber insbesondere zum Zwecke der Erzeugung einer Gesamtempfehlung implementiert.
Dem Empfehlungssystem steht zur Erzeugung einer Empfehlung eine Menge von Algorithmen $\mathcal{A}_1, \ldots, \mathcal{A}_n$ zur Verfügung, die gemäß des aktuellen Konfigurationsprofils mit nicht-negativen Gewichten $w(\mathcal{A}_i)$ versehen sind.
Zunächst wird eine „leere“ Empfehlung R erzeugt, d. h. eine Abbildung, die für ganz $\mathcal{S}$ undefiniert ist. Anschließend werden sukzessive durch Aufruf der Methode `recommend` die Einzelempfehlungen R_i für die Algorithmen $\mathcal{A}_i$ bestimmt und mit der (vorläufigen) Gesamtempfehlung entsprechend ihres Gewichts kombiniert:

$$R := R + (w(\mathcal{A}_i) \cdot R_i).$$

Es lässt sich leicht zeigen, dass aufgrund der Assoziativität und Kommutativität der Addition die konkrete Reihenfolge der Algorithmen keinen Einfluss auf das Ergebnis hat und sich daher zuletzt die Gesamtempfehlung schreiben lässt als

$$R = \sum_{i=1}^{n} w(\mathcal{A}_i) \cdot R_i.$$

Offenbar spielen Algorithmen $\mathcal{A}_i$ mit Gewicht $w(\mathcal{A}_i) = 0$ für die Berechnung der Gesamtempfehlung keine Rolle. Sie werden vom System als inaktiv betrachtet und daher gar nicht verwendet.

Beispiel. Die vorgehenden Beschreibungen sollen nun noch anhand eines Beispiels konkretisiert werden. Dazu seien zwei Algorithmen $\mathcal{A}_1$ und $\mathcal{A}_2$ vorausgesetzt, welche zu den folgenden Bewertungen einiger Dienste des EMBOSS-Pakets führen:

Tabelle 3.1: Empfehlung R_1

Service s $\in \mathcal{S}$	**Bewertung** $\mathbf{R_1}$**(s)**
`pepstats`	0,2
`seqall`	0,8
`seqcount`	0,7

Tabelle 3.2: Empfehlung R_1

Service s $\in \mathcal{S}$	**Bewertung** $\mathbf{R_2}$**(s)**
`pepstats`	0,4
`pepwindow`	0,4
`pepwindowall`	0,5

Bei einer Gewichtung w der Algorithmen mit $w(\mathcal{A}_1) = 1$ und $w(\mathcal{A}_2) = 0,25$ ergibt sich damit eine Gesamtempfehlung R:

Tabelle 3.3: Gesamtempfehlung $R := 1 \cdot R_1 + 0,25 \cdot R_2$

Service s $\in \mathcal{S}$	**Bewertung R(s)**
`pepstats`	0,300
`pepwindow`	0,100
`pepwindowall`	0,125
`seqall`	0,800
`seqcount`	0,700

Externe Repräsentation

Die Bewertungsdetails, d. h. die den einzelnen Services konkret zugeordneten Maßzahlen, sind für die Erstellung einer Gesamtempfehlung essentiell, für den Nutzer eines Workfloweditors jedoch unerheblich. Ihm soll in der grafischen Oberfläche eine (Teil-)Menge der vom Empfehlungssystem verwalteten Dienste in Abhängigkeit vom Kontext vorgeschlagen werden.
In der einfachsten[2] Variante erfolgt dieser Vorschlag in Form einer geordneten Liste der Services. Als Ergebnis eines Aufrufs der Methode `recommend` liefert die verantwortliche Klasse `RecommenderSystem` eine Liste von Service-IDs, die absteigend nach den ihnen durch die interne Repräsentation der Gesamtempfehlung zugeordneten Werten sortiert sind. Dabei werden ausschließlich positiv bewertete Service-IDs in die externe Repräsentation übernommen.
Aus den Service-IDs kann dann die in die grafische Oberfläche des Workfloweditors eingebettete Empfehlungskomponente eine für den Nutzer sinnvollere Beschreibung (etwa aus Namen, Beschreibung und Typ bestehend) unter Rückgriff auf das Datenmodell erzeugen.

[2] Alternative Darstellungsformen könnten sich auf Gruppierungen der empfohlenen Services stützen, Hierarchien abbilden. Auch höherdimensionale Repräsentationen sind vorstellbar. In der Ebene könnten etwa Graph-Cliquen dargestellt werden.

3.1.3 Unifizierung von Services

Services dienen als Grundlage für die Empfehlungen des Recommender-Systems. Diese Services werden aus den in Abschnitt 2.2 beschriebenen Datenquellen importiert. Aufgrund der Vielzahl von Datenquellen kann es zu Redundanzen in der Service-Datenbank kommen. Das Problem hierbei ist, dass Services, die mehrfach in der Service-Datenbank vorkommen, auch mehrfach in einer Empfehlung vorkommen können. Somit ist ein Szenario, in dem ein Anwender ein und denselben Service mehrfach in einer Empfehlung erhält, im Bereich des Möglichen. Eine solche Empfehlung hat eine geringe Qualität, da sich die Empfehlungen ausschließlich durch ihre Quelle unterschieden.
Das Problem, welches es zu lösen gilt, ist somit die Identifizierung solcher redundanten Services und die Zusammenfassung dieser redundanten Services zu Äquivalenzklassen. Die Zusammenfassung in Form von Äquivalenzklassen ist gewählt worden, da sie den Verlust von Informationen verhindert. Ein einfaches Verschmelzen der Services ist nicht möglich, da es Informationen gibt, die nicht verschmolzen werden können.
Im Laufe der Evaluation des Service-Modells ist festgestellt worden, dass es keine sichere Möglichkeit gibt zu bestimmen, ob zwei Services äquivalent sind. Aus diesem Grund wird eine Heuristik verwendet, welche mit hinreichenden und notwendigen Bedingungen arbeitet. Da die Heuristik primär auf notwendigen Bedingungen arbeitet ist es nicht möglich auszuschließen, dass zwei Services in einer Äquivalenzklasse unterschiedlich sind. Dies führt aber nicht zu einer Beeinträchtigung von Empfehlungen, da die Äquivalenzklassen nur dazu genutzt werden, die mehrfache Erwähnung eines Services zu verhindern.

Notwendige Bedingungen reichen nicht aus um festzustellen, ob zwei Services äquivalent sind. Services, die diese Bedingungen jedoch nicht erfüllen, können nicht äquivalent sein.

- Services haben identische Parameter: Nur Services, die die gleichen Parameter haben, können auch identisch sein. Unterscheiden sich die Zahl der Parameter zweier Services oder die Parameter in einem Attribut, ist davon auszugehen, dass es sich um unterschiedliche Services handelt. Bei den Attributen nimmt der Name jedoch nochmal eine gesonderte Rolle ein, da sich die Namen von Parametern unterscheiden können, ohne etwas darüber auszusagen, ob die Parameter identisch sind. Denn anders als die anderen Attribute der Parameter richten sich die Namen nach keinen Regeln.
- Services haben identische Relations: Analog zu den Parametern müssen auch die Relations identisch sein, da die Services sonst auf keinen Fall identisch sein können. Diese Aussage bezieht sich jedoch nur auf Services,

bei denen die Relations angegeben sind. Sollte einer von zwei verglichenen Services keine Relations angegeben haben, so muss diese Bedingung ignoriert werden. Dies kann unter anderem daran liegen, dass bei einem Service keine Relations angegeben worden sind. Wenn jedoch Relations angegeben sind, so müssen diese identisch sein.

Hinreichende Bedingungen sind die Bedingungen, bei deren Erfüllung sofort davon ausgegangen werden kann, dass die Services identisch sind.

- Services haben die gleiche Ressource im Eintrag *ServiceDescriptionResource*. Sollten diese gleich sein, müssen die Services identisch sein. Bei der Ressource handelt es sich beispielsweise um eine Datei, die beschreibt, wie auf den Service zugegriffen werden kann.

Bei genauerem Hinsehen fällt auf, dass nicht alle Attribute für den Vergleich zweier Services genutzt werden. Dies liegt daran, dass es Attribute gibt, die sich nicht für den Vergleich von Services einigen. Ein Beispiel hierfür sind die Attribute *Views* und *favoritedBy*. Diese Attribute geben Auskunft darüber, wie häufig ein Service betrachtet worden ist und wer ihn favorisiert hat. Da aber nicht davon auszugehen ist, dass gleiche Services bei den unterschiedlichen Quellen gleich häufig betrachtet worden sind, oder dass die gleichen Nutzer die Services auch bei den unterschiedlichen Datenquellen favorisiert haben, ist davon abzusehen diese Attribute als Bedingung für die Äquivalenz zweier Services zu nutzen.
Die Äquivalenzklassen der Services dienen nicht ausschließlich dafür die mehrfache Empfehlung eines Services zu verhindern. Sie werden zudem eingesetzt um die Services die bei dem Import von Workflows gefunden werden mit den Services in der Service-Datenbank zu verknüpfen. Hierzu werden die Services, die in den Workflows gefunden werden selber zu Äquivalenzklassen zusammengefasst. Dies passiert mit dem gleichen Ansatz wie bei den Services. Die Äquivalenzklassen für die Services aus den Workflows werden anschließen mit den Äquivalenzklassen der Services verknüpft. Eine genauere Verknüpfung oder gar eine 1:1 Verknüpfung zwischen den Services aus den Workflows und den Services selber ist nicht möglich. Dies liegt an den wenigen Informationen, die über die Services in den Workflows bekannt sind. Viel mehr als die Parameter, Rückgabewerte sowie Quelle und Autor sind nicht bekannt. Aus diesem Grund ist es nur diese ungenaue Verknüpfung möglich. Auch wenn sie nicht so genau ist, bietet sie dennoch die Möglichkeit weitere Informationen für Empfehlungen zu generieren.

Implementierung der Äquivalenzklassen-Bildung

Bei der Implementierung der Äquivalenzklassen-Bildung ist auf die Wiederverwendbarkeit für andere Bereiche, sowie die leichte Erweiterbarkeit geachtet worden. Die Wiederverwendbarkeit für andere Bereiche wird benötigt, da nicht nur Äquivalenzklassen von Services, sondern auch von Workflows gebildet werden. Um diese Flexibilität zu erreichen werden abstrakte Klassen genutzt. Die wichtigste Klasse für die Bildung der Äquivalenzklassen ist die abstrakte Klasse `EquivalenceClassesFinder<T>`, in welcher die Methoden für das Generieren der Äquivalenzklassen definiert

sind. Da die Heuristik sowohl für Services, als auch für Workflows auf die gleiche Art und Weise arbeitet, können viele Methoden schon in der abstrakten Klasse implementiert werden, welches die Redundanz im Quellcode reduziert. In den konkreten Implementierungen muss anschließend lediglich die Methode für das Update der Äquivalenzklassen implementiert werden. Diese Methode wird anschließend bei jeder Aktualisierung der Datenbank von den Datenbankcontrollern[3] dazu verwendet die Äquivalenzklassen zu aktualisieren. Es ist nicht möglich diese Methode schon in der abstrakten Klasse zu implementieren, da in dem Datenbankmodell kein einheitliche Speicherung der Äquivalenzklassen für Services und Workflows vorgesehen ist.
Um die Erweiterbarkeit zu gewährleisten, wurde das Entwurfsmuster *Chain-of-Responsibility*[FFB06] implementiert. Bei diesem Entwurfsmuster aus der Kategorie der Verhaltensmuster werden die einzelnen Bearbeitungsschritte in einzelne Klassen ausgelagert und wie in einer einfach verketteten Liste aneinander gehängt. Hierbei entsteht eine Liste von Bearbeitungsschritten. Bei der Suche nach den Äquivalenzklassen repräsentieren diese Bearbeitungsschritte die Überprüfung der einzelnen Bedingungen. Hierfür ist die abstrakte und generische Klasse `EquivalenceClassCondition<T>` zuständig. Die konkreten Implementierungen dieser Klasse für Services bzw. Workflows filtern alle Services oder Workflows raus, welche die Bedingung nicht erfüllen. Am Ende einer solchen Kette müssen alle Services bzw. Workflows die nicht herausgefiltert wurden zu einer Äquivalenzklasse gehören. Es muss insbesondere darauf geachtet werden, dass das Attribut `isSufficient` bei hinreichenden Bedingungen auf wahr gesetzt ist, da der Algorithmus, der für das Filtern zuständig ist, sonst nicht korrekt arbeiten kann.
Es ist leicht zu erkennen, dass sich die Ketten der Bedingungen leicht erweitern lassen. Auch die Reihenfolge in der die Bedingungen überprüft werden lässt sich einfach anpassen. Die Konfiguration der Ketten findet in der konkreten Implementierung der Klasse `EquivalenceClassesFinder<T>` statt.

[3]Bei den Datenbankcontrollern handelt es sich im die Klassen, die den Datenbankzugriff regeln.

3.2 Architektur

Das Empfehlungssystem ist ein Programmpaket, das sich (auf der obersten Ebene) aus mehreren Modulen zusammensetzt, die in der Anforderungsphase identifiziert wurden. Hierzu gehören insbesondere die diversen Importmodule (jeweils eines für ein bestimmtes Datenportal) sowie die unterschiedlichen Algorithmen, die zur Bewertung der Services beitragen können.
Da absehbar war, dass die Qualität der vom Empfehlungssystem erzeugten Empfehlungen erst verhältnismäßig spät würde beurteilt werden können, erschien die Festlegung auf einige wenige Recommender-Algorithmen und Datenquellen nicht empfehlenswert. Stattdessen sollten Algorithmen, aber auch Importmodule, auf einfache Weise hinzugefügt oder bereits verfügbare Module deaktiviert werden können. Um diesen Anforderungen auch unter der Annahme der praktischen Nutz- und Erweiterbarkeit gerecht zu werden, sind diese Module als Plug-ins konzipiert. Sie müssen daher nicht zur Kompilierzeit bekannt sein, sondern werden dynamisch zur Laufzeit geladen (für eine detaillierte Beschreibung des Plug-in-Konzeptes und des verwendeten Frameworks, vgl. Abschnitt **??**).
Die Hauptmodule des Programmes sind in Abbildung 3.5 dargestellt. In deren Zentrum steht die *Kernkomponente*, welche die Arbeit der anderen Module initiiert und koordiniert.

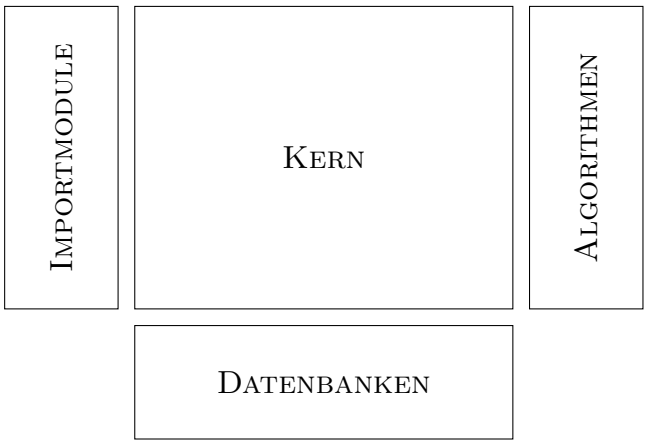


Abbildung 3.5: Architektur des Recommender-Systems: Hauptmodule

Im Folgenden werden der genauere Aufbau und die Aufgaben dieser Hauptmodule erläutert.

3.2.1 Kernkomponente

Die Kernkomponente ist für die Verwaltung aller anderen Module bzw. Komponenten verantwortlich. Ihre Funktionalität ist auf die drei Klassen `RecommenderSystem`, `RecommenderConfiguration` und `RecommenderDatabase` verteilt (im Paket `core`).

System. Dabei bildet die Klasse `RecommenderSystem` den Ausgangspunkt für die Verwendung des Programmpakets. Bei ihrer Initialisierung werden Objekte der anderen beiden Klassen angelegt, welche für die Verwaltung der Konfiguration sowie der Datenbank verantwortlich sind, und miteinander verknüpft, sodass sie für die Erledigung der spezifischen Teilaufgaben verwendet werden können.

Zu den bereitgestellten Funktionalitäten gehört insbesondere das Erzeugen einer Empfehlung (als einer geordneten Liste von Service-IDs), wie es in Abschnitt 3.1.2 beschrieben wurde, und das Durchführen eines Updates der Datenbank, das im Folgenden erläutert wird.

Konfiguration. Die Klasse `RecommenderConfiguration` verwendet die Java-API-Erweiterung „Configuration“ der Bibliothek „Apache Commons“ [Fou12], um eine vorhandene Konfigurationsdatei im XML-Format zu lesen bzw. zu schreiben. Diese Datei trägt den Namen *properties.xml* und ist im Unterverzeichnis *pgsassd/recommender* des Nutzerverzeichnisses gespeichert. Während der Ausführungszeit werden die jeweiligen Einstellungen jedoch intern in den Attributen der betroffenen Datenstrukturen verwaltet. Sollte zu Programmstart keine Konfigurationsdatei vorhanden sein oder diese nicht fehlerfrei gelesen werden können, wird eine Standardkonfiguration erzeugt, deren Struktur weiter unten erläutert wird.

Nicht zuletzt ist diese Klasse auch für die Verwaltung des aktuellen Profils zuständig und nimmt bei einem Wechsel desselben durch den Nutzer die notwendigen Änderungen vor. Dazu gehört insbesondere, dass die aktiven Algorithmen über den Wechsel informiert und mit den aktualisierten Konfigurationsdaten versorgt werden.

Struktur der Konfigurationsdateien. Das Format der XML-Konfigurationsdatei wird durch ein eigenes dafür definiertes Schema in der Datei *Configuration.xsd* im Projektverzeichnis (der Kernkomponente) beschrieben. Eine Konfiguration, welche diesem Schema entspricht, umfasst eine Menge $\mathcal{P}$ von Profilen, die von einem Nutzer verwendet werden. Jedes Profil p, das innerhalb von $\mathcal{P}$ über seinen Namen eindeutig identifiziert werden kann, umfasst Informationen über die bei seiner Nutzung zu verwendenden Importmodule und Algorithmen.

Importmodule werden nach den von ihnen implementierten Schnittstellen (für Services, Workflows oder Ontologie-Konzepte) getrennt verwaltet und mit einem einzigen Attribut `active` versehen. Wird diesem der Wert `true` zugewiesen, so wird

die Schnittstelle des zugehörigen Moduls im Laufe einer Update-Phase verwendet[4] – sofern es momentan tatsächlich verfügbar ist. Schließlich ist es den Modulen für jede ihrer Schnittstellen erlaubt, spezifische Einstellungen in einem Teilbaum, dessen Wurzelknoten den Tagnamen `specific` trägt, abzuspeichern. Diese Informationen werden jedoch vom Empfehlungssystem selbst ignoriert und lediglich an das betroffene Modul weitergeleitet.

Algorithmen können nicht allein als aktiv oder inaktiv markiert werden, der Benutzer hat darüber hinaus die Möglichkeit, ihnen ein beliebiges nicht-negatives Gewicht zuzuweisen. Algorithmen $\mathcal{A}$, denen ein Gewicht $w(\mathcal{A}) = 0$ zugewiesen wurde, sind gemäß Abschnitt 3.1.2 für die Berechnung der Gesamtempfehlung irrelevant und werden dementsprechend als inaktiv angesehen. Obschon das Attribut `active` in diesem Sinne redundant ist, wurde an ihm festgehalten, sodass der Nutzer eines Profils die Möglichkeit hat, einen Algorithmus (kurz- und langfristig) zu deaktivieren, ohne dass er sich die vormals eingestellte Gewichtung merken muss.

Auch die Algorithmen können (für jedes Profil) spezifische Einstellungen abspeichern. Nachfolgend ist beispielhaft ein Auszug der XML-Konfigurationsdatei dargestellt, der den auf „Mutual Information“ basierenden Algorithmus betrifft:

```
<algorithm active="false" id="[...]MutualInformation"
 name="Mutual Information" weight="1.8">
        <specific distanceFactor="1.0"
          maxPredecessorLevel="-1"
          maxSuccessorLevel="-1"/>
</algorithm>
```

Der Algorithmus mit dem Namen „Mutual Information“ ist in dem Profil also deaktiviert und mit einer Gewichtung von 1,8 versehen. Zudem sind mehrere modulspezifische Attribute `distanceFator`, `maxPredecessorLevel` und `maxSuccessorLevel` verzeichnet.

Standardkonfiguration. Das Empfehlungssystem ist für seine Arbeit auf das Vorhandensein eines Profils angewiesen. Sollte keines vorhanden sein, so wird intern ein Standardprofil erzeugt. Für das Nichtvorhandensein kann es zweierlei Gründe geben: Entweder es existiert noch keine Konfigurationsdatei oder diese konnte nicht fehlerfrei gelesen werden. In ersterem Falle wird eine Konfigurationsdatei angelegt, welche genau dieses Standardprofil umfasst; in letzterem Falle wird die bereits bestehende Datei *nicht* überschrieben, um den Verlust von Nutzereinstellungen zu vermeiden.

Ein Standardprofil wird um Einträge für alle vom System registrierten Importmodule und Algorithmen ergänzt, welche jedoch allesamt als inaktiv markiert werden (Algorithmen erhalten zudem eine Standardgewichtung von 1,0). Genau genommen werden die Attribute `active` und `weight` in einem solchen Fall gar nicht gesetzt (sie sind im Schema als optional deklariert) und die Klasse `RecommenderConfigurati-`

[4] Angemerkt sei, dass dies eine feingranulare Konfiguration ermöglicht, sodass bei Modulen, welche mehrere Schnittstellen implementieren, nicht allein das ganze Modul aktiviert oder deaktiviert werden kann, sondern gewisse Teilaspekte davon.

on weist den internen Attributen bei fehlender Spezifikation in der XML-Datei jene Standardwerte zu.[5]
Automatische Änderungen können aber auch an korrekt eingelesenen, von Nutzern erstellten Profilen vorgenommen werden. Dies ist immer dann der Fall, wenn im aktuellen Lauf des Systems Importmodule oder Algorithmen registriert sind, welche keinen Eintrag in einem Profil der Konfigurationsdatei besitzen. Für jedes solche Profil wird ein Standardeintrag erzeugt. Einträge von Modulen, die aktuell nicht geladen sind, werden nicht gelöscht und auch ihre Attribute `active` nicht geändert – dieser Aspekt wird über ein internes Verfügbarkeitsattribut nachgehalten.
Insgesamt verhält sich die Klasse `RecommenderConfiguration` bei der Erzeugung und Modifikation der Konfigurationsdatei also sehr defensiv. Eine offensivere oder interaktivere Verhaltensweise kann durch das Programm umgesetzt werden, in welches das Empfehlungssystem eingebettet ist.

Datenbank. Das Objekt der Klasse `RecommenderDatabase` initialisiert bei seiner Erzeugung zunächst die Hibernate-Bibliothek, die, wie in Abschnitt 3.2.2 näher ausgeführt, für das Laden der Daten aus der (persistenten) Datenbank in die (temporären) Strukturen des Datenmodells verantwortlich ist. Davon abgesehen erschöpft sich der Aufgabenbereich des Objektes in der Verwaltung dreier Referenzen vom Typ `DatabaseManager`.
Diese abstrakte Klasse besitzt konkrete Spezialisierungen für alle drei Bibliothekstypen (`ServiceDatabaseManager`, `WorkflowDatabaseManager`, `OntologyDatabaseManager`), welche zum einen die Sitzungen für eventuell weitere notwendige Datenbankzugriffe verwalten und zum anderen Methoden zur Aktualisierung des Datenbestandes der jeweiligen Bibliothek bereitstellen. Die Verschiedenartigkeit der einzubringenden Typen (im Falle der Service-Bibliothek sind dies etwa Service- und Kategoriebeschreibungen, im Falle der Workflow-Bibliothek hingegen Workflowbeschreibungen) war der Anlass für die Aufteilung in die drei Unterklassen.[6]

Updates. Für die Durchführung des eigentlichen Aktualisierungsprozesses greifen die konkreten Spezialisierungen von `DatabaseManager` wiederum auf sogenannte `DatabaseUpdateManager` zurück. Obwohl diese weitere Aufteilung angesichts des geringen Funktionsumfanges der nutzenden Klasse `DatabaseManager` zunächst unverhältnismäßig erscheinen mag, liegt ihr letztlich doch eine konkrete Motivation zugrunde: Sowohl die Typen mit Oberklasse `DatabaseManager` als auch jene mit Oberklasse `DatabaseUpdateManager` erweitern nämlich die von der Java-API angebotene `Thread`-Klasse und können daher nebenläufig eingesetzt werden. Die parallele Ausführung geschieht dabei in zwei Ebenen, wobei jene Typen die erste bilden und letztere Typen die zweite.
Die Nebenläufigkeit ist aus praktischer Sicht ein wesentlicher Faktor zur Beschleunigung des Update-Prozesses. Genau genommen ist von der parallelen Ausführung

[5] Eine Benutzerschnittstelle sollte deshalb vor erstmaliger Nutzung des Empfehlungssystems auf die Notwendigkeit der Konfiguration hinweisen. Andernfalls wird stets eine leere Empfehlung erzeugt.

[6] Die bibliotheksunabhängige Sitzungsverwaltung hingegen ist in der abstrakten Oberklasse `DatabaseManager` implementiert.

der Threads nicht die Aktualisierung der Datenbank (das eigentliche *Update*) betroffen, sondern die ihm vorhergehende Phase des Daten*imports*. Für die Durchführung des Updates wird auf die Beendigung aller Import-Threads gewartet, da für die Gesamtheit der durch sie gesammelten Daten die Datenbank-Controller Unifizierungsprozesse ausführen sollen (vgl. Abschnitt 3.1.3).[7]
An einem Beispiel-Update soll die Vorgehensweise der `DatabaseManager`, `DatabaseUpdateManager` und der betroffenen Importmodule demonstriert werden. Angenommen wird dabei, dass die folgenden Importmodule im aktuellen Profil als aktiv markiert sind:

Name	**Typ**
EMBOSS	`ServiceImportModule`
BioCatalogue	`ServiceImportModule`
MyExperiment	`WorkflowImportModule`

Wird nun die Methode `update()` der Klasse `RecommenderSystem` aufgerufen, so ermittelt sie mithilfe von `RecommenderConfiguration` die aktuell aktivierten Importmodule und übergibt diese an die Update-Methode ihrer Instanz von `RecommenderDatabase`. Diese unterteilt die Importmodule gemäß ihrer Typen in drei Gruppen und gibt diese an die entsprechenden Objekte vom Typ `DatabaseManager` weiter, die zuletzt für jedes ihnen übergebene Modul den Thread eines `DatabaseUpdateManagers` starten.
Schematisch ist dieser Ablauf (mit den konkreten Typen) in Abbildung 3.6 dargestellt:

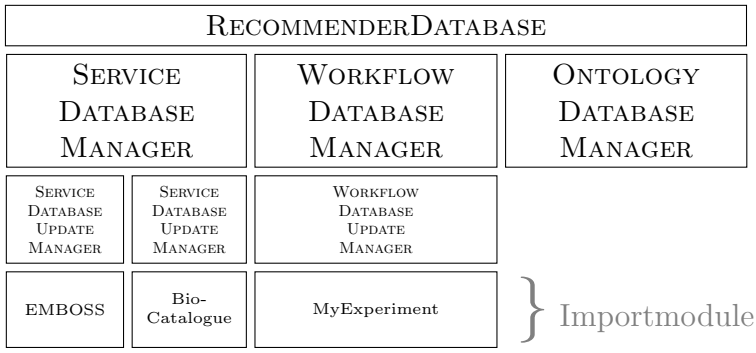


Abbildung 3.6: Beispiel-Update: Nebenläufigkeit in der Importphase.

Nebenläufigkeit ist in der Importphase also insgesamt auf drei Ebenen möglich:

1. Für alle Bibliotheken kann der Import parallel durchgeführt werden. Dazu wird jeweils eine Instanz des Typs `ServiceDatabaseManager`, `WorkflowDatabaseManager` und `OntologyDatabaseManager` ausgeführt.

[7]Beim gegenwärtigen Stand der Implementierung ist der Aspekt tatsächlich gar nicht mehr ausschlaggebend, da in den Merge-Prozess auch die in der Bibliothek verzeichneten Datensätze einbezogen werden. Allerdings bedarf dieser Ansatz noch einer Modifikation, da andernfalls Datensätze, die aus den Quellen gelöscht wurden, niemals aus der lokalen Datenbank entfernt werden. Die oben angestellte Überlegung könnte sich insofern in nächster Zeit wieder als relevant erweisen.

2. In Abhängigkeit von der Anzahl der zu aktualisierenden Module des jeweiligen Typs werden zeitgleich Instanzen der Spezialisierungen von `DatabaseUpdateManager` erzeugt und mit dem Import für ein einziges Modul beauftragt.
   In obigem Beispiel werden zwei Objekte vom Typ `ServiceDatabaseUpdateManager` erzeugt sowie eines vom Typ `WorkflowDatabaseUpdateManager`.

3. Letztlich können die Importmodule selbst auch mehrere Threads verwenden.[8]

Das serialisierte Hinzufügen der Daten startet erst nach Beendigung *aller* Import-Threads. Zunächst wird die Service-, dann die Workflow- und zuletzt die Ontologie-Bibliothek aktualisiert. Zum einen scheint eine Parallelisierung an dieser Stelle keinen Effizienzgewinn mit sich zu bringen, sodass schon unter Berücksichtigung der prinzipiell höheren Fehleranfälligkeit von parallelen Programmen auf eine derartige Umsetzung verzichtet wurde; zum anderen gibt es einseitige Abhängigkeiten unter den Bibliotheken, welche zumindest die Reihenfolge der Aktualisierung von Service- und Workflowbibliothek festlegt.[9]

3.2.2 Datenbanken

In der Entwurfsphase des Projekts erkannte die Projektgruppe, dass der Umfang der zu verarbeitenden Daten relativ groß ist. Schätzungen ergaben zu diesem Zeitpunkt, dass einige Tausend Services und Workflows gespeichert werden müssen. Es wurde auch in Betracht gezogen, dass das Empfehlungssystem auf einem Server ausgeführt werden kann und eine Workflowmanagementumgebung die Empfehlungen als Anfragen an diesen Server sendet. Aus diesem Grund entschied sich die Projektgruppe für die Verwendung einer Datenbank.
Bezüglich der verwendeten Datenbank wurde auf Flexibilität geachtet. So sollte die verwendete Datenbank nicht auf einen bestimmten Hersteller (z.B. Oracle) oder eine bestimmte Implementierung (wie z.B. PostgreSQL) festgelegt werden. Die Programmiersprache Java und deren Klassenbibliotheken bieten dazu zunächst eine gute Basis.

JDBC

Durch die *JDBC*-Schnittstelle (*Java Database Connectivity*) kann prinzipiell jede Datenbank angesprochen werden, für die es einen JDBC-Treiber gibt. Für die bekannten Datenbanken wie MySQL, PostgreSQL und Oracle ist so ein JDBC-Treiber verfügbar.
Wenn nur die JDBC-Schnittstelle ohne zusätzliche Abstraktionsschichten zwischen der Applikation und der Datenbank steht, müssen Anfragen an die Datenbank in

[8] Dies ist etwa für das Importmodul „BioCatalogue“ der Fall, vgl. Seite 61.

[9] Service-IDs in der Workflow-Bibliothek können mit „neuen“ Services der Service-Bibliothek natürlich erst dann assoziiert werden, wenn diese selbst in den Datenbestand aufgenommen wurden. Auf die Details dieses Prozesses geht Abschnitt 3.1.3 ein.

SQL (*Structured Query Language*) formuliert werden. Zwar wird SQL in einem ISO-Standard[Iso] beschrieben und von den Herstellern der Datenbanken implementiert, jedoch gibt es einige Hersteller-spezifische Besonderheiten zu beachten, was mit einem kleinen Beispiel einer Namenssuche ohne Unterscheidung zwischen Groß- und Kleinschreibung verdeutlicht werden soll. In einer erdachten Telefonbuch-Tabelle (siehe Tabelle 3.4) in einer Datenbank sollen mit der SQL-Abfrage aus Listing 3.1 zu einem gegebenen Suchwort die passenden Telefonnummern gesucht werden.

Tabelle 3.4: Die Tabelle address_book_

name_	**phone_**
Alice	555-12345
Bob	555-54321

Listing 3.1: SQL-Anfrage zur Adressbuchsuche

```
SELECT * FROM address_book_ WHERE name_ LIKE 'alice';
```

Diese Anfrage ist problematisch. Bei Microsoft-Datenbanken ist es möglich, das Verhalten bezüglich der Groß- und Kleinschreibung bei der Suche pro Tabelle einzustellen und somit könnte die Anfrage abhängig von der Konfiguration funktionieren. Bei der PostgreSQL hingegen muss statt `LIKE` der Operator `ILIKE` verwendet werden. Es gibt noch viele weitere Inkompatibilitäten zwischen den Datenbanken, wie zum Beispiel die reservierten Wörter, die je nach Datenbank unterschiedlich sind. So kann ein Bezeichner in der einen Datenbank gültig sein während er in einer anderen Datenbank zum Problem wird. SQL ist also aus rein syntaktischer Sicht eine Art kleinster gemeinsamer Nenner. Um die SQL-Anfrage aus dem Beispiel in Java zu verwenden kann der Code aus Listing 3.2 verwendet werden.

Listing 3.2: Beispiel: Adressbuchabfrage mit JDBC

```
String searchString = "alice";

Connection connection = null;
ResultSet res = null;
PreparedStatement stmt = null;

try {
        connection = getConnection();
  try {
        String sql = "SELECT name_, phone_ FROM "
        + "address_book_ where lower(name_) LIKE ?;";
        stmt = connection.prepareStatement(sql);
        stmt.setString(1, "lower(" + searchString + ")");
        res = stmt.executeQuery();
    while (res.next()) {
        System.out.println("Name : " + res.getString(1));
        System.out.println("Phone: " + res.getString(2));
    }
  } catch (SQLException e1) {
         handleException(e1);
  } finally {
        closeResultSet(res);
        closeStatement(stmt);
  }
} catch (SQLException e) {
         handleException(e);
} finally {
        closeConnection(connection);
}
```

Hibernate

Wenn man nun an Stelle von JDBC das ORM-Framework (object-relational mapping, [Orm]) Hibernate[Hib] verwendet, lässt sich das Beispiel erheblich vereinfachen. Die Tabelle `address_book_` wird dabei durch die Klasse `Entry` repräsentiert. Umgekehrt wird die Klasse `Entry` auf die Tabelle `address_book_` abgebildet, man spricht von einer objektrelationalen Abbildung.
Diese Abbildung kann in Form einer XML-Datei bereitgestellt werden oder auch durch sogenannte Annotationen an den Namen und Attributen der Klassen (im Beispiel zu sehen). Jede Zeile in der Tabelle `address_book_` wird durch eine Instanz der Klasse `Entry` repräsentiert. Diese Instanz hält die Werte der Spalten in ihren Attributen.

Listing 3.3: Beispiel: Adressbuchabfrage mit Hibernate

```
@Entity
@Table(name="address_book_")
public class Entry implements Serializable {
        String name_;
        String phone_;
}

// . . .

Criteria c = session.createCriteria(Entry.class);
c.add(Restrictions.ilike("name_", searchString);
List results = c.list();

Iterator i = results.iterator();
while (i.hasNext()) {
        Entry entry = (Entry) i.next();
        System.out.println("Name : " + entry.getName());
        System.out.println("Phone: " + entry.getPhone());
}
```

Hibernate verwendet JDBC, um Anfragen an die Datenbank zu stellen, ist also eine Abstraktionsschicht zwischen Programmlogik und Datenbank. Einer der großen Vorteile von JDBC, die Unabhängigkeit von der verwendeten Datenbank, bleibt erhalten. Die Datenbank wird einfach in einer Properties-Datei konfiguriert.

Hibernate ist in der Lage, Datenbank-Schemata selbst anzulegen und ggfs. zu aktualisieren. Darüber hinaus beseitigt Hibernate viele der zuvor beschriebenen Nachteile. So können Anfragen an die Datenbank in *HQL* (*Hibernate Query Language*) oder auch programmiertechnisch (siehe Beispiel) gestellt werden. Dabei bedient sich Hibernate sogenannter „Dialekte“ um Hibernate-Anfragen in Hersteller-spezifische SQL-Anfragen zu übersetzen. Dieses Beispiel kann nun problemlos auf jeder JDBC-kompatiblen Datenbank ausgeführt werden. Wenn kompliziertere Beispiele konstruiert werden können allerdings auch bei Verwendung von Hibernate Komplikationen auftreten.

Basierend auf diesem Hintergrund entschied die Projektgruppe, dass Hibernate als Schnittstelle zur Datenbank verwendet werden soll. Die Datenbank kann nun je nach Einsatzzweck gewählt werden. Während der Entwicklung kommt zunächst die HyperSQL-Datenbank zum Einsatz. Diese hält alle importierten Daten lediglich im Arbeitsspeicher vor und speichert beim Schließen alle Anfragen die zum aktuellen Zustand geführt haben in einer Text-Datei ab.

EMF

Die Projektgruppe entschied außerdem, dass die zu programmierende Software und insbesondere das Datenmodell mittels UML-Diagrammen oder in ähnlicher Form modelliert werden soll. Zu diesem Zweck wurde auch *EMF*[Emf] näher untersucht.

EMF ist ein Werkzeug zur Metamodellierung und zur Erzeugung von Quelltext aus dem Metamodell. Dabei beschreibt ein Metamodell die Struktur eines Modells. Das Datenmodell besteht aus Java-Klassen, welche den Aufbau von konkreten Services oder Workflows beschreiben und Referenzen zwischen diesen Klassen. Das Metamodell beschreibt hier die Struktur von Klassen und Referenzen. Dies ist im Falle von EMF im Ecore-Metamodell realisiert.
Ähnlich wie es in UML-Programmen üblich ist, bietet auch EMF einen graphischen Editor zur Modellierung. Dieser basiert auf *Eclipse* und ist in den *Eclipse Modeling Tools*, einem auf Modellierung spezialisiertem Paket von Eclipse enthalten. Im Anschluss an die Modellierung kann das Modell in Form von Java-Quelltexten automatisch generiert werden. Dieses Datenmodell kann dann zur Laufzeit des Programms Instanzen des Modells speichern. Es ist außerdem möglich, ein XML-Schema des Modells zu erstellen. Instanzen des Modells können dann in XML-Dokumenten serialisiert werden, welche anhand des Schemas validiert werden.
Die Möglichkeit, Instanzen eines Modells in XML-Dateien zu speichern, ist einerseits überaus praktisch, andererseits wurde zu Beginn die Speicherung der Daten in einer relationalen Datenbank unter Verwendung des Hibernate-Frameworks vereinbart.

Teneo

Wie bereits zuvor angesprochen wurde, wird eine objektrelationale Abbildung benötigt, wenn Java-Objekte mit Hilfe des Hibernate-Frameworks in einer relationalen Datenbank gespeichert werden sollen. Bei der manuellen Erzeugung dieser Abbildung ist es hilfreich, dass die Strukturen von Datenbanken und Java-Modellen ähnlich sind. Dabei besteht die größte Ähnlichkeit zwischen Klassen und Tabellen. Während in Tabellen die Daten in Spalten abgelegt werden, halten Klassen die Daten in Attributen. In beiden Fällen dürfen die Daten aus unterschiedlichen Datentypen bestehen.
Primitive (Java-)Datentypen können direkt als Spalten in Tabellen abgelegt werden. Allerdings gibt es in relationalen Datenbanken das Konzept der zusammengesetzten Datentypen nicht. Daher wird generell eine Tabelle pro Klasse benötigt. Assoziationen zwischen Klassen können dann über Fremdschlüssel, also einen Zeiger auf eine bestimmte Zeile einer anderen Tabelle, abgebildet werden.
Da die Grundidee hinter dieser Abbildung leicht vermittelbar ist und es darüber hinaus eine einfache Bildungsvorschrift dafür gibt, kann die Abbildung auch leicht durch ein Programm erzeugt werden. Genau diese Aufgabe erfüllt *Teneo*, ein Eclipse-Projekt welches zum Eclipse Modeling Framework gehört. Teneo ist in der Lage, eine objektrelationale Abbildung für die Nutzung mit Hibernate oder *EclipseLink*[Eclb] zu erzeugen. Auf diese Weise können Elemente des Recommender-Datenmodells einfach in die Datenbank gespeichert und auch wieder ausgelesen werden.

3.2.3 Plug-ins

Um die Abhängigkeit des Gesamtsystems von den einzelnen Datenquellen, aber auch von den Bewertungsverfahren möglichst gering zu halten und eine einfache Erweiterbarkeit der Funktionalität zu garantieren, wurde schon früh in der Entwicklungsphase die Entscheidung getroffen, alle infrage kommenden Module als Plug-ins umzusetzen und in das System einzubinden.[10]
Zunächst wurden die bereits erwähnten Importmodule und Algorithmen als solche Komponenten identifiziert. Im Laufe der Umsetzung verschiedener Importmodule, welche mitunter mehrere Dateiformate zugleich handhaben mussten, wurde jedoch deutlich, dass auch die Parser für einzelne Formate als Plug-ins implementiert werden sollten. Dieser Beschluss erhöhte die Flexibilität des Ansatzes noch einmal und verringert in einigen Fällen womöglich den Implementierungsaufwand für Autoren neuer Importmodule, indem diese nun auf bereits zur Verfügung stehende Parser über eine einheitliche Schnittstelle zurückgreifen können – zumal wenn diese bereits mit dem System geliefert wurden.

3.2.4 Importmodule

Importmodule sind Plug-ins, welche Daten aus Datenquellen auslesen und diese in ein Datenbank-konformes Format überführen. Diese werden in Importmodule für Services, Workflows und Ontologien unterteilt. Nähere Beschreibungen der einzelnen Module können dem Abschnitt 3.3 entnommen werden.

3.2.5 Parser

Das Paket `parser` umfasst drei Typen von Parser-Schnittstellen. Dazu gehören Parsertypen für die Dateiformate von Service- und Workflowbeschreibungen, aber auch einer für die Auswertung von (hierarchischen) Kategoriestrukturen.
Alle Schnittstellen umfassen eine einzige Methode `parse()`, welche den Speicherort der zu parsenden Dateien und (in den ersten beiden Fällen) eine Referenz auf das „anzureichernde“ Objekt übergeben bekommt. Hingegen verzichtet die Schnittstelle für Kategorie-Parser auf eine derartige Referenz und hat als Rückgabewert stattdessen eine Liste von Objekten des Typs `Category`.
Dieser Unterschied ist darin begründet, dass die Daten für Kategorien aus einzelnen Quellen stammen, wohingegen jene für Services und Workflows mitunter in mehreren

[10] Die Gruppe hat sich auch vor dem Hintergrund der Realitätsnähe und ihrer Lernbegier zu diesem Schritt entschlossen. Während die grundlegende Funktionsweise des Plug-in-Mechanismus bekannt war, lagen konkrete praktische Erfahrungen damit bei den meisten Teilnehmern nicht vor.

Quellen (unterschiedlicher Formate) ihren Ursprung haben,[11] sodass dort tatsächlich nicht eine reine Erzeugung, sondern vielmehr eine „Anreicherung“ der Daten stattfindet.
Die Autoren von Parsern sind deshalb – besonders in Hinblick auf Meta-Daten – dazu aufgefordert, eventuell vorhandene Daten *nicht* zu überschreiben.[12]
Nachfolgend sind alle fünf der im Rahmen dieser Projektgruppe implementierten Parser aufgelistet.

Service-Parser für ACD-Dateien (EMBOSS). Services des umfangreichen EMBOSS-Programmpaketes (vgl. Abschnitt 2.2.1) werden durch Dateien im *ACD*-Format[13] (*Ajax Command Definition*) beschrieben.
Im strengen Sinne handelt es sich bei dem projekteigenen Parser für diese Dateien eher um einen Übersetzer (*Compiler*). Der eigentliche Vorgang des Parsens wird an eine externe Implementierung aus dem „Seahawk“-Projekt[14] delegiert. Dessen Ausgabe, die sich auf andere Datenstrukturen stützt, wird dann in Datenstrukturen des Meta-Modells übersetzt.

Workflow-Parser für RapidMiner. Workflows für die RapidMiner-Umgebung werden im .xml-Format gespeichert. Sie bestehen aus drei verschiedenen Arten von Komponenten mit den Namen Operator, Process und Connect. Prinzipiell sind bei RapidMiner verschachtelte Workflows möglich, allerdings wird diese Möglichkeit in der Praxis nur selten genutzt. So sind von den rund 200 RapidMiner-Workflows im MyExperiment-Portal nur etwa 5% verschachtelte Workflows. Aus diesem Grund wurde entschieden, verschachtelte Workflows für unser Projekt nicht weiter zu berücksichtigen und lediglich einfache Workflows zu parsen und zu importieren.

Ein simpler RapidMiner-Workflow besteht aus einer Operator-Komponente als Hülle, die eine einzelne Process-Komponente enthält, welche den Workflow darstellt. In dem Process befinden sich Operator-Komponenten, welche die Services darstellen, und Connect-Komponenten, die den Verbindungen zwischen den einzelnen Services entsprechen. Da Services verschiedene Ports als Ein- und Ausgang haben können, werden diese in den Connect-Komponenten ebenfalls mit angegeben. Unser Workflow-Parser für RapidMiner-Dateien besitzt für jede Art von Workflow-Komponente eine eigene Parse-Methode, so dass der komplette Workflow erfolgreich eingelesen werden kann.

[11] Dies trifft etwa auf das Importmodul für „MyExperiment“ zu, bei dem Meta-Informationen (etwa Autorennamen, Beschreibung etc.) von der projekteigenen Webseite im HTML-Format geladen werden, während für die eigentlichen Workflow-Beschreibungen (Services und Verknüpfungen) – in Abhängigkeit von dem genutzten Format – verschiedene weitere Parser verwendet werden.

[12] Auch hier gibt das Importmodul für „MyExperiment“ Aufschluss: Während im XScufl-Format Workflow-Beschreibungen auch mit den oben genannten Meta-Daten versehen sein können, ist dies im T2Flow-Format nicht möglich.

[13] Für eine Syntaxbeschreibung des ACD-Dateiformates siehe `http://emboss.sourceforge.net/developers/acd/syntax.html`.

[14] Für weitere Informationen zu „Seahawk“ siehe `http://moby.ucalgary.ca/seahawk/`.

Workflow-Parser für T2Flow. *T2Flow* ist das XML-basierte Wokflowformat, das von der Toolsuite *Taverna 2.x*[15] benutzt wird. Ein in diesem Format abgespeicherter Workflow besitzt mindestens einen Datenfluss (Abbildung 3.7 `dataflow [1..*]`). Ein Datenfluss besteht aus beliebig vielen Services (Abbildung 3.7 `processor [0..*]`), die untereinander verbunden sein können (Abbildung 3.7 `datalink [0..*]`). Eine Verbindung ist dabei durch genau eine Quelle (Abbildung 3.7 `source`) und genau eine Senke (Abbildung 3.7 `sink`) ausgezeichnet. Ein Service verfügt über einen Namen (Abbildung 3.7 `name`), mögliche Ein- und Ausgänge (Abbildung 3.7 `inputPorts`, `outputPorts`) und kann auf verschiedene Weisen Aufgaben (Abbildung 3.7 `activities`) erledigen.

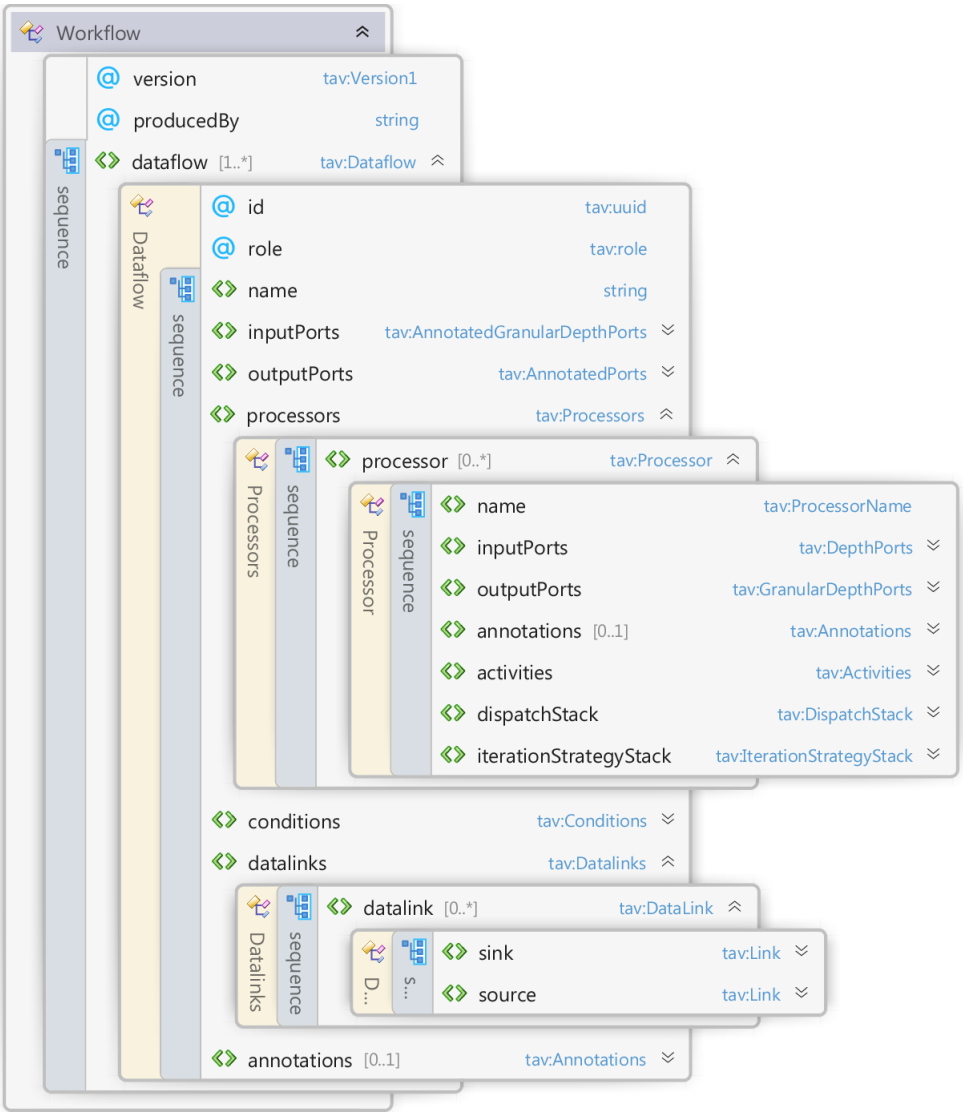


Abbildung 3.7: Visualisierung des T2Flow Formats

Da *T2Flow* ein intern in *Taverna 2.x* eingesetztes Format ist, lässt sich nicht genau aufzählen, wie die Aufgaben eines Services definiert werden können. Eine mögli-

[15] http://www.taverna.org.uk/documentation/taverna-2-x

che Definition ist ein Verweis auf eine *Web Services Description Language* Ressource. Dabei haldelt es sich um eine standardisierte[16] Beschreibungssprache für Services. Der Parser für *T2Flow* erkennt die Aufgaben eines Services durch einen WSDL-Verweis, ein internes Skript oder eine Konstante. Falls eine solch definierte Aufgabe erkannt wird, werden Felder `ServiceMetaInfo.type` und `ServiceMetaInfo.operation` gemäß der Tabelle 3.5 gesetzt.

Erkannte Aufgabe	`ServiceMetaInfo.type`	`ServiceMetaInfo.operation`
WSDL-Verweis	„wsdl“	Verweis auf die WSDL-Ressource
Internes Skript	„script“	Inhalt des Skriptes
Konstante	„value“	Konstante als Zeichenkette

Tabelle 3.5: `ServiceMetaInfo`-Format

Falls zusätzlich zu einem WSDL-Verweis eine genaue Anfrage angegeben wird, so wird diese unter `ServiceMetaInfo.extraInfo` als „operation: Inhalt der Anfrage“ gespeichert.

Workflow-Parser für XScufl. Dieser Parser für das XML-basierte Format *XScufl* setzt nicht auf anderen Parsern auf, sondern extrahiert die Daten eigenständig aus dem zum Dokument gehörigen Objektmodell (*Document Object Model*, *DOM*).
Die verschiedenen Ebenen einer solchen Beschreibung (das Format gestattet die Spezifikation von verschachtelten Workflows) werden ignoriert, da sie im Meta-Modell nicht dargestellt werden können. Dies ist das Resultat einer ganz bewusst gefällten Entscheidung in der Entwurfsphase, vgl. hierzu Abschnitt Metamodell. Verknüpfungen über die Grenze zweier Ebenen hinweg werden zu den atomaren Start- oder Endservices der betroffenen Teilworkflows „umgebogen“.
Beschreibungen im XScufl-Format umfassen häufig auch Angaben zu den Autoren des Workflows sowie textuelle Beschreibungen desselben. Diese werden in den Attributen des Workflow-Objektes jedoch nur dann abgespeichert, sofern sie nicht bereits gesetzt sind.

Kategorien-Parser für XML-Beschreibungen. Implementiert wurde ebenfalls ein Parser für hierarchische Beschreibungen von Kategoriestukturen in einem eigens dafür definierten XML-Format (die Schema-Definition liegt als Datei *Categories.xsd* dem Projektverzeichnis dieses Parsers bei). Er wurde umgesetzt, damit das Importmodul für EMBOSS die in der Syntaxbeschreibung für ACD-Dateien angegebene Gruppenstruktur [Emb] aus einer leicht änder- und erweiterbaren Datei einlesen kann.

[16] `http://www.w3.org/TR/wsdl`

3.2.6 Algorithmen

Algorithmen werden grundsätzlich durch Plug-ins dargestellt. Dabei ist jedoch zu beachten, dass Plug-ins gleich mehrere Algorithmen umfassen können. Jeder solche Algorithmus wird durch eine eigene Klasse repräsentiert, welche die Schnittstelle `RecommenderAlgorithm` implementiert. Über die Methoden `initialize()` und `onUpdate()` werden dem Algorithmus vom Empfehlungssystem die aktuellen Referenzen auf die Bibliotheken mitgeteilt – bei Start des Systems oder nach Durchführung einer Datenbankaktualisierung.
Am bedeutensten ist jedoch die Methode `recommend()`, deren Argumente den aktuellen Nutzerkontext beschreiben und deren Rückgabe in einer (internen) Empfehlung gemäß Abschnitt 3.1.2 besteht:

Listing 3.4: Beispiel: Kopf der `recommend()` Methode

```
Recommendation
recommend (final String serviceID,
           final String port,
           final Workflow workflow);
```

Das Argument `workflow` beschreibt dabei den Workflow aus dem Editor. Durch die beiden anderen Argumente kann die Empfehlung auf Nachfolger oder Vorgänger eines bestimmten Services innerhalb dieses Workflows oder sogar auf die eines seiner Ports eingeschränkt werden.
Entwickler von Algorithmen müssen dabei beachten, dass eines oder mehrere dieser Argument auf `null` gesetzt sein können. Im Falle der Argumente `serviceID` und `port` bedeutet dies, dass entsprechende Einschränkungen nicht gemacht werden sollen (allerdings sollte der Port nur dann spezifiziert werden, wenn dies auch auf die Service-ID zutrifft). Ist ein konkreter Service nicht spezifiziert, so soll eine „globale“ Empfehlung ausgesprochen werden – gegebenenfalls in Abhängigkeit vom Workflow.
Eine detailliertere Übersicht über die während des ersten Semesters dieser Projektgruppe implementierten Algorithmen bietet Abschnitt 3.4.

3.3 Importmodule

Die zuvor genannten Daten müssen in die Datenbank importiert werden. Zu diesem Zweck haben wir eine Struktur entwickelt, um die Daten entsprechend zu erfassen, zu analysieren und zu verwalten. Diese Struktur hat als oberste Instanz das sogenannte Importmodul. Grundlegend gibt es drei verschiedene Arten von Importmodulen, welche alle Arten von Daten die für das Projekt gebraucht werden erfassen.
Die erste Art ist das Importmodul für Services. Dieses Importmodul versucht nur, Services aus anderen Datenbanken zu erhalten und diese in der Datenbank zu speichern. Im Vergleich zum ersten Modul ist die zweite Art der Importmodule nur für die Workflows zuständig. Alle Importmodule verwenden Parser (siehe Abschnitt 3.2.5). Diese Parser sind die Arbeitseinheiten, die Informationen aus den erworbenen Daten extrahieren.
Für jeden Datentyp und für jedes System, das von uns als Datengrundlage verwendet wurde, wurde ein separater Parser geschrieben. Die Parser für Services und Workflows unterscheiden sich im Detail. Die inneren Vorgänge eines Parsers sind für das Importmodul irrelevant. Über das Importmodul erfolgt ein Aufruf der Funktion `parse()` der Parserklassen und dadurch erhält der Importer die Services oder entsprechend Workflows, die es dann in die Datenbank schreibt.
Die dritte Art des Importmoduls ist das Importmodul für Ontologien. Dies ist prinzipiell sehr ähnlich zu den anderen beiden Techniken und wird im Folgenden weiter beschrieben. Zunächst wird die Arbeitsweise der Importmodule erläutert und in Kontext mit dem Gesamtprojekt gebracht.

3.3.1 Importmodule für Services

Importmodule für Services sind Plug-ins, welche die vorhandenen Beschreibungen der Services (beispielsweise in Form einer Datei) parsen und in die Datenbank-Objekte vom Typ `Service` überführen. Für jedes Importmodul wird die Schnittstelle `ServiceImportModule` implementiert. Diese umfasst zwei Methoden `importServices()` und `importCategories()`, die jeweils Listen von `Service` bzw. `Category`-Objekten zurückgeben.
Gegenwärtig wurden zwei Importmodule für Services implementiert und in das Projekt eingebunden: ein Importmodul für die EMBOSS-Toolsuite und ein Importmodule für die Webseite BioCatalogue.

EMBOSS

Das Importmodul für EMBOSS [Emb] erzeugt eine Liste von Services aus den sogenannten ACD-Dateien, die in der EMBOSS-Toolsuite enthalten sind. Die ACD-

Dateien enthalten Informationen über die Parameter der einzelnen EMBOSS-Services. Dazu gehören: Name und Beschreibung des Services, die benötigten und optionalen Eingaben, Ausgaben und weitere optionale Parameter. Jede ACD-Datei besteht aus einigen Sektionen, die durch die Schlüsselwörter "*section*" und "*endsection*" begrenzt sind. Der gesamte Diest sowie seine Parameter werden (wenn möglich) in die EDAM-Ontologie durch das Schlüsselwort "*relation*" eingeordnet. Einige Services enthalten eine Zuordnung zu EMBOSS-Kategorien. Die Kategorien stellen eine Hierarchie von Gruppen dar. Anhand der Kategorien können Services verschiedenen Gruppen zugeordnet werden.
Alle Dateien werden zunächst heruntergeladen und in einem beliebigen Ordner gespeichert. Das Importmodul bekommt als Parameter einen Pfad zu diesem Ordner. Jede dieser Dateien wird vom Parser verarbeitet und in das Objekt `Service` überführt. Dabei werden die gleichen Parameter bzw. EDAM-Relationen auf gleiche Objekte abgebildet. Dadurch enthält die Datenbank keine Redundanzen. Jeder Service enthält eine ID, die zum Verschmelzen mit Services aus anderen Quellen dient.
Das Parsen und Hinzufügen von Kategorien kann einige Schwierigkeiten bereiten. In den ACD-Dateien kommen Gruppen in Form von Pfaden im Hierarchie-Baum vor. Die Gesamthierarchie wird mithilfe des Kategorien-Parsers erstellt (s. Abschnitt Parser). Sie wird vom Importmodul übernommen, sodass dem Service eine Kategorie aus der definierten Taxonomie zugeordnet wird.

BioCatalogue

Ein anderes Importmodul für Services ist jenes für "BioCatalogue" [Bioa]. Dieses Modul arbeitet im Gegensatz zum EMBOSS-Importmodul mit Daten die im Web bereitgestellt werden. Um an diese Daten zu gelangen stellt BioCatalogue eine umfangreiche REST-API bereit. Die API bietet eine große Menge an Endpunkten, welche die Daten zumeist als XML- und/oder JSON-File zurückliefern. Leider ist die API-Dokumentation an dieser Stelle lückenhaft, so dass teilweise recht wichtige Endpunkte nicht dokumentiert sind.
Für Informationen zu einzelnen Services steht der Endpunkt

"".../services/[id]/summary/""

bereit. Um alle Services von BioCatalogue in unsere Datenbank aufzunehmen, wird über alle IDs dieses Endpunktes iteriert und die einzelnen Dateien werden geparst. Hierzu wird die XML-Repräsentation verwendet. Die einzelnen Informationen werden in ein Service-Objekt gekapselt und in dieser Form in die Datenbank geschrieben. Diese Objekte enthalten neben grundlegenden Informationen wie Namen und Beschreibung eine große Menge weiterer Informationen, die für die einzelnen Empfehlungs-Algorithmen nützlich sind. Hierzu zählen unter anderem die Anzahl der Views oder auch von Nutzern angelegte Tags.
Zusätzlich zum eigentlichen Import der Services wurde ein Import der Kategorien der Services implementiert. Hierzu eignete sich der Endpunkt "/categories". Dieser liefert die Top-Level-Kategorien der Services von BioCatalogue zurück. Zu jeder dieser Top-Level-Kategorien wird außerdem eine Liste mit den Unterkategorien zurückgegeben. Die Kategorien können wiederum per Endpunkt aufgerufen werden

und bieten wieder eine Liste ihrer Unterkategorien. Durch sukzessives Aufrufen der Endpunke konnte also ein vollständiger Kategoriebaum aufgebaut und in der Datenbank gespeichert werden.
Nach erstmaliger Implementierung des Service-Imports trat ein recht unerwartetes Problem auf. Der Import aller Services vom BioCatalogue (weniger als 4000) dauerte mehr als 2,5 Stunden. Nach ausführlicher Überprüfung des Programmcodes stellte sich heraus, dass sich dieses Problem auf die langen Reaktionszeiten des Webservers von BioCatalogue zurückführen lässt. Um die Zeit für den Import zu verkürzen wurden einige Veränderungen am Programmcode vorgenommen. Die wesentlichste Veränderung bestand in der Nutzung von Multi-Threading beim Import der Services.

3.3.2 Importmodul für Workflows

Das Importmodul für Workflows nutzt als Datengrundlage das Online-System "myExperiment"[Mye] zur Verfügung. "myExperiment" hat keine Programmierschnittstelle, die den direkten Zugriff auf Workflows ermöglicht, um diese direkt zu erfassen. Aus diesem Grund war die Projektgruppe gezwungen, auf einem anderen Wege an die nötigen Informationen zu kommen.
Zunächst wurde das Verzeichnis mit allen Workflows aus "myExperiment" in Form einer XML-Datei heruntergeladen. In diesem Verzeichnis werden URLs zu allen Workflows wie z. B.

http://www.myexperiment.org/workflow.xml?id=4

aufgelistet. Das Importmodul liest die gespeicherte XML-Datei ein und bearbeitet die aufgelisteten Workflows. Zusätzlich zur URL werden die Informationen über den Typ des Workflows verwendet, um den entsprechenden Parser zu starten. Die Workflows unterscheiden sich im Format und zu jedem Format existiert ein Parser, der aus dem formatierten Workflow die Instanz `Workflow` erstellt.
Nachdem alle Workflows einzeln ausgelesen worden sind und die Parser die Workflows in Services aufgeteilt haben, werden die Informationen in der Datenbank (siehe Abschnitt 3.2.2) zur weiteren Verwendung gespeichert.

3.3.3 Importmodul für Ontologien

Das Importmodul für Ontologien nutzt den von "BioPortal"[Biob] zur Verfügung gestellten Webservice um an die Informationen zu gelangen. Der Webservice bietet verschiedene REST-Endpunkte. Mit Hilfe dieser Endpunkte ist es möglich Metainformationen über Ontologien sowie die Ontologien selber zu erhalten. Hierbei ist zu beachten, dass die Metainformationen in Form einer XML-Datei und Ontologien in Form einer OWL- oder oder OBO-Datei vorliegen. Bei den Ontologien werden ausschließlich die OWL-Dateien beachtet, da das zusätzliche Parsen von OBO-Dateien mit erheblichen Aufwand einher gegangen wäre.

Damit der Webservice in vollem Umfang genutzt werden kann, muss sich die Applikation gegenüber dem Webservice authentifizieren. Dazu wird ein sog. API-Key verwendet, eine Zeichenkette, die kodierte Zugangsdaten enthält. Dieser muss dem Importmodul zur Verfügung gestellt werden. Anschließend kann mithilfe des Webservices auf benötigte Informationen zugegriffen werden. Die Ontologien werden über Identifikationsnummern (IDs) ausgewählt. Die IDs werden von BioPortal selber vergeben. Mithilfe dieser IDs kann das Importmodul alle Metainformationen und die Ontologie selber laden. Das Importmodul übernimmt in diesem Fall auch den Vorgang des Parsens. Hierbei werden die Metainformationen mithilfe des Java-internen DOM-Parsers geparst wohingegen für die OWL-Dateien auf einen externen Parser zurückgegriffen wird. Bei diesem Parser handelt es sich um die OWL-API [Owl]. Mithilfe dieser API können OWL-Dateien einfach geparst werden. In diesem speziellen Fall werden die benötigten Informationen ausgelesen und als das vom Recommender bereitgestellte Modell für Ontologien zurückgegeben. Das Speichern wird entsprechend von einem Datenbankcontroller übernommen.

3.4 Algorithmen

Kommen wir jetzt zum Herzstück der Funktionsweise des Empfehlers selbst: In diesem Abschnitt werden die einzelnen Algorithmen vorgestellt, die dem von uns entwickelten Empfehlungssystem zum Zwecke der Generierung seiner Empfehlungen zur Verfügung stehen.

3.4.1 Association Rule Mining

Wie bereits in Kapitel 2.3.4 beschrieben, lässt sich *Association Rule Mining* anwenden, sobald ein beobachtetes Verhalten als eine Menge von *Transaktionen*, die wiederum Mengen von *Entitäten* entsprechen, vorliegt. In dem gegebenen Kontext liegt es nahe, Services als *Entitäten* zu betrachten und bereits vorhandene Workflows als *Transaktionen*. Der naive Ansatz, einen Workflow in eine *Transaktion* umzuwandeln, besteht darin, den Workflow nur als Menge seiner Services anzusehen und die Information der Verbindungen dazwischen zu vernachlässigen. Aus den beschriebenen Quellen für Workflows lässt sich mit dem naiven Ansatz eine Transaktionen-Datenbank aufbauen und mit Hilfe der in Kapitel 2.3.4 vorgestellten Mittel analysieren. Die zur Analyse benötigten Parameter *minimale Confidence* und *minimaler Support* sind dabei im Profil des Algorithmus einstellbar. Die auf diese Weise aus vorhandenen Workflows entstandenen *Assoziationsregeln* haben die Form $\{s_{i_1}, \ldots, s_{i_n}\} \rightarrow \{s_{o_1}, \ldots, s_{o_m}\}$. Eine solche Regel wird interpretiert als: wenn in einem Workflow Services $s_{i_1}, \ldots, s_{i_n}$ vorkommen, dann ist es wahrscheinlich, dass in ihm auch Services $s_{o_1}, \ldots, s_{o_m}$ vorkommen. Um eine Empfehlung abzugeben, wird jede Regel auf ihre Anwendbarkeit im aktuellen Workflow überprüft und ggf. angewandt, d.h. wenn im aktuellen Workflow Services $s_{i_1}, \ldots, s_{i_n}$ vorkommen, dann werden Services $s_{o_1}, \ldots, s_{o_m}$ empfohlen. Die Relevanz eines auf diese Weise empfohlenen Services entspricht dabei der *Confidence* der dafür angewandten Regel. Wenn mehrere Regeln den gleichen Service empfehlen, dann entspricht die Relevanz dem Maximum der *Confidences* der in Frage kommenden Regeln.

3.4.2 Feedback

Wie in Kapitel 2.3.3 beschrieben, kann mit Hilfe von Feedback seitens des Benutzers eine Empfehlung verbessert werden. Diesen Ansatz verfolgt der *Feedback Recommender*. Zunächst erzeugt dieser eine vorläufige Empfehlung basierend auf vorliegenden Service- und Workflowinformationen. Eine detaillierte Beschreibung der Erzeugung einer vorläufigen Empfehlung befindet sich im Abschnitt ‘Erzeugung vorläufiger Empfehlung des Feedback Recommenders‘. Danach wird dem Benutzer die

Möglichkeit gegeben, die vorläufige Empfehlung zu bewerten. So kann er vorläufig gewählte Services in Kategorien *relevant*, *irrelevant* oder *keine Aussage* einteilen (siehe Abbildung 3.8).

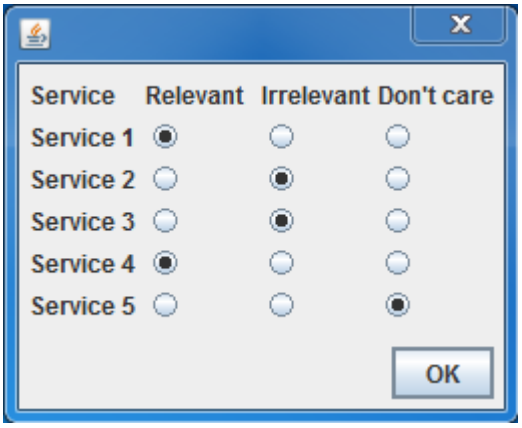


Abbildung 3.8: Benutzer-Feedback

Mit Hilfe dieser Einteilung lässt sich die Empfehlung mittels *Relevance Feedback (Rocchio)* (Kapitel 2.3.3 Algorithmus 2) verfeinern. Anschließend wird die verfeinerte Empfehlung ausgegeben. Die benötigten Gewichte α, β, γ lassen sich im Profil des Algorithmus einstellen. Das Feedback des Benutzers ist notwendig für die Ausführung des Algorithmus. Da die Klasse `RecommenderSystem` keine Möglichkeit der notwendigen Benutzerinteraktion vorsieht, implementiert der Algorithmus selbst die entsprechende Funktionalität.

Erzeugung vorläufiger Empfehlung des Feedback Recommenders

Der in Kapitel 2.3.3 vorgestellte Algorithmus 2 (*Relevance Feedback (Rocchio)*) schreibt weder vor, wie der zugrundeliegende Vektorraum aufgebaut werden soll, noch wie eine vorläufige Empfehlung zu erzeugen ist. Es wird nur verlangt, dass sowohl die Anfrage, auf deren Grundlage eine vorläufige Empfehlung erzeugt wird, als auch die Menge empfehlbarer Services sich als Elemente eines $\mathbb{Q}$-Vektorraums darstellen lassen.
In der Implementierung wird als $\mathbb{Q}$-Vektorraum der unendlich-dimensionale Vektorraum $V = \mathbb{Q}^{([a-z]+[0-9])^*}$ eingesetzt, wobei $([a-z]+[0-9])^*$ der Menge beliebig langer Zeichenketten über a bis z und 0 bis 9 entspricht.
Ein Service s mit der Beschreibung d_s und Tags $t_1, \ldots, t_s$ wird auf folgende Weise auf ein Element v_s im Vektorraum V abgebildet:

Algorithmus 4 *Abbildung von Services in V*
Eingabe: *Service s mit der Beschreibung d_s und Tags $t_1, \ldots, t_n$*

1. *Konkateniere d_s mit durch Leerzeichen getrennten Tags $t_1, \ldots, t_n$.*

2. *Wandle Großbuchstaben in d_s in Kleinbuchstaben um.*
3. *Ersetze in d_s alle Zeichen, die keine Kleinbuchstaben oder Zahlen sind, durch Leerzeichen.*
4. *Wandle d_s in eine Multimenge d'_s um, die genau diejenigen Zeichenketten aus $([a-z]+[0-9])^*$ enthält, die in d durch mindestens ein Leerzeichen getrennt sind.*
5. *Entferne alle Zeichenketten aus d'_s, die zwei oder weniger Zeichen enthalten.*
6. *Ermittle zu jeder Zeichenkette z aus d'_s ihre Relevanz $r_z \in \mathbb{Q}$ mittels des TF-IDF-Verfahrens (Algorithmus 1).*

Ausgabe: *Vektor* $v_s = z \mapsto \begin{cases} r_z & \text{falls } z \in d'_s \\ 0 & \text{sonst} \end{cases}$

Ein Workflow w mit Services $(s_1, \ldots, s_m)$ wird auf
$v_w = \frac{1}{m} * \sum_{i=1}^{m} v_{s_i}$ abgebildet.
Eine Anfrage q bestehend aus dem aktuellen Service s und dem aktuellen Workflow w wird auf $v_q = v_s + v_w$ abgebildet.
Um eine vorläufige Empfehlung abzugeben, wird die Nachbarschaft der abgebildeten Anfrage v_q betrachtet. Als Abstandsbegriff wird *Cosinus-Similarity* (siehe Definition 1) verwendet.

3.4.3 Syntaktischer Filter

Der syntaktische Filter betrachtet lediglich die Eingabe- und Ausgabeparameter der Services. Ein Service wird genau dann empfohlen, wenn seine Eingabeparameter zu der Ausgabe des letzten Services im aktuellen Workflow passen. Abbildung 3.9 zeigt, dass der syntaktische Filter zum aktuellen Service mit Ausgabeparameter "sequence" den Service mit Eingabe "sequence" als Empfehlung auswählt.
Falls der Ausgabe-Port des aktuellen Services unbekannt ist, werden alle Ausgabeparameter untersucht. Es werden Services empfohlen, die zu mindestens einem Ausgabe-Port passen. Wenn der Kontext-Service nicht bekannt ist (z.B. am Anfang eines Workflows), werden alle Services empfohlen, da alle gleichermaßen syntaktisch passen.
Der syntaktische Filter kann mithilfe der Option *Filterungsart* angepasst werden. Beim *strengen Filtern* werden Parametertypen direkt miteinander verglichen (Eingabetyp muss dem Ausgabetyp genau entsprechen). Bei der *schwachen Filterung* wird jedem Parametertyp die Oberkategorie zugeordnet ("String" und "Text" wer-

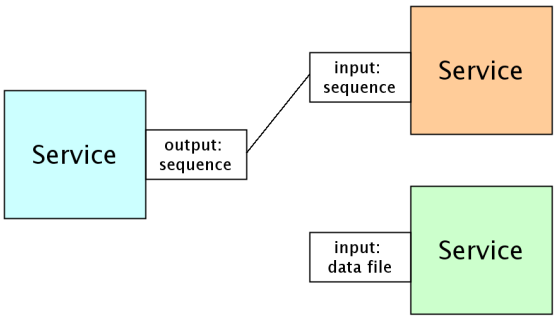


Abbildung 3.9: Funktionsweise des syntaktischen Filters.

den beispielsweise auf "Text" abgebildet). Danach werden die Oberbegriffe miteinander verglichen. Somit entsteht eine Taxonomie von Parametertypen, welche entweder vom Nutzer stammt oder aus den importierten Ontologien entnommen werden kann.

3.4.4 TF-IDF

In Kapitel 2.3.3 wurde das mathematische Modell des *TF-IDF-Verfahren* vorgestellt. Auf die Implementierung dieses Verfahrens wird in diesem Abschnitt eingegangen. Das TD-IDF Verfahren basiert auf dem Vergleich von Texten. Es ist anzunehmen, dass nahezu alle Services, die durch die unterschiedlichen Portale und Datenbanken in unsere Datenbank eingepflegt worden sind, mit entsprechenden Beschreibungstexten versehen sind. Diese Texte dienen dem Benutzer als Information zu einem Service und lassen sich als Datengrundlage für den TF-IDF Algorithmus nutzen.
Zunächst werden dazu die Beschreibungen der Services von Stoppwörtern befreit. Das heißt Wörter wie „and, if, when...“, die sehr häufig in der englischen Sprache vorkommen, werden entfernt, da diese das Vorkommen relevanter Wörter prozentual verkleinern. Um die Texte für den Algorithmus zu optimieren, werden des Weiteren alle Buchstaben kleingeschrieben und alle Satz- sowie Sonderzeichen entfernt. Nach diesem Schritt werden die Beschreibungstexte als Vektoren repräsentiert. Die Einträge des Vektors eines Beschreibungstextes entsprechen den mittels des TF-IDF-Verfahrens ermittelten Relevanzen der Worte im Beschreibungstext. Als Korpus (Eingabe des TF-IDF-Verfahrens) wird die Menge der Beschreibungstexte verwendet. Zur Erzeugnung einer Empfehlung bezüglich eines Services, wird die Nachbarschaft (Kapitel 2.3.3) seines Beschreibungsvektors bestimmt. Als Maß der Relevanz der empfohlenen Services wird die *Cosinus-Similarity* (Definition 1) verwendet.

3.4.5 Category Filter

Eine weitere Art eine Empfehlung zu geben ist die Empfehlung nach der Kategorie eines Services. Wie im Kapitel 3.1.1 beschrieben. haben Services eine Reihe von Eigenschaften, die diese charakterisieren. Dazu gehören eine Identifikationsnummer

(kurz „Id“), eine Beschreibung und noch weitere Eigenschaften. Unter diesen Eigenschaften gibt es auch eine Zuteilung zu einer Kategorie. Kategorien sind hierarische Unterteilungen des Fachgebietes in kleinere Unterthemen. In dem behandelten Themengebiet der Bioinforamtik sind als Beispiel die Kategorien „Sequenzanalyse“ oder „Chemoinformatik“ zu nennen. Bei der Entwicklung des „Category Filters“ wurden die Kategorien als Ähnlichkeitsmaß verwendet. Bei dem Algorithmus werden die Kategorien der verwendeten Service miteinander verglichen. Je mehr Überschneidungen es zwischen den Kategorien gibt, desto wahrscheinlicher ist ein Zusammenhang zwischen zwei Services. Dabei gilt, dass ein Service höher bewertet wird, wenn zum Beispiel zwei Kategorien übereinstimmen anstatt von einer Kategorie. Der Algorithmus überprüft die Kategorien eines Services und sucht nach Services die eine Schnittmenge zu den Kategorien des Ausgangsservices haben. Dabei wird eine Empfehlung nach folgendem Maß gegeben.

Definition 8 *Kategorieähnlichkeitsmaß*

Empfehlung: $f(x)$
Anzahl der Kategorien eines Service: $NUM(x)$
Anzahl der Übereinstimmenden Kategorien der Services A und B: $SUM_{NUM}(A, B)$

$$f(x) = \frac{1/NUM(B)+SUM_{NUM}(A,B))}{max((1/NUM(B)+SUM_{NUM}(A,B))}$$

Die Ähnlichkeit zweier Services wird über ein Stringmatching der Kategorien realisiert. Das Stringmatching ist binär realisiert. Es gibt keine fast ähnlichen Services, sondern es wird strikt getrennt. Das heißt entweder eine Kategorie stimmt überein oder nicht. Durch diese Vorgehensweise werden die Ergebnisse einer Recommandation, sehr stark dezimiert. Dies soll garantieren, dass wirklich nur Services vorgeschlagen werden, die einen Zusammenhang besitzen. Es ist durchaus realistisch, dass zwei Services aus dem gleichen Themengebiet einen logischen Zusammenhang in einem Workflow besitzen. Von dem anderen Standpunkt aus betrachtet, ist es unwahrscheinlich, dass zwei Services aus verschiedenen Themengebieten bzw. Kategorien einen sinnvollen und logischen Zusammenhang in einem Workflow finden. Die Ergebnisse zur Güte des Algorithmus auf den verwendeten Testdaten finden sich in Kapitel 5.1.3 wieder.

3.4.6 Mutual Information

Die im Rahmen dieser Projektgruppe entwickelte Umsetzung eines Empfehlungsverfahrens, das auf dem Prinzip der „Mutual Information“ basiert, richtet sich grob nach den in Abschnitt 2.3.3 angegebenen Definitionen, weicht aber in einigen Details davon ab oder spezifiziert sie genauer.

Eingabe

Von den drei dem Algorithmus übergebenen Parametern (aktueller Service, Port und Workflow-Kontext) wird nur von ersterem Gebrauch gemacht. Sollte die ID des angegebenen Dienstes nicht in der Workflow-Datenbank verzeichnet sein, so wird eine leere Empfehlung erzeugt. Andernfalls wird eine Empfehlung gemäß der berechneten Verhältnisse $MI(i, j)$ zwischen dem aktuellen Dienst i und allen anderen registrierten Diensten j erzeugt.
Als Empfehlung ergibt sich also eine Abbildung $E_i : \mathcal{S} \rightharpoonup \mathbb{R}_0^+$ mit $E_i(j) := MI(i, j)$.

Verhältnis zur theoretischen Definition

In der Implementierung soll eine einzige Merkmalsausprägung bestimmt werden, welche mit der Wahrscheinlichkeit des gemeinsamen Auftretens zweier Dienste innerhalb der Workflowgrenzen korreliert, sodass der im Theorie-Abschnitt angegebene Koeffizient (dessen Summe auf einen einzigen Summanden reduziert wird) außer Acht gelassen werden kann. Stattdessen konzentriert sich diese Darstellung auf die Berechnung des Maßes $MI(i, j)$ für zwei Dienste i und j. Der Einfachheit halber ist im Folgenden noch einmal die ursprüngliche Definition dieser Abbildung angegeben (vgl. Seite 23):

$$MI(i,j) = \begin{cases} \operatorname{ld}\left(\frac{p(i,j)}{p(i)\cdot p(j)}\right), & \text{falls } i \neq j \\ 1, & \text{sonst} \end{cases}$$

Wahrscheinlichkeitsverteilung. Die dabei beteiligte Wahrscheinlichkeitsverteilung p gibt für zwei Dienste i, j an, wie wahrscheinlich das gemeinsame Auftreten beider Dienste innerhalb eines Workflows ist, wenn man die Gesamtheit aller Workflows betrachtet. Dieser Wert ist dann gerade $p(i, j)$. Entsprechend ergibt sich die Wahrscheinlichkeit für das Auftreten eines einzelnen Dienstes i (über der Menge aller Workflows) *im einfachsten Falle* als Randwahrscheinlichkeit $p(i)$.
Eine erste Abweichung von obiger Definition weist das implementierte Verfahren für den Fall identischer Dienste ($i = j$) auf. Anstatt auf den Wert 1 abzubilden, bleibt eine Empfehlung in diesem Fall ganz aus, da es im Regelfall nicht sinnvoll erscheint, den Ausgangsservice als (unmittelbaren) Nachfolger vorzuschlagen.
Während die formale Definition in Hinsicht auf die Anzahl der Argumente keinen Unterschied zwischen beiden Verteilungen macht, werden sie in der algorithmischen Umsetzung durch zwei getrennte Datenstrukturen repräsentiert (als partielle Abbildungen von (Paaren von) Service-IDs in die reellen Zahlen, genauer als `Map` mit Wertebereich `Double`).

Caching. Bei erstmaligem Aufruf des Algorithmus und jeder Aktualisierung der Datenbank durch das Empfehlungssystem werden diese Datenstrukturen neu berechnet und im Dateisystem abgespeichert, sodass sie beim nächsten Start von dort gelesen werden können und nicht ein weiteres Mal berechnet werden müssen.

Konfigurierbarkeit. Während die theoretische Definition symmetrisch in ihren Argumenten erscheint und sich die Wahrscheinlichkeitsverteilung $p(i)$ wie oben erwähnt

als Randverteilung ergibt, können bei der Verwendung des Algorithmus durch Anpassung verschiedener Parameter (vgl. Tabelle 3.6) Abweichungen von diesem Verhalten erreicht werden.

Name	Wertebereich	Standardwert
`maxPredecessorLevel`	$\{-1\} \cup \mathbb{N}_0$	-1
`maxSuccessorLevel`	$\{-1\} \cup \mathbb{N}_0$	-1
`distanceFactor`	$(0; 1] \subset \mathbb{R}$	1,0

Tabelle 3.6: Konfigurierbare Parameter des Algorithmus „Mutual Information“

Sobald einer dieser Parameter vom jeweiligen Standardwert abweicht, ergibt sich die Verteilung für einzelne Dienste nicht weiter als Randverteilung, da dann im Allgemeinen für einige Dienste $i \in \mathcal{S}$ die nachstehende Ungleichung echt ist:

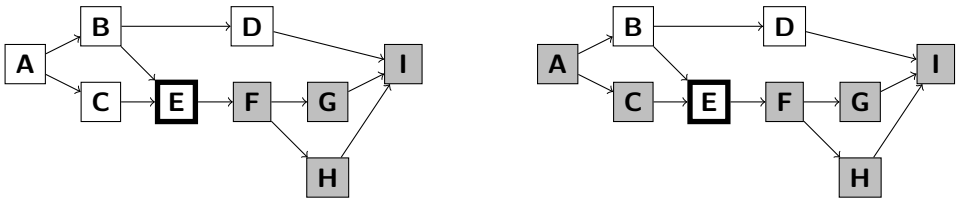
$$p(i) \;\geq\; \sum_{j \in \mathcal{S}} p(i, j).$$

Da die Berechnung der Verteilung p für einzelne Dienste auf kanonische Weise geschieht und von der Festlegung der Parameter nicht betroffen ist, beschränken sich die weiteren Betrachtungen auf die Berechnung von $p(i, j)$ für *Paare* von Service-IDs und deren Abhängigkeit von der Wahl der Parameter.

Die beiden Parameter `maxPredecessorLevel` und `maxSuccessorLevel` schränken den Suchraum innerhalb eines Workflows ein und zwar in Rückwärts- bzw. Vorwärtsrichtung. Diesem Ansatz liegt die Idee zugrunde, dass das gemeinsame Auftreten zweier Dienste in einem Workflow bei größerer Distanz (minimale Anzahl der Datenflussverbindungen in der grafischen Repräsentation) möglicherweise als unerheblich erachtet werden soll. Wird einer der Parameter auf den Wert -1 gesetzt, so ist der Suchraum in der entsprechenden Richtung unbegrenzt, bei einem positiven Wert ℓ werden ausgehend von Service i maximal Pfade der Länge ℓ verfolgt und bei Angabe des Wertes 0 wird die Suche in der betroffenen Richtung überhaupt nicht durchgeführt, vgl. Abbildung 3.10.
Die Festlegung auf einen einen bestimmten positiven Wert mag aus Nutzersicht schwierig, vielleicht gar wenig sinnvoll erscheinen. Als das wesentlichere Merkmal wird vom Autor dieses Algorithmus stattdessen die Möglichkeit zur Einschränkung der Suche in *eine Richtung* angesehen. Offensichtlich führt das im Allgemeinen zu einer in ihren Argumenten asymmetrischen Verteilung $p(i, j)$.

Erweitert und verfeinert werden kann diese Idee durch Festlegung des Parameters `distanceFactor` auf einen von der Standardkonfiguration abweichenden Wert. In einem solchen Falle wird der Anteil des gemeinsamen Auftretens zweier Dienste i, j innerhalb eines Workflows an der „Gesamt“-Wahrscheinlichkeit $p(i, j)$ gemäß der Entfernung zwischen ihnen vermindert. Diese Minderung ist exponentiell in der Distanz, wobei die Basis gerade durch den Parameter `distanceFactor` gegeben ist.[17]

[17] Seinen Namen verdankt der Parameter `distanceFactor` dem Umstand, dass die Potenz einen Faktor definiert, welcher mit dem für den aktuellen Workflow eigentlich ermittelten Wert multipliziert wird.



(a) Teilbeschränkter Suchraum: `maxPredecessorLevel`=0, `maxSuccessorLevel`=−1

(b) Unbeschränkter Suchraum: `maxPredecessorLevel`=−1, `maxSuccessorLevel`=−1

Abbildung 3.10: Suchräume für den aktuellen Dienst E (der selbst nicht dazu gehört) in einem Beispielworkflow

Bei Wahl eines Parameterwertes von 1/2 ergibt sich bei einem Nachfolger über drei Kanten (sofern dieser Teil des Suchraumes ist) also ein Faktor von $(1/2)^{3-1} = 1/4$. Ein Beispiel dafür ist der Dienst I in Abbildung 3.11. Für selbige Abbildung ist im Folgenden die berechnete Empfehlung (als partielle Abbildung) für den gewählten Parameterwert 1/2 in Tabellenform angegeben:

Service	A	C	F	G	H	I
Wertung	1/2	1	1	1/2	1/2	1/4

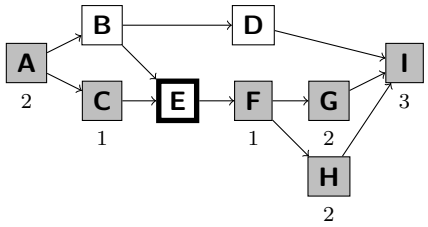


Abbildung 3.11: Distanzen vom aktuellen Dienst E für markierte Dienste

Es erscheint natürlich, solchen Diensten, die in der Workflow-Datenbank nah beieinanderliegen, eine höhere Bewertung zu erteilen als jenen, die weit voneinander entfernt sind. Zu diesem Behufe kann der Parameterwert aus dem (offenen) Intervall $(0; 1)$ gewählt werden. Die ebenfalls gültige Wahl von 1 führt offenbar zu keiner Änderung der Bewertung. Welche Werte dabei allerdings zu guten Resultaten führen, bleibt zu evaluieren.

3.4.7 Kollaborative Algorithmen

In Ergänzung zu den inhaltsbasierten Algorithmen der vorangegangenen Abschnitte wird nun auf kollaborative Algorithmen eingegangen, welche eine Empfehlung anhand rein statistischer Daten über Benutzer des Portals “BioCatalogue” erstellen. Bei der kollaborativen Empfehlung wird auf die Nutzung der semantische Eigenschaften von Services verzichtet, stattdessen werden Benutzerpräferenzen und -bewertungen untersucht.

Order By Views

Dieser Algorithmus sortiert Services nach Anzahl der Klicks auf deren Beschreibungen und wählt die Services aus, die von vielen Nutzern angeklickt werden. Dass viele Nutzer einen bestimmten Dienst für ihre Workflows auswählen, deutet auf gute Qualität dieses Dienstes. Obwohl hier keine semantischen Informationen miteinbezogen werden, kann dieser Algorithmus in Kombination mit einem semantischen Filter gute Ergebnisse erzielen, denn semantisch passende Services werden nach Qualität und Beliebheit bei Nutzern sortiert.

Service User Association

Dieser Algorithmus basiert auf der Idee, dass Nutzer, die ähnliche Workflows erstellen, ähnliche Dienste wählen. Zu einem gegebenen Service werden Services empfohlen, welche von Nutzern mit ähnlichen Präferenzen häufig benutzt oder gut bewertet wurden. Dies geschieht in zwei Schritten. Im ersten Schritt werden Nutzer ermittelt, welche ähnliche Services benutzt haben. Danach werden alle Services durchgegangen und für jeden Service wird die durchschnittliche Bewertung durch die ausgewählten Nutzer berechnet. Wenn ein bestimmter Service von vielen der ausgewählten Nutzer gut bewertet bzw. favorisiert wird, enthält dieser einen höheren Gewicht in der Empfehlung.

Die Informationen über explizite Nutzerbewertungen enthalten die Spalten “Rating” und “Favourite”, Tabelle “Service”. Wenn der letzte Service im aktuellen Workflow von keinem anderen Nutzer bewertet wurde, werden alle Services mit 1.0 bewertet (alle Services werden empfohlen) In diesem Fall können andere Algorithmen und Filter eine aussagekräftige Bewertung ausgeben, wie zum Beispiel Workflow User Association.

Workflow User Association

Dieser Algorithmus baut auf dem vorherigen Algorithmus auf. Als Eingabe wird der Funktion `recommend()` der bisher aufgebaute Workflow übergeben. Der Algorithmus untersucht alle Services aus dem aktuellen Workflow und für jeden Service wird eine Empfehlung mittels **“Service User Association”** erzeugt. Einzelne Empfehlung werden aufsummiert und normalisiert (Bewertung zwischen 0 und 1).

Die Methode weist einen Vorteil gegenüber dem vorherigen Verfahren auf: Die Betrachtung des gesamten Workflows erhöht die Wahrscheinlichkeit, dass Bewertungen in der Datenbank gefunden werden können.

4 Integration

Nachdem im ersten Semester der Projektgruppe das Recommender-System für Services zu einem wesentlichen Teil entworfen und implementiert worden war, sollte es im zweiten Semester vor allem darum gehen, das entwickelte System in eine bestehende Workflowmanagementumgebung einzubinden.
In diesem Kapitel wird dieser Integrationsvorgang dokumentiert, wobei alle wichtigen Entscheidungen festgehalten und erläutert werden, die während der Bearbeitung der Kernaufgaben getroffen wurden. Zu diesen Aufgaben zählen insbesondere die Wahl eines geeigneten Workfloweditors, sowie Entwurf und Implementierung einer angemessenen grafischen Benutzeroberfläche für den Recommender.
Schließlich soll noch beschrieben werden, wie genau das Recommender-System aus technischer Sicht in die gewählte Workflowmanagementumgebung integriert werden konnte und welche Hindernisse bei diesem Vorgang zu überwinden waren.

4.1 Wahl eines Workfloweditors

In diesem Abschnitt soll mit *jABC* die Workflowmanagementumgebung vorgestellt werden, die von den Teilnehmern der Projektgruppe als Plattform für das entwickelte Empfehlungssystem ausgewählt wurde. Dabei werden alle Features und Eigenheiten des Tools erläutert, die für die Integration und die Ausführung des Recommenders relevant sind.
Zunächst wird jedoch noch einmal die Alternative *Eclipse4Bio* diskutiert, die sich letztlich bei unserer Auswahl nicht durchsetzen konnte. Nach Möglichkeit wird hierbei kurz auf die konkreten Gründe eingegangen, die gegen eine Entscheidung für diesen Editor gesprochen haben.

4.1.1 Entscheidung gegen Eclipse4Bio

Ursprünglich war für die PG die Verwendung der Plattform *Eclipse4Bio* angedacht, die Rahmen der PG 558 entwickelt worden war [55812]. Diese Plattform wurde

dementsprechend früh in der Planung berücksichtigt, aber auch relativ früh für die weitere Verwendung ausgeschlossen. In diesem Abschnitt sollen die wesentlichen Eigenschaften von *Eclipse4Bio* vorgestellt und hinsichtlich unserer Anforderungen erläutert werden.

Bei der Software *Eclipse4Bio* handelt sich nicht nur um eine Management-, sondern gleichzeitig auch um eine Ausführungsumgebung für Workflows. Das bedeutet, dass die Prozesse in der durch den Workflow spezifizierten Reihenfolge ausgeführt werden können.

Eclipse4Bio war zunächst als Zielplattform des von uns entwickelten Empfehlungssystems vorgesehen. Es schien eine logische Wahl zu sein, da unsere Projektgruppe somit auf der Arbeit einer anderen Projektgruppe hätte aufbauen können. Leider hat sich schon während der Seminarphase herausgestellt, dass das Programm deutlich weniger ausgereift ist als entsprechende Alternativen wie *jABC* und *Taverna*, wodurch sich die Bedienung zum Teil recht umständlich gestaltet.

Die Dokumentation ist ebenfalls weniger ausführlich gehalten, was neben der Nutzung auch eine mögliche Erweiterung erschwert.

Weiterhin muss *Eclipse4Bio* immer als Java-Projekt innerhalb einer speziell konfigurierten *Eclipse*-Entwicklungsumgebung ausgeführt werden, da eine Standalone-Fassung vor Abschluss des Projektes nicht mehr fertig gestellt werden konnte. Dies stellt infrage, ob die Software jemals eine nennenswerte Nutzerzahl erreichen wird.

Eine Besonderheit von *Eclipse4Bio* ist die Möglichkeit, Workflows sowohl textuell als auch grafisch darstellen zu können, um dem Nutzer hier die Wahl der von ihm bevorzugten Methode zu überlassen. Diese Wahlmöglichkeit wurde insbesondere vor dem Hintergrund realisiert, dass Bioinformatiker die Zielgruppe darstellen, da in diesem Bereich mit Biologen und Informatikern Nutzer aufeinandertreffen, die möglicherweise unterschiedliche Vorlieben besitzen.

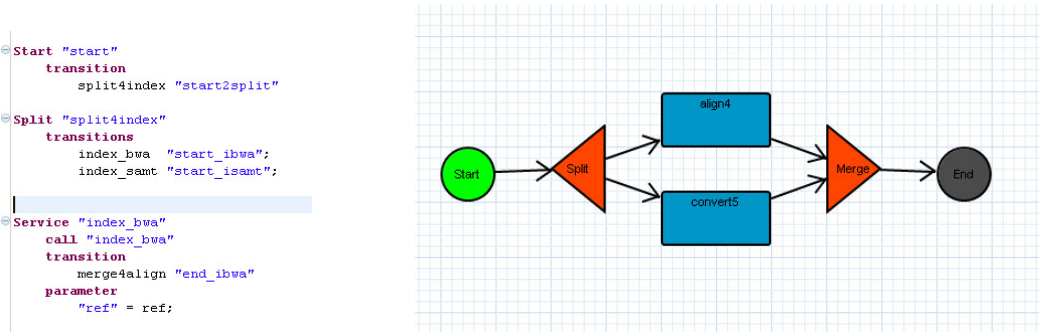


Abbildung 4.1: Textueller und grafischer Editiermodus für Workflows in *Eclipse4Bio*

Sowohl für die Definition von Services in ausschließlich textueller Form, als auch für die beiden genannten Varianten der Workflow-Definition wurden von der Projektgruppe *Eclipse4Bio* eigene Beschreibungssprachen entwickelt. Dieses Vorgehen erweist sich jedoch in Bezug auf die Ziele der gegenwärtigen Projektgruppe als problematisch, da beispielsweise die Services und Workflows der Plattformen *BioCatalogue*, *EMBOSS* und *MyExperiment* nicht in diesem Format vorliegen und daher

zunächst entsprechend angepasst werden müssten, um sie innerhalb von *Eclipse4Bio* verwenden zu können.
Auch bei dem letztendlich gewählten *jABC* sind zwar derartige Anpassungen nötig, jedoch sind diese weniger gravierend, da wir bei *jABC* insbesondere auch auf die mögliche Auführung der Workflows verzichten. Eine genauere Beschreibung der Anpassungen folgt in Abschnitt 4.3.1.
Eclipse4Bio bietet somit auf der einen Seite weitergehende Funktionalitäten, als sie für die von uns angedachte Erweiterung erforderlich wären, löst auf der anderen Seite die für uns relevanten Aufgaben auf eine umständlichere Art und Weise – insbesondere im Vergleich zu den Konkurrenzprodukten, welche außerdem von einer größeren Anzahl von Nutzern verwendet werden.
Abschließend betrachtet divergieren die Stärken von *Eclipse4Bio* und die Anforderungen der gegenwärtigen Projektgruppe deutlicher als dies im Vergleich mit den anderen Wahlmöglichkeiten *jABC* und *Taverna* der Fall ist.

4.1.2 Entscheidung für Java Application Building Center

Bei dem *Java Application Buildung Center* (*jABC*) handelt es sich um ein modulares Framework zur Modellierung von komplexen Software-Systemen. Letztere können hierbei als Workflows in einer grafischen Benutzeroberfläche erstellt und bearbeitet werden.
Im Folgenden werden einige interessante Aspekte der Software im Detail vorgestellt und die Stärken und Vorzüge der Plattform, aufgrund derer sich die Projektgruppe für *jABC* entschieden hat, hervorgehoben.

Benutzeroberfläche

Die Benutzerschnittstelle von *jABC* ist übersichtlich und intuitiv verständlich aufgebaut. Ähnlich wie bei anderen bekannten Programmen im Bereich der Softwareentwicklung, wie zum Beispiel *Eclipse*, ist die Arbeitsfläche in verschiedene Unterfenster aufgeteilt (vgl. Abbildung 4.2). Ganz oben befinden sich eine Toolbar und ein Menüsystem, wodurch der Nutzer Standardfunktionen schnell auswählen kann.
Weiterhin ist zu erkennen, dass sich im Hauptbereich oben links der sogenannte *Project Explorer* mit den beiden Tabs `Projects` und `SIBs`, mit dessen Hilfe man durch die vorhandenen Daten und Ordnerstrukturen navigieren kann, befindet. Workflows können dort per Doppelklick geöffnet werden. Sie werden dann grafisch auf der rechten Seite des Hauptbereiches im Fenster *Graph Canvas* dargestellt. Per Drag and Drop können sie ebenfalls zur Bearbeitung in diesen Bereich gezogen werden.
Die dritte Komponente des Hauptfensters befindet sich unten links und wird *Inspector-Panel* genannt. Hier können zum Beispiel verschiedene Informationen zu den im Graphen aktuell selektierten Objekten abgefragt werden. Als Beispiele dienen hier der SIB-Inspektor, welcher für das Bearbeiten der SIB-Eigenschaften vorhanden ist, und der Graph-Inspektor, mit dessen Hilfe grundlegende Eigenschaften des Modells (z. B. der Name) geändert werden können.

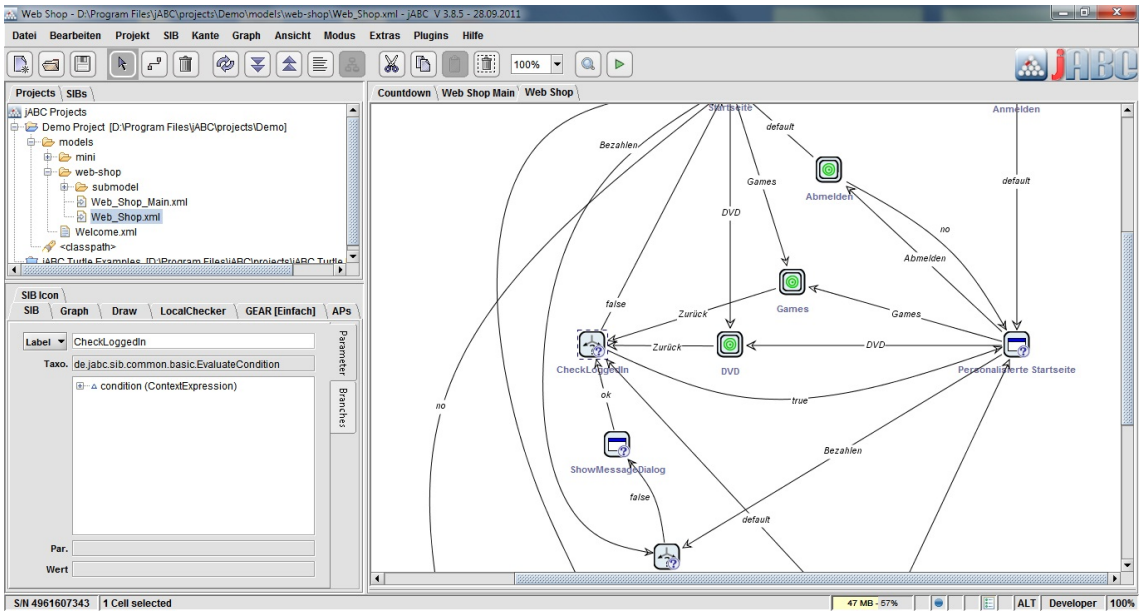


Abbildung 4.2: jABC Benutzeroberfläche

Service Independent Building Blocks

Allgemein. Im Framework *jABC* werden Service-Entitäten als *Service Independent Building Blocks*, (*SIBs*) bezeichnet. Dieser Begriff wurde vom Vorgänger *ABC* (*Agent Building Center*) geprägt. Jedoch ist der entscheidende Unterschied, dass hier SIBs, im Gegensatz zum *ABC*, nicht über mehrere Dateien und Formate definiert sind, sondern versucht wird, alle notwendigen Informationen und Definitionen in einer Datei zusammenzufassen. Somit sind z. B. Inkonsistenzen durch fehlende Dateien ausgeschlossen. In der Umgebung von *jABC* ist seitens der Entwickler die Definition

> „Eine SIB-Komponente ist eine Java-Byteklasse, die mit der Annotation `@SIBClass` gekennzeichnet ist“ [Nag, S. 42]

geläufig. Damit der Zugriff auf die SIB gesichert ist, wird die Annotation `@SIBClass` mit einer *UID* (*Unique Identifier*) versehen, so dass auch bei Änderungen der Klassen oder Pakete die Referenz zwischen dem Modell und dem SIB aufrechterhalten bleibt. Somit muss bei Änderungen in diesem Bereich keine Neureferenzierung stattfinden. Die Sprachkonvention ist an die von Java angelehnt und kann dadurch leicht auf diese abgebildet werden. Folgende Tabelle aus [Nag, S. 43] zeigt einen Ausschnitt aus den Konventionen.

SIB	**Java-Klassen**
Name	Klassenname
Taxonomie	Paket der Klasse
Parameter	Felder der Klasse

Durch die Nutzung der Sprache Java können viele javaspezifische Standardmechanismen genutzt werden und es resultieren beispielsweise folgende Eigenschaften für das Komponentenmodell:

- der Java-Compiler kann zur Erzeugung der SIBs genutzt werden;
- die Plattformunabhängigkeit seitens Java wird weitergegeben;
- die Erweiterung der Funktionalität durch Java-Bibliotheken ist möglich.

Weiterhin ist die Anlehnung an Java dadurch zu erkennen, dass die SIBs in Paletten gebündelt und in JAR-Dateien gespeichert werden.
Für wichtige Funktionen existiert im *jABC* ein sogenannter SIB-Container, der zur Interaktion mit den SIB-Klassen vorgesehen ist. Diese werden z. B. dazu verwendet, das Neuladen der SIBs zu realisieren. Einige dieser Funktionen sind das *Scannen*, welches zum Parsen von Eigenschaften für die Identifikation der SIBs vorhanden ist, der `Refresh`, welcher für das Neuladen nach Änderungen vorgesehen ist und viele mehr. Auch die spätere Objekterzeugung und die damit verbundene Struktur wird durch diesen Container bereitgestellt. Ein solches Objekt hat vier feste Bestandteile. Diese sind die oben definierte `SIB-Klasse`, der `SIBClassWrapper`, welcher Funktionalitäten zur Aktualisierung der SIB-Parameter bereitstellt, die `SIBBase`, die der Bereitstellung von Standardmethoden zur Änderung der SIB-Parameter dient und der `SIBProxy`, eine Art Platzhalter für Situation, in denen der SIB nicht unterstützt wird, sodass der Workflow dennoch erfolgreich geladen werden kann.

SIB Pattern. Von den Entwicklern des *jABC* wurde auch ein Entwurfsmuster für die SIBs entwickelt, damit eine klare Struktur der SIB-Nutzung vorhanden ist. Somit wird gewährleistet, dass Nutzer der SIBs ein gemeinsames Vokabular nutzen und gleiches Verständnis vorweisen. Es sollte z. B. darauf geachtet werden, dass die Klassen in keiner Abhängigkeit zum Prozesseditor stehen. Das Muster besteht aus Komponenten bzw. Schichten, welche in der folgenden Abbildung[Nag, S. 48] dargestellt werden:

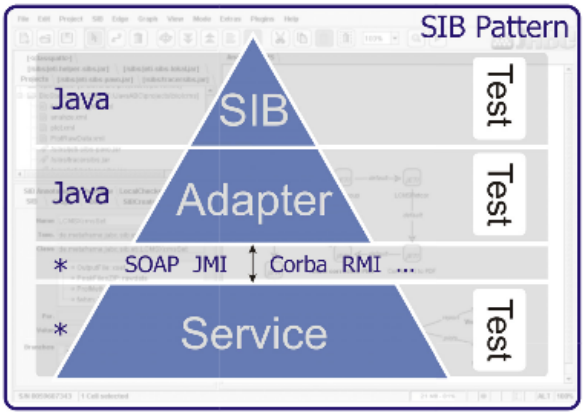


Abbildung 4.3: SIB-Adapter Entwurfsmuster

- *SIB*: SIB-Klassen leiten alle Aufrufe durch die Schnittstellen an die Adapterschicht weiter.
- *Adapter:* Verbirgt technische Details und steht für den Aufruf durch ein oder mehrere SIBs bereit. Hierbei kann es sich um beliebige Java-Adapter handeln, welche schließlich Aufrufe an die Service-Schicht weiterleiten.
- *Service:* Es handelt sich hierbei um plattformunabhängige Funktionalitäten einer Software. Jede beliebige Java- Adapterklasse kann einen solchen Service aufrufen.

Insgesamt wird sich damit eine Entkopplung der Service-Implementierung vom Komponentenmodell erreicht.

Erweiterbarkeit

Bei *jABC* ist das Konzept der Plug-ins, zur Erweiterung des Frameworks, vorhanden. Eine als Plug-in-Interface definierte Schnittstelle dient der Kommunikation zwischen den Plug-ins und den SIBs. Bevor das Plug-in Methodenaufrufe durchführen kann, muss sich das entsprechende Interface beim SIB-Container registrieren. Um Fehlern bei nicht vorhandenen Methoden seitens der SIBs, die jedoch aufgerufen werden, vorzubeugen, werden diese in den SIBs ausgefiltert. Das für die Zugiffssicherheit auf die SIBs verantwortliche Objekt ist der bereits erwähnte `ProxySIB`.
In *jABC* wird zwischen zwei Typen von Plug-ins unterschieden. Auf der einen Seite gibt es die Semantik-Plug-ins, welche eigene Schnittstellen haben und entsprechend auch eigene Semantiken. Auf der anderen Seite stehen die Feature-Plug-ins. Diese sind für die Erweiterung der Funktionalität des *jABC* gedacht. Sie können an allen Stellen im *jABC* eingesetzt werden. Dieses Mechanismus der Funktionalitätserweiterung bedienten sich auch die Entwickler der Projektgruppe. Sie nutzten diese Plug-in-Möglichkeit, um ihren Recommender in das *jABC* zu integrieren [Nag, S. 35f].

4.2 Grafische Benutzeroberfläche

Damit das von uns entworfene Empfehlungssystem wie gewünscht auch von Nicht-Informatikern einfach und intuitiv bedient werden kann, wurde von der Projektgruppe eine grafische Benutzeroberfläche (engl. *graphical user interface*, *GUI*) dafür entwickelt. Neben der Anzeige der empfohlenen Services, welcher als weiterer Inspektor im jABC Hauptfenster integriert wurde, ist ein Konfigurationsdialog wichtiger Bestandteil dieser GUI. Die Anzeige stellt dem User die vorgeschlagenen Services in einer Liste dar. Die Vorschläge werden, der Übersichtlichkeit wegen, in Gruppen von 7 Vorschlägen pro Seite dargestellt.
Der Konfigurationsdialog, welcher im folgenden Abschnitt näher erläutert wird, erlaubt insbesondere Auswahl sowie Festlegung spezifischer Parameter der Importmodule und Empfehlungsalgorithmen gebündelt zu *Profilen*. Der Dialog ist Teil des Recommender-Module für *jABC* und kann durch das Untermenü `Profil konfigurieren` aufgerufen werden.

4.2.1 Konfigurationsdialog

Um das von unserer Projektgruppe entworfene Empfehlungssystem effektiv in der *jABC*-Umgebung nutzen zu können, wurde mit Hilfe der Bibliotheken *DesignGridLayout* und *JFreeChart* eine grafische Oberfläche erstellt. Über diese sollte insbesondere die Konfiguration der verschiedenen Importmodule und Empfehlungsalgorithmen möglich sein.
Möglich ist im Dialog das Kopieren, Bearbeiten und Löschen von Profilen. Die Konfiguration eines Profils umfasst die Gewichtung der einzelnen Algorithmen und die Auswahl der zu nutzenden Importmodule. Zu diesem Zweck besteht der Dialog aus zwei Teilen, nämlich dem Profil-Panel und dem Tab-Panel. Während im Profil-Panel die angesprochenen Profile verwaltet werden können, dient das sogenannte Tab-Panel der eigentlichen Konfiguration der einzelnen Plug-ins.
Bei der Entwicklung des Konfigurationsdialoges wurde sichergestellt, dass der Nutzer um eine Bestätigung gebeten wird, wenn er beim Verlassen des Konfigurationsfensters seine letzte Änderungen am aktuellen Profil noch nicht gespeichert hat. Auf diese Weise soll die Benutzerfreundlichkeit der Oberfläche und letztendlich auch des Gesamtsystems so weit wie möglich optimiert werden.
Das Tab-Panel besteht aus einem Tab für die Importmodule und einem Tab für die Algorithmen. Ein einzelnes Tab ist wiederum aufgeteilt in eine Listen- und eine Detail-Ansicht. In der Listen-Ansicht werden die verfügbaren Module vom jeweiligen Typ (Importmodul oder Algorithmus) aufgelistet und in der Detailansicht kann man sich die spezifischen Einstellungen für ein ausgewähltes Modul anzeigen lassen und bearbeiten.

In der Listen-Ansicht können einzelne Module für das gewählte Profil auch komplett deaktiviert werden. Bei den Algorithmen kann zudem eine Gewichtung vorgenommen werden, die der Übersichtlichkeit halber durch ein Säulendiagramm dargestellt wird. Die Detail-Ansicht kann für jedes Plug-in unterschiedlich aussehen, um sich auf die entsprechenden Gegebenheiten und die vorhandenen Parameter einzustellen.

Profile

Es wird mit Profilen gearbeitet, da es einem Nutzer möglich sein sollte, mehrere Profile anlegen zu können. Des Weiteren werden dem User die Möglichkeiten des Speicherns und Ladens von unterschiedlichen Konfigurationen angeboten. Als Illustration für das Profil-Panel soll die Abbildung 4.4, welche das Neuanlegen eines Profils darstellt, dienen. Das Dropdown-Menü `Vorlage` ermöglicht, auf bereits angelegte Profile und deren Einstellungen aufzubauen.

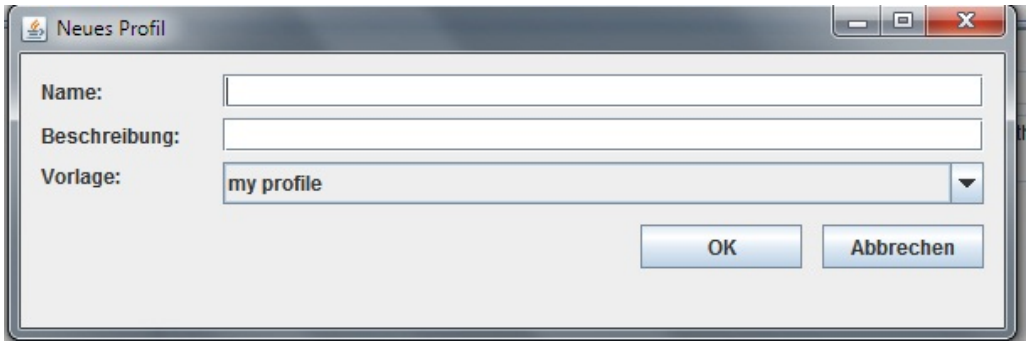


Abbildung 4.4: Neues Profil anlegen

Plug-ins

Der zweite Bestandteil des Konfigurationsdialoges ist das Tab-Panel, das zur Steuerung der Algorithmen und der Importmodule verwendet wird. Diese sind jeweils in einem Tab organisiert. Ein einzelnes Tab ist wiederum aufgeteilt in eine Listen- und eine Detailansicht.
In der Listenansicht werden die verfügbaren Plug-ins aufgelistet. Sie können für das gewählte Profil mit Hilfe der vorhandenen jeweiligen Checkboxen aktiviert oder deaktiviert werden. Für die Benutzerfreundlichkeit ist ein Button vorhanden, der die Ab- und Anwahl aller Importmodule bzw. Algorithmen durch einen Klick erlaubt.
In der Detailansicht werden die festen Eigenschaften – Name, Beschreibung und Typ – eines Algorithmus (Abbildung 4.5(a)) oder Importmoduls (Abbildung 4.5(b)) angezeigt. Außerdem wird hier die Konfiguration der veränderlichen Eigenschaften vorgenommen, sodass der Benutzer die Funktionsweise der Module an seine Bedürfnisse anpassen kann.
Bei der Umsetzung wurden die Detailansichten dynamisch in die GUI integriert, da sie in den jeweiligen Modulen selbst implementiert sind.
Zu den Einstellungen der Algorithmen zählt die Gewichtung. Das Einstellen der Gewichte kann auf zwei unterschiedliche Arten vorgenommen werden: Zum einen durch

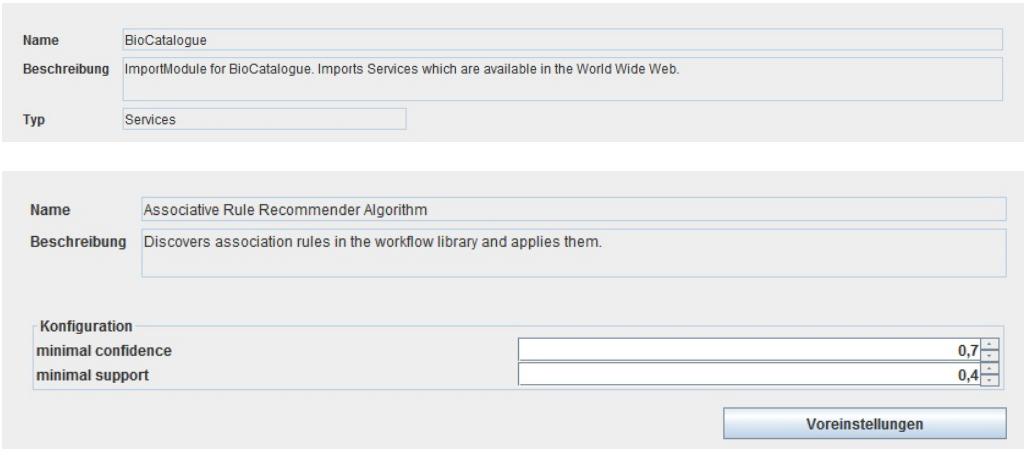


Abbildung 4.5: Detailansicht für Importmodule und Algorithmen

Eingabe konkreter Zahlen und zum anderen durch Spinner neben dem Eingabefeld (siehe Abbildung 4.6).

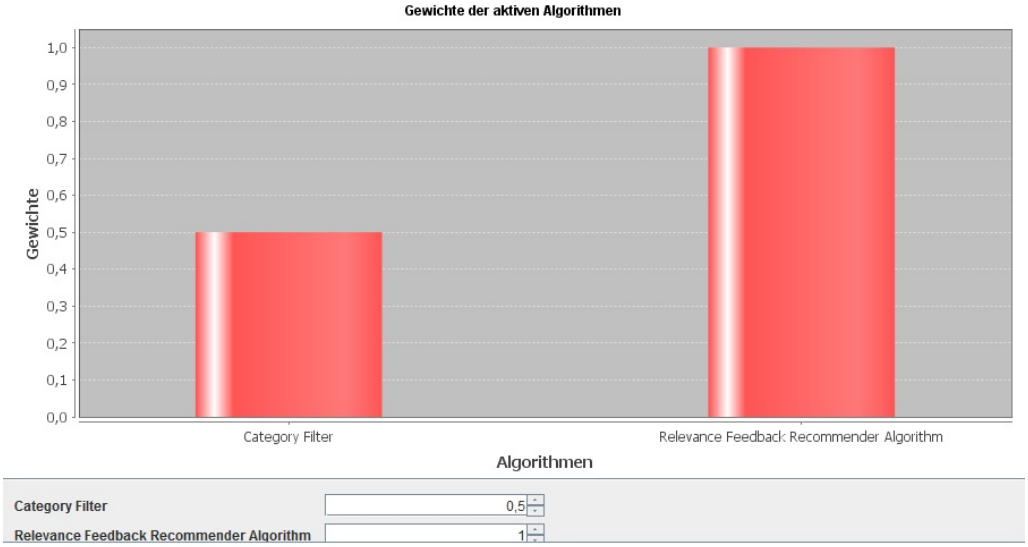


Abbildung 4.6: Detailansicht für die Algorithmengewichtungen

Im Folgenden werden einige Beispiele für GUI-Elemente und Dialoge vorgestellt.

Mutual Information Dieser Algorithmus hat die drei Parameter *maximum number of predecessors*, *maximum number of successors* und *distance factor*, die jeweils einzeln innerhalb der Detailansicht eingestellt werden können. Diese Einstellung erfolgt über Eingabe der Zahlwerte oder durch Klicks auf die Pfeilbuttons, wodurch die Werte in den erlaubten Grenzen vergrößert bzw. verkleinert werden (vgl. Abbildung 4.7).

Association Rule Miner Der Algorithmus „Association Rule Miner“ benötigt die Eingabe zweier Parameter *minimal confidence* und *minimal support*. Mithilfe

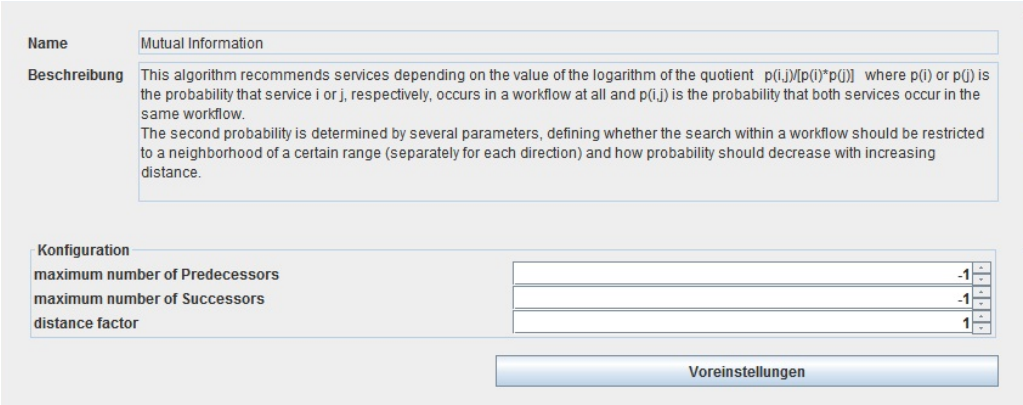


Abbildung 4.7: Detailansicht für den Algorithmus „Mutual Information“

dieser sucht er nach Assoziationsregeln. Die Einstellungsmöglichkeiten sind analog zu denen von „Mutual Information“ (vgl. Abbildung 4.8).

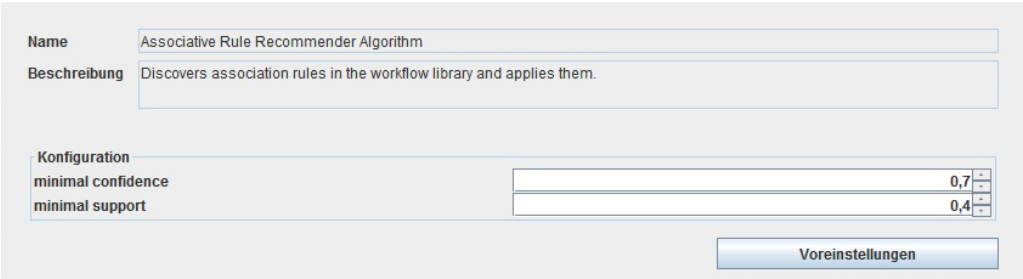


Abbildung 4.8: Detailansicht für den Algorithmus „Association Rule Miner“

Importmodule für EMBOSS Dieses Importmodul benötigt, im Gegensatz zu den oben erwähnten Algorithmen, ACD-Dateien damit es arbeiten kann. So wurde es seitens der Entwickler mit einem Dateiauswahldialog versehen. Dieser kann in der Detailansicht durch den Klick auf den Button „Suche“ geöffnet werden und der Nutzer kann das entsprechende Verzeichnis mit ACD-Dateien auswählen (vgl. Abbildungen 4.9 und **??**).

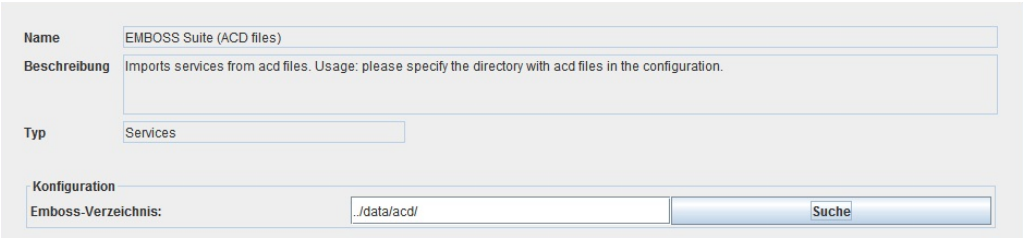


Abbildung 4.9: Detailansicht des Importmoduls für EMBOSS

Speicherdialog Wenn noch nicht gespeichert wurde, öffnet sich beim Schließen des Konfigurationsdialoges eine Nachrichtenbox, welche darauf hinweist, dass die Änderungen an der Konfiguration noch nicht gespeichert wurden. Nun hat der Benutzer die Möglichkeit zu speichern ohne den Dialog zu verlassen.

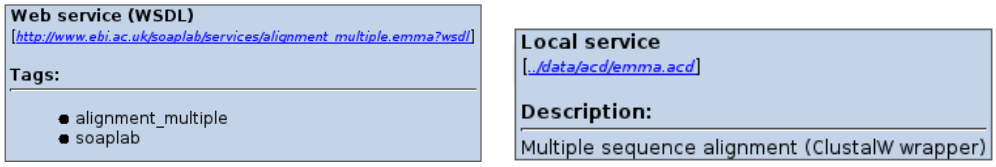
4.2.2 Inspektor

Das jABC-Plug-in registriert während der Initialisierungsphase die Klasse `RecommendationList`, welche die Schnittstelle `Inspector` implementiert und neben den Inspektoren anderer Plug-ins in dem dafür vorgesehenen Bereich eingebettet ist und über einen Klick auf das zugehörige Tab angezeigt werden kann.
Die Anzeige teilt sich vertikal in drei Bereiche auf: Titel, Empfehlungsliste und Seitennavigation (vgl. Abbildung 4.10). Im Titel enthalten ist die Gesamtzahl der empfohlenen Services. Die Navigationsleiste gibt die aktuelle sowie die maximale Seitenzahl an und erlaubt einen direkten Wechsel zur ersten und letzten Seite sowie zur unmittelbar vorhergehenden und nachfolgenden. Falls eine solche Möglichkeit sinnwidrig ist (etwa Navigation zum Anfang auf Seite eins), so sind entsprechende Elemente deaktiviert.
Empfehlungen werden in Abhängigkeit von den aktuell markierten SIBs im Workflow berechnet. Allerdings führt das Demarkieren jeglicher SIBs nicht zur Änderung der bisherigen Empfehlungen, um die alte Empfehlung weiterhin verwenden zu können und so die Nutzbarkeit des Plug-ins zu erhöhen.

Abbildung 4.10: Empfehlungsinspektor im Inspektorenbereich

Die Empfehlungen werden seitenweise mit maximal sieben Einträgen pro Seite in einer Liste angeordnet. Jeder Eintrag ist links mit einem Icon versehen, welches die Identifizierung des Servicetyps für den Benutzer erleichtern soll (Web-Services etwa sind durch einen Globus repräsentiert, lokale Services durch einen Monitor, Skripte durch ein Dateisymbol). Zusätzlich sind die Icons mit entsprechenden Schriftzügen versehen, sodass der Benutzer sogar über verschiedene Web-Service-Typen informiert wird (WSDL, SoapLab etc.).
Rechts neben dem Icon sind weitere verfügbare Informationen zum Service angegeben, insbesondere in Fettdruck und bei erhöhter Schriftgröße der Servicename.

Darunter befinden sich gegebenenfalls in schwarzer Kursivschrift Angaben zum Autor sowie die Namen einiger zugeordneter Tags. Eine weitere Zeile darunter wird in blauer Farbe der Anfang der Servicebeschreibung dargestellt. Alle Daten (auch Servicenamen) sind für die Darstellung auf eine Maximallänge begrenzt und werden im Falle der Überschreitung gekürzt und die Auslassung durch „. . . “ gekennzeichnet, sodass bei normaler Inspektorgröße die Notwendigkeit eines horizontalen Scrollings in der Liste vermieden wird.



(a) Tooltip mit Taginformationen (b) Tooltip mit Beschreibung

Abbildung 4.11: Tooltips mit Taginformationen bzw. Beschreibung

Tooltips. Jeder Listeneintrag ist zudem mit einem Tooltip versehen, welcher beim Bewegen der Maus über das Feld angezeigt wird. Im Tooltip wird ebenfalls der Typ angegeben, ergänzt durch eine URL zur Quelle. Außerdem findet sich dort eine Auflistung zugeordneter Tags, wobei maximal zehn angezeigt werden, und die Beschreibung (maximal 20 Zeilen mit höchstens 80 Zeichen pro Zeile), sofern vorhanden (siehe Abbildung 4.11).

Auswahl. Durch einen Doppelklick auf den entsprechenden Listeneintrag kann der zum Service gehörige SIB dem aktuellen Workflow hinzugefügt werden. Die Positionierung dieser SIB ist abhängig vom Kontext der Empfehlung: Ist lediglich ein Service markiert, so wird der neue SIB um einige Pixel rechts unter ihr eingefügt, andernfalls in der oberen linken Ecke des Editorbereiches.

4.3 Einbinden des Recommenders

In diesem Abschnitt werden verschiedene relevante Aspekte erörtert, die bei der Integration unseres Recommender-Systems in die Workflowumgebung von *jABC* zu beachten waren. Hierzu zählen zum Beispiel die Umwandlung der Services aus den von uns verwendeten Datenquellen in SIBs und die sogenannte Datenflussrekonstruktion. Mit OSGi wird außerdem eine Technologie angesprochen, welche nicht erfolgreich in die Endversion des Recommender-Systems eingebunden werden konnte, deren Verwendung aber über einen längeren Zeitraum angestrebt wurde.

4.3.1 SIB-Generierung

Wie bereits beschrieben, werden Services in jABC durch SIBs repräsentiert. Diese SIBs sind Java-Klassen mit der Annotation `SIBClass` und spezieller Struktur für mögliche Ein- und Ausgabeparameter.

SIB-Struktur. Zu jedem Service der Servicedatenbank des Recommender-Systems wird genau eine Klasse mit der entsprechenden Annotation erzeugt, um diesen Service als SIB zu repräsentieren. Dabei werden Informationen über den Service in der Struktur der erzeugten Klasse gespeichert. Der Name der Klasse entspricht dem Identifikator des Services, wobei Konventionen der Namensgebung für Klassen in Java berücksichtigt werden. Die Eingabeparameter werden als Attribute des Typs `ContextExpression` und Ausgabeparameter als Attribute des Typs `ContextKey` verwaltet, um die in Abschnitt 4.3.2 beschriebene Vorgehensweise der Datenflussrekonstruktion zu ermöglichen.

Die Beschreibung des Services wird als HTML-Dokument in der durch jABC aufgerufene Methode `getDocumentation()` mit Parameter `DocType.SIB` gespeichert. Dabei werden Sonderzeichen in HTML-Quelltext umgewandelt, um mögliche Quelltext-Injektionen bei der SIB-Generierung zu vermeiden. Auf diese Weise kann die Beschreibung des Services auf einheitliche Weise durch jABC bei der Navigation durch den SIB-Baum angezeigt werden.

Damit jede erzeugte Klasse später einem Service zugeordnet werden kann, wird der Identifikator des Services in der erzeugten Klasse gespeichert. Um die Datenflussrekonstruktion zu vereinfachen, erweitert jede erzeugte Klasse die Klasse `AbstractSIB`. Da genau die erzeugten Klassen `AbstractSIB` erweitern, ist deren Identifizierung durch die Instanz-Abfrage möglich. Zusätzlich bietet `AbstractSIB` mit der Methode `getServiceId()` die Möglichkeit während der Datenflussrekonstruktion die ursprünglichen Services der Servicedatenbank zu referenzieren.

Kompilation. Die Kompilierung der Klassen geschieht mit Hilfe des *Java Development Kits*[1] und wird bei jeder Aktualisierung der Servicedatenbank ausgeführt. Dies macht eine Installation des Java Development Kits für die Ausführung des Recommender-Systems notwendig.
Es wurde versucht die Abhängigkeit zum Java Development Kit zu umgehen, indem die *Byte Code Engineering Library*[2] zur Klassenerzeugung eingesetzt wurde. Diese sollte zwar in der Lage sein SIBs zu erzeugen, jedoch aufgrund der hohen Komplexität sowie der unzureichenden Dokumentation dieser Bibliothek führte ihr Einsatz zu keinem Erfolg.

Registrierung. Die kompilierten Klassen werden in einem Java-Archiv (*JAR*) gebündelt und automatisch im aktuellen Projekt als benutzbare SIBs gemäß ihrem Präfix `de.jabc.sib.pgsassd` in der Baumstruktur registriert.

4.3.2 Datenflussrekonstruktion

Dieser Abschnitt beschreibt die Konvertierung von Kontrollflussgraphen, wie sie durch die Worflowumgebung jABC gegebenen sind, in Datenflussgraphen, dem in den genutzten Bioinformatik-Plattformen eingesetzten Modellierungskonzept für Workflows. Die Konvertierung erlaubt es, Workflowinformationen den Algorithmen zur Verfügung zu stellen, ohne eine parallel bestehende Datenstruktur verwalten zu müssen.

Kontrollflussgraphen

Ein Kontrollflussgraph $G = (V, E, s)$ besteht aus einer Knotenmenge V, einer Menge E gerichteter Kanten und einem Startknoten $s \in V$, wobei jeder Knoten $t \in V$ von s aus erreichbar sein muss.
Semantisch (im Kontext von Workflows) beschreibt ein Kontrollflussgraph die Ausführungsreihenfolge von Services (Knoten des Kontrollflussgraphen). Bei der Ausführung können Daten zwischen Services über globale Kontext-Variablen übermittelt werden, die ein Kontrollflussgraph per se nicht modelliert.

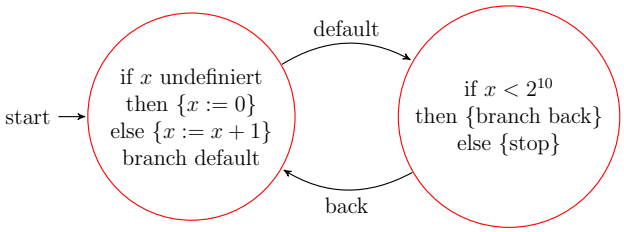
Datenflussgraphen

Ein Datenflussgraph $G' = (V', E')$ ist ein beliebiger gerichteter Graph, dessen Kanten semantisch der Datenabhängigkeit zwischen zwei Services entsprechen. Bei der Ausführung eines Datenflussgraphen kann ein Service ausgeführt werden, sobald alle zu seiner Ausführung benötigten Daten bereitstehen.

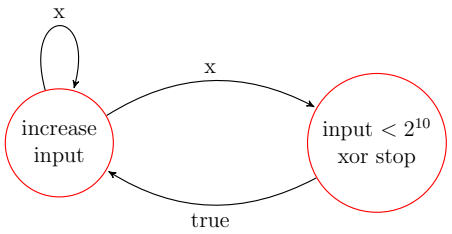
Konvertierungsprobleme

[1] `http://www.oracle.com/technetwork/java/javase/downloads/index.html`
[2] `http://commons.apache.org/proper/commons-bcel/`

Kanten im Datenflussgraphen entsprechen *nicht* den Kanten im Kontrollflussgraphen, sondern sind von Zugriffen auf Kontext-Variablen abhängig. Im Allgemeinen lässt sich ein Kontrollflussgraph nicht direkt in einen Datenflussgraph übersetzen, wie das in Abbildung 4.12 dargestellte Beispiel demonstriert. Dabei soll ein Service die Variable x initialisieren bzw. inkrementieren und ein anderer Service eine Eigenschaft dieser Variable auswerten und gegebenenfalls die Ausführung stoppen. Eine naive Beschreibung des Kontrollflussgraphen als Datenflussgraph, indem lediglich Datenabhängigkeiten beschrieben werden, führt zu einem Problem. Während der Kontrollflussgraph eine klare Ausführungssemantik besitzt, ist die bei der naiven Übersetzung entstehende zirkuläre Datenabhängigkeit im Datenflussgraph nicht erlaubt. Kein Service innerhalb eines Kreises kann ausgeführt werden, bevor sein Vorgänger Daten bereitgestellt hat.



(a) Kontrollflussgraph



(b) Naive Beschreibung des Kontrollflussgraphen als Datenflussgraph

Abbildung 4.12: Beispiel für mögliche Probleme bei der Konvertierung von Kontrollfluss- zu Datenflussgraphen

Konvertierung

Im Folgenden wird die Übersetzung eines Kontrollflussgraphen (gegeben als Instanz der Klasse `SIBGraphModel`) in einen Datenflussgraphen (Instanz der Klasse `Workflow`) beschrieben. Diese Konvertierung ist notwendig, damit alle Algorithmen, die Workflows gemäß dem Metamodell als Datenflussgraphen auffassen, genügend Informationen über den aktuellen Workflow bekommen. Einen weiteren Zweck besitzt die Konvertierung nicht.
Die Knoten eines `SIBGraphModel`s sind Instanzen der Klasse `SIB`. Der Datenfluss zwischen einzelnen SIBs definiert sich über die Zuweisung entsprechender Parameter der SIBs. Die Typen der relevanten SIB-Parameter sind `ContextExpression` und

`ContextKey`. Ein Parameter vom Typ `ContextKey` definiert eine Kontext-Variable, auf die schreibend zugegriffen werden kann. Ein Parameter vom Typ `ContextExpression` definiert einen Ausdruck, der Kontext-Variablen und Konstanten verwenden kann, auf die lesend zugegriffen wird. Die Information dieser Parameter wird verwendet, um den Datenfluss des Kontrollflussgraphen zu *approximieren*.
SIBs können in zwei Kategorien unterteilt werden:

1. SIBs, die Services aus der Servicedatenbank entsprechen;
2. andere SIBs.

In die erste Kategorie fallen alle Services, die vom SIB-Generator erstellt wurden und als Instanzen der Klasse `AbstractSIB` eine vordefinierte Struktur besitzen. So besitzt die SIB für jeden Eingabeparameter vom Typ `type` und Namen `name` des entsprechenden Services in der Servicedatenbank einen Parameter mit dem Namen `in_type_name` und dem Typ `ContextExpression`. Für jeden Ausgabeparameter vom Typ `type` und Namen `name` des entsprechenden Services in der Servicedatenbank besitzt die SIB einen Parameter mit dem Namen `out_type_name` und dem Typ `ContextKey`.
Die SIBs der ersten Kategorie werden unmittelbar durch ihre Entsprechung in der Servicedatenbank als Knoten der `Workflow`-Instanz repräsentiert.
SIBs der zweiten Kategorie werden lediglich auf einen gleich benannten Knoten abgebildet, ohne eine Entsprechung in der Servicedatenbank zu haben.
Die Approximation des Datenflusses findet durch die Verbindung derjenigen Servicepaare (s_1, s_2) statt, für die gilt:

- s_1 hat einen Parameter vom Typen `ContextKey` mit dem Wert x;
- s_2 hat einen Parameter vom Typen `ContextExpression` mit dem Wert $\${\alpha}$;
- α ist ein arithmetischer Ausdruck mit mindestens einem Vorkommen der Variable x.

Falls der Wert eines Parameters des Typen `ContextExpression` leer ist oder eine Variable beinhaltet, die keinem Wert eines Parameters des Typen `ContextKey` entspricht, so gilt der entsprechende Input des Services als unbelegt, anderenfalls (insbesondere falls der Ausdruck nur aus Konstanten besteht) gilt der entsprechende Input als belegt.
Da in der Workflowumgebung jABC vordefinierte SIBs die oben beschriebene Parametertypstruktur aufweisen, werden im auf diese Weise erzeugten Workflow sinngemäß Kanten zwischen Services aus der Servicedatenbank und anderen (unbekannten) Services erstellt.
Die beschriebene Methodik liefert nur eine Approximation des Datenflusses, da z. B. zwei aufeinanderfolgende Services, die in dieselbe Kontext-Variable schreiben, beide als Quellen für Services gelten, die später diese Kontext-Variable lesen. Der schlimmste Fall ist eine Schleife mit n Services und nur einer Kontext-Variablen x, wobei jeder Service x liest und schreibt. Die beschriebene Methode liefert in diesem Fall einen vollständigen Graphen mit n Knoten.

4.3.3 OSGi

In diesem Abschnitt wird die *OSGi*-Technologie beschrieben, deren Verwendung für unser Projekt im Laufe der Entwicklungsphase vorgesehen war. Außerdem wird erläutert, warum dieses Vorhaben nicht abschließend umgesetzt werden konnte.

Grundlagen

Für die Erweiterung der *Eclipse*-Plattform steht Entwicklern der Mechanismus der Plug-in-Entwicklung zur Verfügung. Diese *Eclipse*-Plug-ins basieren auf *OSGi*-Bundles, welche die Bausteine der Komponentenarchitektur bilden.

OSGi stand früher für *Open Service Gateway Initiative*, welche von der *OSGi-Alliance* [Osga], einem Industriekonsortium, spezifiziert wurde. Dieses wurde 1999 gegründet und besteht aus Unternehmen verschiedener Branchen, wie z. B. der *Eclipse Foundation*, *Nokia*, *Siemens*.
OSGi ist hardwareunabhängig und stellt eine dynamische Softwarebasis dar. Es bietet die Möglichkeit der Etablierung und Verwaltung von Komponenten und Modulen bzw. Bundles in Java Applikationen. Die dynamische Ausrichtung ermöglicht, dass die Bundles und Services zur Laufzeit ausgetauscht, gelöscht oder neu hinzugefügt werden können. Auch eine Suchfunktion für die einzelnen Komponenten, welche im weiteren Verlauf dieses Abschnitts nähere Erläuterung findet, ist in die Plattform integriert. Jedoch sollte erwähnt werden, dass durch *OSGi* nur die API spezifiziert und eine Referenzimplementierung zur Verfügung gestellt wird. Somit ist es alleine nicht für den Produktiveinsatz gedacht, sondern stellt eine Vorlage für andere Software dar. Eine bekannte Umsetzung von *OSGi* ist z. B. *Equinox* [Equ]. Dabei handelt es sich um eine quelloffene Software, welche ein auf Java basierendes Framework bereitstellt.
Die Basis von *OSGi* bildet das *OSGi*-Framework, welches eine *Service Delivery Plattform* auf Basis von Java darstellt. Es handelt sich dabei um einen Container bzw. eine Laufzeitumgebung für die Bundles und Services. Es ist als Schichtenmodell spezifiziert, wobei alle Schichten aufeinander aufbauen. Mithilfe der Schichten werden die Funktionalität in Java und der entsprechenden *Java Virtual Machine* erweitert und die möglichen Abhängigkeitsproblematiken aufgelöst. Abbildung 4.13 [Osgb] gibt die die einzelnen Schichten und ihr Verhältnis untereinander wieder.
Bei dem Schichtenmodell stehen die Schichten *Execution Environment*, *Modules*, *Life Cycle*, *Services* und die optionale Schicht *Security* im Fokus. Im Folgenden werden die Schichten exemplarisch erläutert.

- **Execution Environment:** Das Konzept von *OSGi* soll die Ausführbarkeit und Kompatibilität auf verschiedenen Java Plattformen sein. Dies wird erreicht, indem nur die Definition eines Repräsentanten, anstatt einer konkreten Version der Laufzeitumgebung stattfindet. Dieser Repräsentant beschreibt lediglich, welche Klassen, Schnittstellen und Methoden vorhanden sein müssen.

- **Modules:** *Modules* bildet die unterste logische Schicht in *OSGi*. Hier werden die Komponenten (Bundles) definiert. Diese bestehen aus Java-Klassen und benötigen Ressourcen. Somit haben Bundles die Möglichkeit, dass sie einzeln installiert und genutzt werden können. Auch eine Datei *MANIFEST.mf* ist vorhanden, damit Metadaten, für mögliche Abhängigkeiten zwischen den Bundles, eingepflegt werden können. Jedoch müssen für die Sichtbarkeit der Bundles untereinander, die jeweiligen Klassen in die Datei exportiert werden.

- **Life Cycle:** Die in *Modules* definierten Bundles können verschiedene Zustände haben, welche Abbildung 4.14 [Osgc] illustriert.

  Die einzelnen Zustände sind in der folgenden Tabelle zusammengefasst:

Zustand	**Erklärung**
starting	Start des Bundles
active	Bundle ist gestartet und läuft
stopping	Bundle wird gestoppt
installed	Bundle ist installiert
resolved	Alle benötigten Java Klassen verfügbar. Bundle ist fertig zum Starten oder wurde gerade beendet
uninstalled	Bundle deinstalliert und kann nicht mehr in einen anderen Status und genutzt werden

  Ein möglicher Ablauf eines Bundle-Lebenszyklus ist, dass das Bundle installiert wird und daraufhin mögliche Abhängigkeiten aufgelöst werden. Somit kann das Bundle starten und geht in den Zustand *active* über. Wenn es fertig ist, geht es in den Zustand *stop* und daraufhin in *resolved*, wo es entweder

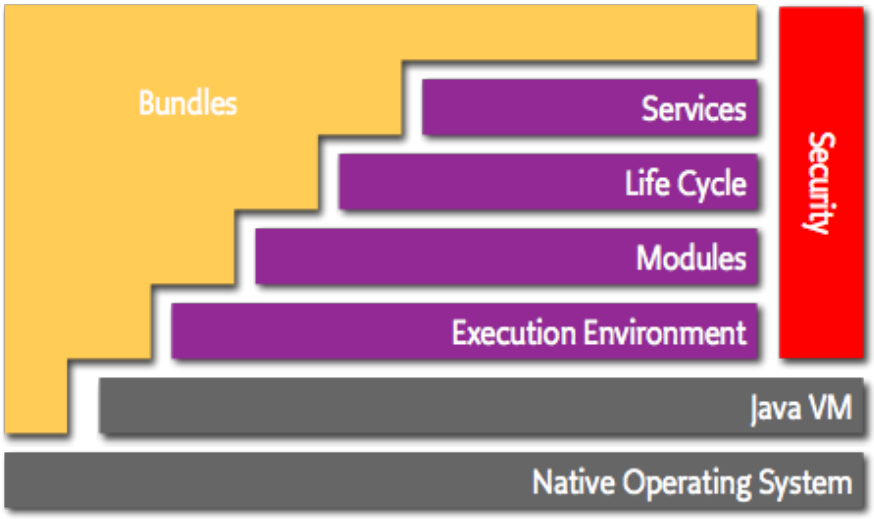


Abbildung 4.13: OSGi-Architektur Schichtenmodell

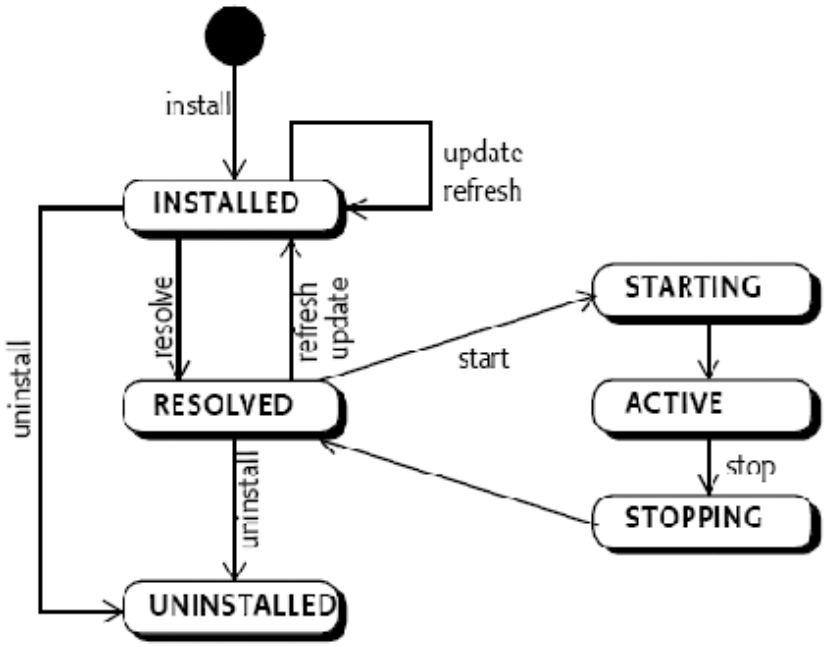


Abbildung 4.14: Lebenszyklus eines Bundles

deinstalliert oder erneut gestartet werden kann. Die einzelnen Zustände werden von einem Agenten (`BundleContext`) gesteuert. Dieser ist auch von außen steuerbar, da er eine Schnittstelle zum Framework implementiert [Fur12].

- **Services:** Diese Schicht kann in zwei Unterpunkte unterteilt werden. Zum einen die Standardservices und zum anderen die *Registry*. Bei den Standardservices finden sich vordefinierte Lösungen für verschiedene Bereiche, welche direkt von den Bundles implementiert werden können. Hier kann als Beispiel der HTTP-Service erwähnt werden. Die Registry hingegen stellt eine Art Zentrale dar. Die Bundles benutzen diese, um Services anzumelden, sodass andere Bundles diese nutzen können. Hierbei wird zentral ein Service-Java-Objekt in einem Register angemeldet, welches nur zur Laufzeit besteht, wodurch sogar zur Laufzeit, aufgrund von den möglichen An- und Abmeldungen der Bundles, die Persistenz nicht gewährleistet werden kann. Um mit den angemeldeten Services arbeiten zu können, sind folgende drei Mechanismen vorhanden:

  Service Listener dienen der Verfolgung von Änderungen;

  Service Tracker dient, basierend auf dem *Service Listener*, der Spezifikation der Services über den Namen der Schnittstelle.

  Declarative Service sind für die Beschreibung der Services in XML.

  Wie schon bereits bei der Schicht *Modules* angedeutet wurde, können zwischen den Bundles Abhängigkeiten vorhanden sein. Diese sind teilweise schon durch den Zweck des Bundle-Mechanismus bedingt. Wenn ein Plug-in zu groß wird,

wird es in verschiedene Komponenten aufgeteilt. Somit entstehen Abhängigkeiten zwischen den Bundles. Es entsteht die Notwendigkeit, dass der Zugriff geregelt werden muss, damit Ressourcen aufeinander zugreifen können. Dieser wird durch das Im- und Exportieren von Bundles mit Hilfe des Java-Class-Loading-Mechanismus erreicht. Die Funktionsweise wird hier abstrahiert und stichpunktartig angeführt.

1. das Exportieren findet auf Ebene der Pakete statt;
2. *MANIFEST.MF*: Hinzufügen einer Anweisung `Export-Package: org.test.myPackage`;
3. Andere Plug-ins müssen `myPackage` entsprechend importieren.

Bei Exporten sind Versionen und Versionsbereiche möglich, womit noch feingranularer spezifiziert werden kann. Wenn z. B. folgender Code

```
Import-Package: myPackage1; version="[1.0.0,1.5.0]"
```

in die Datei *MANIFEST.mf* eingefügt wird, bedeutet dies, dass das Paket `myPackage1` nur in der Version größer oder gleich 1.0.0 und kleiner als 1.5.0 verwendet werden kann.

Entscheidung der Projektgruppe gegen OSGi

Zusammenfassend lässt sich sagen, dass *OSGi* zur modularen Gestaltung der Plug-ins gut gegeignet ist. Jedoch gab es in der Projektgruppe Probleme mit den Abhängigkeiten, so dass die Software nicht innerhalb von *jABC* auf einem OSGi-Server starten konnte. Die Ursachenforschung führte zu neuen Ergebnissen und Umstrukturierungen, die im folgenden Abschnitt dargestellt werden.
Da das Recommender-System der Projektgruppe ursprünglich als *Eclipse*-Plug-in konzipiert wurde, kam die Verwendung von *OSGi* für den Recommender als *jABC*-Plug-in in Frage, denn da *Eclipse* ebenfalls *OSGi* nutzt, sind alle *Eclipse*-Plug-ins auch *OSGi*-Bundles. Nun stellte sich das Problem, dass das vom Datenbank-Modul verwendete Modelling Framework *EMF* mit *OSGi* (ohne *Eclipse*) inkompatibel ist. Es wurde schnell erkannt, das eine Abhängigkeit an *Eclipse* im *OSGi-BundleActivator* existiert. Eine letzte Möglichkeit wäre gewesen, das *Eclipse Equinox OSGi*-Framework zu verwenden, da hier die *Eclipse*-Abhängigkeiten vorhanden sind und *EMF* dann funktionieren sollte. Jedoch erwiesen sich jegliche Bemühungen der Projektgruppe als vergeblich und es war nicht möglich, das Vorhaben, das *Equinox*-Framework ohne *Eclipse* zu starten und nur unsere Bundles zu installieren, umzusetzen.
Die Projektgruppe stand vor der Entscheidung, entweder das Modell zu verändern und *OSGi* zu behalten oder aber *OSGi* durch einen anderen Mechanismus zu ersetzen. Durch die *EMF*-Dokumentation war bekannt, dass *EMF* auch außerhalb von *Eclipse* genutzt werden kann. Da die Abhängigkeit zu *Eclipse* nur im *OSGi-BundleActivator* existiert, ist eine Konfiguration komplett ohne *OSGi* möglich. Da das Datenmodell sehr umfangreich ist und fast alle anderen Module darauf aufbauen, würde das Ersetzen von *EMF* umfangreiche Änderungen mit sich bringen. Daher wurde entschieden, eine Alternative für *OSGi* zu suchen.

Da jedes *OSGi*-Bundle auch wieder nur eine JAR-Datei ist, welche um Metadaten angereichert wurde, werden diese Metadaten als nicht vorhanden gesehen und die JARs als normale JARs verwendet. Somit wird deren Inhalt (Klassen) zum Classpath hinzugefügt und das Plug-in dann manuell gestartet. Da jedoch der Recommender bisher darauf ausgelegt ist, dass die Algorithmen und Import-Module als Plug-ins vorhanden sind und die Projektgruppe diese Gegebenheit beibehalten wollte, bediente sie sich der Standardmechanismen aus Java. Mit dem `ServiceLoader` enthält Java ebenfalls einen Mechanismus, der für das Laden von Plug-ins genutzt werden kann. Dazu werden Services (hier: Algorithmen und Module) durch Dateien in *plugin.jar/META-INF/services/** exportiert. Diese tragen den Namen desjenigen Interfaces, dass sie implementieren. Im Recommender-Kern wird dann der `ServiceLoader` aufgerufen und bekommt das Interface übergeben. Dadurch werden alle Klassen, die dieses Interface implementieren, solange sie auf dem Classpath liegen, gefunden. Im Gegensatz zu *OSGi*-Bundles fehlt jetzt hier die eingeschränkte Sichtbarkeit, was jedoch keine weiteren Probleme bereitet.
Durch die mehrfache Umstrukturierung des Lademechanismus für die Plug-ins gibt es nun Lademechanismen für *Eclipse*-Plugins, für *OSGi*-Bundles und für den *Java ServiceLoader*. Da das *EMF*-Projekt aus Gründen der Rückwärtskompatibilität das ursächliche Problem nicht zu lösen vermag, ist es unwahrscheinlich, dass der Lademechanismus für *OSGi*-Plugins außerhalb von Eclipse genutzt werden kann.
Das Ziel der Projektgruppe, den Recommender in einer Plugin-Architektur umzusetzen, konnte daher zumindest teilweise erfüllt werden.

5 Evaluation

Nach Abschluss der Implementierung sollte das Ergebnis dieser gewissenhaft evaluiert werden, so dass sichergestellt werden kann, dass die Software ihren Ansprüchen in Bereichen wie Korrektheit und Qualität tatsächlich entsprechend gerecht wird. Der Ablauf dieser Evaluation soll in diesem Kapitel näher beschrieben werden.

5.1 Evaluation

Wie bei jedem größeren Software-Projekt stellen das Testen und die Evaluation des erzeugten Programmcodes wichtige Bestandteile des Entwicklungsvorgangs dar. Hierbei gilt es, gleich mehrere Aspekte zu beachten. Zunächst muss überprüft werden, ob der Import der diversen Service-, Workflow- und Ontologie-Informationen von den unterschiedlichen Portalen einwandfrei verläuft. Weiterhin sollten die einzelnen Algorithmen des Recommenders durch entsprechende Tests auf ihre Korrektheit überprüft werden. Schließlich stellt sich die Frage nach der semantischen Qualität und Sinnhaftigkeit des Recommenders und seiner Empfehlungen im Hinblick auf den Praxisbetrieb. Die Evaluation im Bezug auf diese einzelnen Bereiche soll in den folgenden Unterabschnitten jeweils kurz dargelegt und beschrieben werden.

5.1.1 Datenbestand

Um einen Eindruck von der Praxistauglichkeit des Recommender-Systems zu erhalten, ist – noch vor der Bewertung der Nützlichkeit der verschiedenen Algorithmen – die Inspektion des Datenbankinhaltes unumgänglich. Nachfolgend sind einige Kenndaten aufgeführt, welche dem Leser einen ersten Eindruck vom Datenbestand nach Verwendung der in der Projektgruppe implementierten Importmodule vermitteln

sollen. Ausgenommen davon ist allerdings das Importmodul für die Quelle *BioPortal*, da dieses fehlerhaft ist und keine Informationen beiträgt.[1]
Wann immer im Folgenden von *zur Verfügung stehenden* oder *spezifizierten* Daten die Rede ist, ist ein Attributwert gemeint, der ungleich `null` und auch ungleich dem leeren Wort – im Falle des Datentyps `String` – ist. Genauer wurden, aufgrund des großen Datenumfanges, nur einige Teile der Datenbank manuell auf ihre Sinnhaftigkeit überprüft. Diese Stichproben haben im Laufe des Evaluationsprozesses mehrfach auf Probleme, die zwischenzeitlich behoben wurden, aufmerksam gemacht.

Service-Bibliothek

Für den Import von Service- und Kategoriedaten wurde auf das Importmodul für die *EMBOSS*-Programmbibliothek wie auch auf das Importmodul für das Internetportal *BioCatalogue* zurückgegriffen.

Services. Insgesamt umfasst die Datenbank 5.109 Service-Beschreibungen. Davon entstammen 316 dem *EMBOSS*-Paket, welche allesamt mit textuellen Beschreibungen versehen sind. Weitere 2.181 Einträge wurden aus *BioCatalogue* importiert, jedoch verfügen 403 davon nicht über Beschreibungen.

Parameter. Alle nicht-internen Services besitzen zusammen 4.783 Parameter. Das sind im Durchschnitt ungefähr zwei (⌀ 1,92), im Maximum 25. Dieser Mittelwert erscheint zunächst ungewöhnlich, resultiert jedoch daher, dass etliche Services lediglich einen ein- oder ausgehenden Parameter besitzen (Konstanten leiten fixe Ausgaben an Nachfolger-Services weiter; Variablen nehmen Zwischen- oder Endergebnisse auf). Das Verhältnis zwischen Ein- und Ausgabeparametern ist mit 2.588 zu 2.195 relativ ausgeglichen.
Der weitaus häufigsten Parametertyp ist eine Zeichenkette (2.132). Das Recommender-System unterscheidet dabei (aufgrund einer fehlenden Unifizierung) zwischen den Typen `string` und `xsd:string`, die jeweils 1.154 bzw. 978 Mal vorkommen. Die nächsthäufigeren Typen können Tabelle 5.1 entnommen werden.

Beziehungen. In der Bibliothek sind 795 Verknüpfungen von Services zu „Beziehungen“ (repräsentiert durch die Klasse `Relation`) gespeichert. Sie alle stammen aus dem Importmodul für das *EMBOSS*-Paket (was einem Mittelwert von 2,51 `Relation`-Instanzen pro Service entspricht).
Dabei sind 424 Beziehungen vom Typen `topic`, die restlichen 371 vom Typen `operation`. Die fünf wichtigsten Beziehungsidentifikatoren sind mit der Anzahl ihrer Vorkommen in Tabelle 5.2 aufgeführt.

Nutzerstatistiken. Die Informationen stammen lediglich aus dem *BioCatalogue*-Importmodul. Die dort ermittelten Angaben beschränken sich auf die Anzahl der registrierten Nutzer, welche eine bestimmte Service-Beschreibung betrachtet haben.
Im Mittel ergibt sich dabei ein Wert von ca. 520 Aufrufen, bei einem Minimalwert von 16 und einem Maximalwert von 8.942 Aufrufen.

[1] Dieser Umstand ist nicht zu unterschätzen, denn dieses Modul ist im Wesentlichen das einzige, welches Ontologie-Informationen bereitstellen kann. In sehr beschränktem Rahmen leistet dies sonst nur das Importmodul für die EMBOSS-Bibliothek (vgl. 3.3.1).

Tabelle 5.1: Häufigste Parametertypen

Typname	Vorkommen
string	1154
xsd :string	978
outfile	176
boolean	152
seqall	109
sequence	57
retries	53
timeout	53
seqoutall	48
infile	46
integer	41
soapenc :string	40
run	37

Tabelle 5.2: Häufigste Typen für Beziehungen

ID	Name	Vorkommen
EDAM:0000091	Data handling	82
EDAM:0000090	Data retrieval	56
EDAM:0000220	Data file processing	31
EDAM:0000157	Sequence composition	28
EDAM:0000516	Data retrieval (database annotation)	25

Tags. Auch die in der Datenbank registrierten Tags stammen allesamt aus dem Portal *BioCatalogue*. Jeder Service ist im Durchschnitt mit 1,23 Tags annotiert (im Maximum 24). Insgesamt werden die 885 verschiedenen Tags damit 6.282 Mal genutzt.

Kategorien. Ein Teil der 247 Kategorien wird vom *EMBOSS*-Importmodul übernommen. Dabei handelt es sich um 76 Kategorien, die in zwei Ebenen organisiert sind – 20 davon bilden die oberste Ebene. Weitere 53 Top-Level-Kategorien stammen von der *BioCatalogue*-Plattform.

Workflow-Bibliothek

Der Datenbestand der Workflow-Bibliothek ist aus dem Portal *MyExperiment* importiert worden. Die Bibliothek umfasst insgesamt 1.483 Workflows, von denen alle mit Namen, Quelle und Autorenangaben versehen sind. Der Großteil dieser Arbeitsabläufe ist im *T2Flow*-Format spezifiziert (921), die zweitmeisten im XML-Format *XScufl* (338).

Service-IDs. In der gesamten Workflow-Bibliothek sind 14.345 Service-IDs verzeichnet. Damit umfasst im Mittel jeder Workflow zwischen 9 und 10 Services (⌀ 9,67). Der größte Workflow beinhaltet 243 Services.
Allerdings gibt sind nur rund die Hälfte dieser Service-IDs (7.038) mit unterschiedlichen Namen versehen. Die Service-IDs, welche am häufigsten mehrfach in Workflows auftreten, sind in Tabelle 5.3 angegeben.

Links. Erwartungsgemäß ist die Zahl der Links deutlich höher (häufig sind zwei Services gleich über mehrere Ports miteinander verbunden); sie beläuft sich in der Bibliothek auf 18.114 (das entspricht etwa drei Links pro Service). Im Durchschnitt

Tabelle 5.3: Häufigste Service-IDs in Workflows

ID	Vorkommen
Split_string_into_string_list_by_regular_expression	105
Concatenate_two_strings	93
Beanshell	70
Merge_String_List_to_a_String	70
regex_value	62
Read_Text_File	55
Get_Web_Page_from_URL	50
Flatten_List	46
checkStatus	41
Get_Image_From_URL	35
Job_params	33
regex	33
Input_data	31
Write_Text_File	31

weist ein einzelner Workflow 12 bis 13 Links auf (⌀ 12,8). Das Maximum liegt hier bei 596 Verknüpfungen.

Meta-Informationen. Allein das Importmodul für *MyExperiment* versteht drei verschiedene Formate für Workflow-Beschreibungen. In solchen Beschreibungen wiederum können Services diverser Art referenziert werden, etwa WSDL-Schnittstellen oder lokal definierte (Programmiersprachen-)Skripte.
Die häufigsten Typen können Tabelle 5.4 entnommen werden.
In Tabelle 5.5 ist der absolute und relative Anteil der Services, bei denen die Attribute Servicetyp und -quelle und weitere spezifiziert sind, wiedergegeben.

Association Rule Mining im Kontext gegebener Datenquellen

Im Abschnitt 3.4.1 ist ein Recommender auf Basis des *Association Rule Mining* dargestellt. Um eine sinnvolle Funktionalität dieses Recommenders zu gewährleisten, müssen die Datenquellen (Servicedatenbank siehe Abschnitt 2.2.1 und Workflowdatenbank siehe Abschnitt 2.2.2) einerseits groß genug sein, um eine gewisse Aussagekraft zu garantieren, und andererseits in Bezug zueinander stehen, d.h. in Workflows der Workflowdatenbank müssen Services der Servicedatenbank vorkommen. Im Kontext gegebener Datenquellen sind beide Anforderungen kaum erfüllt.
Die Workflowdatenbank enthält ca. 1500 Workflows mit durchschnittlich 9 bis 10 Services pro Workflow. Da die Anzahl unterschiedlicher Services in Workflows der gesamten Workflowdatenbank ca. 14000 ist, sind die Workflows untereinander häufig disjunkt. Die Disjunktheit der Workflows spricht für eine nicht ausreichende Größe der Datenbasis und macht Methoden des *Association Rule Mining* schwierig. Um dieses Problem in den Griff zu bekommen, müssen die Parameter des *Association Rule Mining Recommender* sensibel genug eingestellt werden. Darunter leidet unmittelbar die Qualität der Empfehlung, da die Sensibilität der Parameter die Entdeckung irrelevanter Zusammenhänge begünstigt.
Darüber hinaus ergibt sich eine weitere Problematik. Die Mehrheit der Services in Workflows der Workflowdatenbank sind nicht in der Servicedatenbank enthalten, da sie entweder Skripte oder Konstanten in den jeweiligen Workflows sind. Wenn diese Services nicht für die Analyse verwendet werden würden, dann wäre die Datenbasis zu klein, um Methoden des *Association Rule Mining* anzuwenden. Wenn diese Services hingegen in der Analyse berücksichtigt werden, dann werden größtenteils nur Zusammenhänge unter diesen Services aufgedeckt. Um mit diesem Problem fertig zu

Tabelle 5.4: Häufigste Servicetypen in Workflows

Typ	Vorkommen
Skript	3676
Konstante	2676
WSDL	1722
Lokal	1142
SOAPLab	105

Tabelle 5.5: Anteil spezifizierter Attribute der Meta-Informationen für Services

Attribut	abs. Anteil	rel. Anteil
Typ	9.321	65%
Quelle	13.153	92%
Operation	8.084	56%
Extra	1.105	8%

werden, werden solche Services in die Servicedatenbank aufgenommen und können deshalb auch empfohlen werden.

5.1.2 Korrektheitstests

Die Entwickler der einzelnen Algorithmen, Parser und Importmodule haben die korrekte Funktionsweise ihrer jeweiligen Methoden mit Hilfe von diversen Tests überprüft, für die teilweise eigene Klassen geschrieben wurden und die auch während der Entwicklung zum Zwecke von Fehlersuche und Laufzeitanalyse verwendet wurden. Bei diesen Tests handelt es sich nicht ausschließlich um JUnit-Tests. Zum Teil wurden aufgrund von Zeitgründen auch einfache Testfälle in simple Klassen implementiert und ein einzelner Durchlauf dieser genügte um zu prüfen, ob das Ergebnis dem zu erwarteten Wert entspricht. Auch wenn viel Wert darauf gelegt wurde JUnit-Tests zu verwenden war dies aufgrund der Komplexität mancher Methoden nicht immer mit dem Aufwand zu vereinbaren.
Beim Testen wurde zum Teil auf die sogenannte Bottom-Up-Methode zurückgegriffen. Dies bedeutet, dass die einzelnen Unterkomponenten des Gesamtsystems zunächst jeweils einzeln einer Überprüfung unterzogen wurden. Durch dieses Vorgehen sollte sichergestellt werden, dass alle Komponenten des Systems unabhängig voneinander funktionieren. Erst im Anschluss darauf würde dann auf die nächste Hierarchie-Ebene gewechselt werden und auch die Interaktion zwischen zwei oder mehr Komponenten mit getestet werden, was sich jedoch bei unserem konkreten Projekt aufgrund von vielen Abhängigkeiten im Systemkern nicht gerade einfach gestaltet. Daher hat sich auch ein direkter Übergang von den Komponententests zum kompletten Systemtest angeboten.
Bei den Komponententests per JUnit haben sich die Tester zunächst die einzelnen Empfehlungsalgorithmen vorgenommen, da für diese zur gleichen Zeit auch schon eine grafische Benutzeroberfläche (GUI) in Arbeit war. Es schien daher sinnvoll die Recommender frühstmöglich zu testen, da sich eventuell nötige Änderungen in diesen auch auf die GUI-Entwicklung würde auswirken können. Tatsächlich konnten bei den JUnit-Tests noch einige Fehler und Probleme in verschiedenen Algorithmen entdeckt werden, so dass sich diese Tests als voller Erfolg herausgestellt haben und das Qualität des Gesamtproduktes nachhaltig verbessern konnten.
Auch die bereits angesprochenen GUI-Elemente für die einzelnen Recommender wurden nach ihrer Fertigstellung wiederum Tests unterzogen, um ihre Funktionalität zu gewährleisten. Hierbei haben sich die GUI-Entwickler auch mit den Programmierern

der jeweiligen Algorithmen ausgetauscht, um die Anforderungen an die einzelnen Fenster noch besser verstehen zu können und somit durch vorbildliche Teamarbeit die Benutzerfreundlichkeit der endgültigen Oberfläche noch weiter zu erhöhen.
Nach dem Test der Recommender wurde mit der Überprüfung der einzelnen Importmodule fortgefahren. Hierbei galt es sicherzustellen, dass die Services bzw. Workflows fehlerfrei und erfolgreich in die hauseigene Datenhaltung eingefügt werden.

5.1.3 Algorithmen

Im Folgenden werden die implementierten Algorithmen anhand einiger Beispielergebnisse auf der Basis des in Abschnitt 5.1.1 geschilderten Datenbestandes demonstriert.

Association Rule Mining

Funktionsweise des Algorithmus. Der Association Rule Mining Algorithmus empfiehlt auf Basis einer Menge von Services eine Menge anderer Services, indem er Assoziationsregeln anwendet. Je stärker die Assoziationsregel ist, desto höher ist die Relevanz der Elemente der empfohlenen Menge. Assoziationsregeln werden mit in Kapitel 2.3.4 beschriebenen Methoden aus vorhandenen Workflows abgeleitet, indem Workflows als Mengen von Services betrachtet werden.

Korrektheitstest. Die Grundlage für die stichprobenartige Überprüfung des Verhaltens des Recommenders bildete ein Workflow[2] aus der myexperiment Bibliothek. Dieser wurde als Menge benutzter Services aufgefasst. Vorher generierte Assoziationsregeln wurden dann auf ihre Anwendbarkeit geprüft und gegebenenfalls angewandt. Die Relevanz der empfohlenen Services in Bezug auf den gegebenen Workflow ist der ausschlaggebende Faktor für die Sinnhaftigkeit der Empfehlung. Der gegebene Workflow heisst „EBI InterProScan“ und arbeitet mit XML-Dokumenten. Die empfohlenen Services waren in der Evaluation auch XML verarbeitende Services. Selbst ein Service `isDone` im Workflow, der oft zur Statusüberprüfung benutzt wird, wurde erkannt und ein ähnlicher Service `Success` empfohlen.

Verhalten auf unserem Datenbestand. Der Datenbestand liefert leider keine gute Basis für den Association Rule Mining Algorithmus. In Abschnitt 5.1.1 wurde bereits erläutert, dass aufgrund der Disjunktheit vorliegender Workflows die Parameter der Empfehlungen nicht präzise sind. Deswegen ist die eingestufte Relevanz empfohlener Servies nahuzu immer maximal. Bei einer stichprobenartigen Überprüfung wurde beobachtet, dass mit maximaler Relevanz empfohlene Services auch zum gegebenen Workflow semantisch passten.

[2] http://www.myexperiment.org/workflows/4.html

Relevance Feedback

Funktionsweise des Algorithmus. Der Relevance Feedback Algorithmus ist ein iterativer Algorithmus, der durch Feedback des Benutzers seine Empfehlung iterativ verbessert. Zur initialien Empfehlung werden vorhandene Informationen über Namen, Tags und Beschreibungen von Services verwendet, um die Nachbarschaft des gegebenen Services zu bestimmen. Zusätzlich berücksichtigt der Algorithmus Informationen der Services im Workflow, bezüglich dessen eine Empfehlung erzeugt wird. Bei jeder Iteration bewertet der Benutzer empfohlene Services und beeinflusst dadurch ihre Relevanz für die nächste Iteration. Da die Recommender-Schnittstelle keinen Feedback spezifiziert, wird lediglich die initiale Empfehlung des Algorithmus verwendet.

Korrektheitstest. Die stichprobenartige Überprüfung des Algorithmus wurde anhand des *EMBOSS*-Service `diffseq`[3] durchgeführt. Hierzu wurde zum einen nur auf Basis des Services und zum anderen im Kontext eines Workflows[4] eine Empfehlung erzeugt. Der Service `diffseq` findet die Gemeinsamkeiten und Unterschiede von nahezu identischen Sequenzen.

Empfehlung zum Service. Die durch den Algorithmus vorgenommene Analyse des Services `diffseq` ergab, dass Worte wie „compare“, „report“ und „similar“ relevant, während Worte wie „and“ und „two“ weniger relevant für den vorliegenden Service sind. Die ersten Services der erzeugten Empfehlung sind:
`BioCatalogue:PRECISService`, `EmbossACD:seealso`, `EmbossACD:showseq`, `EmbossACD:merger`, `EmbossACD:showpep`, `EmbossACD:extractfeat`, `BioCatalogue:OrgHub`. Es fällt auf, dass der erste empfohlene Service `BioCatalogue:PRECISService`[5] in seiner Beschreibung „Retrieves similar sequences and creates a protein family report“ beinhaltet. Damit besitzt er eine semantische Ähnlichkeit zu `diffseq`. Die Beschreibungen der anderen empfohlenen Services enthalten meist Worte „features“ und „sequences“ oder „similar“. Bei einzelner Betrachtung der Beschreibungen der emphohlenen Services fällt positiv auf, dass der Algorithmus semantisch ähnliche Services findet und dabei weder lange noch kurze Beschreibungen bevorzugt. Während längere Beschreibungen mehr gemeinsame Worte mit der Beschreibung des gegebenen Services haben können, ist die Relevanz dieser Worte niedriger.

Empfehlung zum Service mit Workflow-Informationen. Falls zusätzlich ein Workflow zur Empfehlungsgrundlage hinzugefügt wird, so werden die in ihm vorkommenden Services berücksichtigt. Bei der Betrachtung der ersten Services der Empfehlung bezüglich `diffseq` und des oben genannten Workflows fällt auf, dass die meisten nur bezüglich `diffseq` empfohlenen Services weiterhin empfohlen wer-

[3] http://emboss.sourceforge.net/apps/release/6.0/emboss/apps/diffseq.html
[4] http://www.myexperiment.org/workflows/754.html
[5] http://www.biocatalogue.org/services/2781

den. Der relevanteste Service der Empfehlung ist `BioCatalogue:Ensembl`[6]. Dies ist als positiv zu bewerten, da in dem vorliegenden Workflow mehrere mit `BioCatalogue:Ensembl` verwandte Services benutzt werden.

Syntaktischer Filter

Funktionsweise des Filters. Der „Syntaktische Filter“ ist einer der intuitivsten und effektivsten Filter des Recommenders. Wie an der Bezeichnung zu erkennen generiert er weniger Empfehlungen auf Basis einer Vorauswahl als dass er Services ausschließt, die ungeeignet sind. Die Eignung eines Services wird hier also rein syntaktisch überprüft, indem die Parameter-Typen überprüft werden. Services, deren Parameter äquivalent zum angegebenen Service-Port-Paar sind, werden empfohlen, andere gefiltert.
Ein Beispiel wäre der Service `EmbossACD:descseq` mit dem Parametertyp `sequence`, auf dessen Auswahl hin eben alljene Services empfohlen würden, deren Ein- oder Ausgabe-Parameter `sequence` Sequenzen sind.

Korrektheitstest. Der Korrektheitstest des Filters erfolgte durch die Überprüfung, ob die richtige Zahl von Empfehlungen generiert wurde. Für einen bestimmten Parametertyp lässt sich leicht überprüfen, wie viele Services in der Datenbank mindestens einen Parameter haben, der den Typ `sequence` hat. Die Zahl dieser Services lässt sich nun mit der Zahl der empfohlenen Services vergleichen, wenn der Algorithmus auf einem Service mit dem Parametertyp `sequence` getestet wird. Die Zahl der Empfehlungen muss die Zahl der in der Datenbank vorhandenen Services genau um eins unterschreiten, denn der verwendete Service selbst sollte nicht als sein eigener Nachfolger empfohlen werden. Eine Überprüfung dieses Ergebnisses kann für alle Parametertypen und Services erfolgen.

Verhalten auf dem Datenbestand. Das Verhalten des syntaktischen Filters wurde auf 4783 Service-Port-Paaren überprüft. Dabei ergaben sich 4262 nicht-leere Empfehlungsmengen, also Recommendations, die mindestens einen Service enthalten. Damit kann für eine deutliche Mehrheit der Services (89%) eine Empfehlung generiert werden; dennoch stellt sich natürlich die Frage nach den übrigen Services und der Ursache für fehlende Empfehlungen.

Untersuchung des Datenbestands bezüglich der Kriterien des Syntaktischen Filters. Um zu verstehen, wieso zu einigen Service/Parameter-Anfragen keine Empfehlungen generiert werden kann, muss man sich die Verteilung der Parametertypen und ihre Verwendung in Services in der Datenbank ansehen.
Der Filter sollte einerseits möglichst viele unpassende Services anhand ihrer Parameter ausschließen können, andererseits aber eine gewisse Auswahl an gut passenden Services präsentieren. Dementsprechend würde man sich eine Verteilung der Parameter so wünschen, dass jeweils zusammenhängende Gruppen beschränkter Größe

[6] http://www.biocatalogue.org/services/24

durch diese gemeinsamen Parameter erkannt werden können. Dabei würde jeder Parameter von mehreren Services genutzt und logischerweise jeder von einem Service verwendete Parameter auch in einem anderen Service gefunden.
In unseren Daten finden sich 575 Parametertypen, aber nur 72 davon werden in mehr als einer Service-Definition erwähnt. Diese ca. 500 nur einmal verwendeten Parametertypen zeichnen sich größtenteils dadurch aus, dass sie nicht-standardisiert sind, wie zum Beispiel „tns1:ArrayOf_xsd_string“. Oftmals handelt es sich damit bei den nur einzeln verwendeten Parametertypen um eine Kombination aus klassischen, standardisierten Typen (wie eben String, Boolean,...), die nur von einem Service so erkannt beziehungsweise bezeichnet wird. Da die Bezeichnungen relativ willkürlich gewählt sein können, lassen sich auch eigentlich übereinstimmende Typen („Array of Strings“, „StringArray“) nicht automatisch erkennen und zuordnen. Bei Typen wie „tns1:ArrayOf_xsd_string“ ist für den menschlichen Leser zwar zu erraten, dass es sich wohl um ein Array von bestimmten Strings handelt, dies maschinell zu erfassen und zu parsen ist allerdings schwierig; im Rahmen dieser Projektgurppe zumindest unmöglich.
Ein weiteres Indiz für die Theorie der Abhängigkeit von Standardisierung der Typen ist, dass die am häufigsten (mehrfach) verwendeten Typen zu den „üblichen“ Typen der Informatik gehören (Strings(1154 Erwähnungen), File (276), Boolean (152)) oder den im weitesten Sinne häufigsten Untersuchungsgegenstand der Bioinformatik bezeichnen (Seqall (109) und Sequence (57)).
Eine häufige Gemeinsamkeit derjenigen Services, die einander syntaktisch ähneln, ist, dass sie aus EMBOSS stammen. Dies wird darauf zurückgeführt, dass hier bereits die Services stärker standardisiert wurden und deswegen weniger eigenwillige, persönliche Bezeichnungen bekommen haben, als beispielsweise Services aus BioCatalogue, die nicht angepasst wurden und häufig nicht zu anderen Services zuzuordnen sind.

SimilarTags

Funktionsweise des Algorithmus. Der Algorithmus „Similar Tags“ nutzt die an die Services vergebenen Tags und interpretiert sie als starken Hinweis auf Gemeinsamkeiten. Tags werden von Menschen vergeben, um damit anderen Menschen und Maschinen einen Hinweis auf die Inhalte des Services zu geben, die vielleicht nicht aus dem Namen erkennbar sind. Damit sind Tags neben den Beschreibungen der Services, welche maschinell schwieriger zu interpretieren sind, potentiell sehr nützliche Hinweise für eine Empfehlung.
Prinzipiell empfiehlt der Algorithmus diejenigen Services, die eine möglichst große Menge von Tags mit dem aktuellen Service gemeinsam haben, weswegen von einem starken semantischen Zusammenhang der Services ausgegangen werden kann.
Dabei stellte sich schnell auch die Frage nach der Relevanz einzelner Tags: Natürlich ist gerade bei der freien Wahl von Tags für Services nicht auszuschließen, dass irrelevante Bezeichnungen vergeben werden oder andererseits relevante Informationen zum Zweck der Services ausgelassen werden. Zur Klärung dieser Frage musste der Datenbestand auf ebendiese Aspekte genauer untersucht werden.

Korrektheitstest. Zunächst wurde auch für diesen Algorithmus ein Korrektheitstest konstruiert. Hierbei wurden zunächst diejenigen Services aufgelistet, die mit einem bestimmten Tag versehen wurden. Diese Liste wurde mit den vom Algorithmus empfohlenen Services daraufhin verglichen, ob tatsächlich alle Services mit der gleichen Tag-Bezeichnung vorkommen.[7]
Wichtig ist, hier nicht nur auf die Zahl der zurückgegebenen Services zu achten, sondern tatsächlich die Liste gegen diejenigen Services abzugleichen, die mit dem gleichen Tag versehen wurden; immerhin generiert der Algorithmus seine Empfehlungen nicht nur auf Basis eines der Tags.

Der Test wurde vom Algorithmus bestanden.

Verhalten auf unserem Datenbestand. In unserer Datenbank mussten wir zunächst untersuchen, wie die Tags verteilt sind. Gäbe es nur einige wenige Tags, die sehr häufig verwendet werden, hätten diese wenig Aussagekraft; Tags, die dagegen nur ein einziges Mal zugeteilt wurden, liefern überhaupt keine Ergebnisse.

Verteilung der Tags. Eine wichtige Beobachtung ist, dass nicht allen Services der Servicedatenbank Tags zugeordnet sind. Diese Services sind für den Algorithmus also prinzipiell irrelevant und werden infolgedessen nicht empfohlen.
Es gibt insgesamt 885 Tags, von denen 444 Tags nur einem einzigen Service zugeordnet sind und damit ebenfalls nicht für Empfehlungen genutzt werden können.
136 Tags kommen in genau 2 Services vor, weitere 145 in bis zu 5 Services, 73 wurden bis zu 10 Mal vergeben. Auf bis zu 20 Referenzen kommen 45 Tags, 26 werden bis zu 50 Mal referenziert, 10 bis zu 100 Mal und genau 4 mehr als 100 Mal.
Es lohnt sich also ein Blick auf die Tags, die besonders häufig vergeben werden. Sofort fällt auf, dass diese Tags nur die Herkunft eines Services bezeichnen (`BioMoby` mit Frequenz 742, `soaplab` - 576, `EMBRACE` - 312, `EMBOSS` mit 244 Services) und wenig über den Inhalt der Services verraten.
Tags, die inhaltlich aussagekräftig sind und deswegen sehr nützlich für Empfehlungen sein könnten, sind etwas weniger stark genutzt. Es ist auch zu vermuten, dass die geringe Nutzung der Tags hier starke Auswirkungen hat: Beispielsweise ist eigentlich zu erwarten, dass sich mehr als 52 der Services mit Proteinsequenzen beschäftigen. Dennoch wurde der Tag `protein sequence` nur 52 Mal vergeben; ähnlich ist es mit `Mutation` (21), `Sequence Analysis` (38), `database` (77) und `edit` (92).
Aus diesen Beobachtungen heraus ergab sich die Notwendigkeit, die Empfehlungen noch einmal weiter zu filtern. Inhaltlich unbedeutsame Tags – wie die zur Herkunft des Algorithmus – sollten automatisch aussortiert werden. Natürlich könnte man durch eine semantische Analyse oder zumindest eine Checkliste von „irrelevanten Begriffen“ hier aufwändig filtern. Stattdessen werden die Tags nicht semantisch untersucht, sondern nach ihrer Häufigkeit geordnet, wobei der modifizierte Algorithmus dann besonders häufig und freigebig zugeordnete Tags ignoriert und somit die Relevanz der Tags in die Empfehlung miteinbezieht.

[7] Der beschriebene JUnit-Test befindet sich in der Klasse `SimilarTagsTest` im Projekt `pg-sassd.recommender.test`.

TF-IDF

Das Verhalten des in Kapitel 3.4.4 vorgestellten TF-IDF-Recommenders soll hier beispielhaft anhand des *EMBOSS*-Services `diffseq`[8] demonstriert und evaluiert werden.
Der Algorithmus analysiert lediglich Beschreibungen von Services. Die Beschreibung von `diffseq` in der Datenbank ist „Compare and report features of two similar sequences“. Die Analyse dieser Beschreibung ergab, dass Worte wie „compare“ und „similar“ relevant, während Worte wie „sequences“ und „two“ weniger relevant für den vorliegenden Service sind. Auffällig ist, dass den Worten „and“ und „of“ keinerlei Relevanz zugewiesen wird. Dies lässt sich durch die Verwendung von in Kapitel 2.3.3 beschriebenen Stoppwörtern erklären, die eingesetzt werden, um das Verfahren zu verbessern.
Die ersten empfohlenen Services sind:
`BioCatalogue_EMBOSS_seealso`, `BioCatalogue_seealsoService`, `EmbossACD_seealso`, `BioCatalogue_extractfeatService`, `BioCatalogue_EMBOSS_extractfeat`, `EmbossACD_extractfeat`, `BioCatalogue_showseqService`.
Es ist zu erkennen, dass alle Beschreibungen der empfohlenen Services mindestens ein für `diffseq` relevantes Wort enthalten. Im Fall des empfohlenen Services `showseq`, der Eigenschaften von Sequenzen anzeigt und in seiner Beschreibung Worte „features“ und „sequences“ beinhaltet, führt es zur korrekten Feststellung einer semantischen Beziehung zwischen dem gegebenen und dem empfohlenen Service. Im Fall des empfohlenen Services `seealso` ist die Einschränkung auf Beschreibungen hingegen ein Nachteil. Dieser Service hat semantisch keine Ähnlichkeit zu `diffseq`, sondern enthält in seiner Beschreibung das für `diffseq` relevante Wort „similar“. Diese Problematik ist darauf zurückuzführen, dass gegebene Beschreibungen der Services der Datenbank meist nur aus wenigen relevanten Worten bestehen.
Es fällt weiterhin auf, dass es sich bei einigen der Empfehlungen um identische Services zu handeln scheint, die jedoch in der Gesamtheit der verwendeten Datenquellen unterschiedlich parametrisiert auftauchen. Die Beschreibungen der einzelnen Vorkommen sind jedoch nahezu identisch. Folglich weist der Recommender den einzelnen Worten der beschreibungen dieser Services jeweils identische Relevanz zu und empfiehlt die Services auch mit dem gleichen Gewicht.

[8] http://emboss.sourceforge.net/apps/release/6.0/emboss/apps/diffseq.html

Category-Filter

Die im Kapitel 3.4.5 vorgestellte Implementierung des „Category Filters“ soll in diesem Abschnitt anhand von Beispielen näher in der Arbeitsweise vorgestellt werden. Zudem sollen die Ergebnisse des „Recommenders“ evaluiert werden. Die Evaluation beinhaltet unterschiedliche Aspekte. Erstens wird betrachtet ob zu einem „Service“ eine Recommendation eine Lösung liefert. Dies kann durch fehlende Kategorieinformationen eines Services unter Umständen fehlschlagen. Zweitens wird die Lösung näher betrachtet und der logische Zusammenhang zwischen zwei Services beispielhaft erläutert. Drittens wird untersucht, ob eine „Cross-Plattform-Recommendation“ existiert. Die gesammelten Daten stammen aus unterschiedlichen Plattformen (wie BioCatalogue oder EMBOSS) und wurden in die Datenbank eingefügt, dadurch ist es interessant, ob diese Services einen Zusammenhang zueinander finden.
Im Anhang A.2 angefügt stehen die Ergebnisse der Recommendation für zehn Beispielservices.
Durch die Tests wurde ersichtlich, dass es viele unvollständige Daten im System gibt. Dadurch ist es dem „CategoryFilter“ häufig nicht möglich eine Empfehlung zu geben. Diese Beispiele ergeben eine leere Recommandation und wurden in diesem Beispiel nicht aufgenommen.
Die Ergebnisse zeigen, dass die Empfehlungen, die der „CategoryFilter“ berechnet hat, auf Basis einer Kategoriezugehörigkeit korrekt sind. Der logische Zusammenhang zwischen zwei Services ist leider allein durch den Recommender nicht immer klar. Durch die alleinige Betrachtung der Kategoriegleichheit werden viele Informationen der Services nicht beachtet und dadurch entstehen „unlogische“ Empfehlungen. Services gehören zum Gleichen Kategoriebereich, können aber nicht innerhalb eines Workflows miteinander verknüpft werden. Deswegen sollte der „CategoryFilter“ nicht als alleiniger Empfehlungsalgorithmus verwendet werden. Er ist in der Lage den Suchraum der logisch verknüpften Services zu begrenzen, jedoch ist er auf Grund von Informationsmangel nicht in der Lage genaue Empfehlungen zu geben.
Die Ergebnisse sind stellvertretend für einige Testreihen. Hieraus hat sich ergeben, dass die „Cross-Plattform-Recommendation“ existiert, jedoch selten vorkommt. Dies liegt an den abweichenden Kategoriebezeichnungen in den Systemen. Es ist denkbar, dass durch ein geschicktes „mergen“ der Kategorien eine höhere „Cross-Plattform-Recommendation-Quote“ bewirkt. Am Beispiel des Services
„BioCatalogue_TreeBASE“ wird ersichtlich, dass der Recommender auch ein Service mit der gleichen Kategorie aus dem Emboss-System vorschlägt.

MutualInformation

Funktionsweise des Algorithmus. Maßgeblich für die von diesem Algorithmus berechneten Empfehlungen sind zwei Wahrscheinlichkeitsverteilungen $p(\cdot)$ und $p(\cdot,\cdot)$, welche aus der Häufigkeit des Auftretens eines einzelnen Services in der Workflowdatenbank bzw. eines Paares von Services in demselben Workflow abgeleitet werden. Darüber hinaus besteht die Möglichkeit zur Einschränkung und Gewichtung gemeinsamer Vorkommen von Services hinsichtlich der Distanz (das ist die Anzahl der Kanten im Datenfluss) zwischen ihnen.

Dahinter verbirgt sich die Idee, dass Services, welche gemeinsam in Workflows auftreten, auch in anderen Workflows sinnvoll zusammen eingesetzt werden können, wobei diese relative Häufigkeit durch die Kehrwerte der Wahrscheinlichkeiten für das einzelne Auftreten der beteiligten Services gewichtet wird; denn das gemeinsame Auftreten kann als weniger bedeutend gewertet werden, falls mindestens einer der beteiligten Services überhaupt (also auch ohne den anderen) häufig genutzt wird.

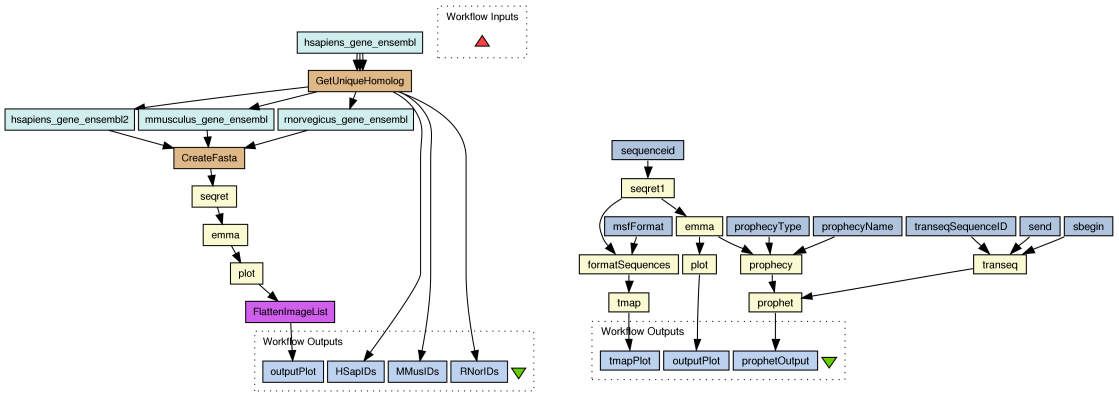
Korrektheitstest. Die grundlegende Funktionsfähigkeit der Implementierung wurde anhand einer Menge von vier sehr einfachen, konstruierten Workflows überprüft. Da der Algorithmus seine Stärke erst auf einem ungleich größeren Datenbestand entfaltet, konnten einige Fehler erst in der manuellen Evaluationsphase aufgefunden und behoben werden.

Verhalten auf unserem Datenbestand. Die Implementierung des auf dem Prinzip der „Mutual Information“ basierenden Algorithmus wird anhand einer Beispielanfrage für den aktuellen Service[9] `seqret` aus der EMBOSS-Suite demonstriert. Besonderes Augenmerk wird dabei auf den Parameter `distanceFactor` gelegt. Auf eine Variation des Parameters `maxPredecessorLevel` wird hingegen verzichtet, da sie analoge Ergebnisse zu jenen herbeiführt, die im Falle der Modifizierung von `maxSuccessorLevel` ermittelt werden.

Anmerkungen. Angemerkt werden muss, dass die „Wirksamkeit“ des Algorithmus durch den immer noch zu geringen Umfang des tatsächlichen Datenbestands beeinträchtigt wird. Wie in Abschnitt 3.4.6 erläutert, ergibt sich für (verschiedene) Services i und j der Wert als Logarithmus des Verhältnisses zwischen der Wahrscheinlichkeit $p(i, j)$ und dem Produkt $p(i) \cdot p(j)$. Diese Wahrscheinlichkeiten werden ausgehend von der Anzahl der Vorkommen der Dienste in der Workflowdatenbank berechnet. Da jedoch nur solche Dienste, zu denen ein SIB-Objekt erzeugt wurde (also genau jene, welche auch in der Servicedatenbank verzeichnet sind), empfohlen werden sollen, wird die Berechnung auf diejenigen Service-IDs der Workflowdatenbank eingeschränkt, welche über Alias-Einträge mit Services der Service-Bibliothek assoziiert sind. Durch diese Einschränkung ergibt sich der mit 137 bzw. 1.021 Elementen verhältnismäßig geringe Umfang der Definitionsbereiche der Abbildungen $p(\cdot)$ und $p(\cdot, \cdot)$.
Ein weiteres Problem ergibt sich daraus, dass einzelne Services anscheinend nicht so häufig in unterschiedlichen Workflows genutzt werden, wie es der fundamentale Aspekt der Web-Services – die Wiederverwendbarkeit – vermuten ließe. Wie der Evaluation des Datenbestandes (vgl. Abschnitt 5.1.1) zu entnehmen ist, sind die am häufigsten in Workflows verwendeten Services (kopierte) *Skripte*, können also nicht von dem Recommender-System empfohlen werden.
Der im Folgenden genutzte Beispielservice `seqret` ist außerdem für eine wissenschaftliche Auswertung insofern wenig aussagekräftig, als er zwar in 18 Workflows vorkommt, diese aber lediglich aus 12 Kopien des Workflows „Biomart and EMBOSS

[9] Der Algorithmus macht von zusätzlichen Angaben wie etwa dem selektierten Port oder dem umfassenden Workflow keinen Gebrauch.

Analysis“[10] (Abbildung 5.1(a)) sowie 6 Kopien des Workflows „Workflow Version of the EMBOSS Tutorial“[11] (Abbildung 5.1(b)) bestehen – offenbar eine Schwäche der Plattform. Einen Vorteil bieten diese Workflows immerhin doch: ihre geringe Größe erleichtert die manuelle Auswertung.



(a) „Biomart and EMBOSS Analysis“ (b) „Workflow Version of the EMBOSS Tutorial“

Abbildung 5.1: Beispiel-Workflows (Quelle: MyExperiment)

Ergebnisse. Wie bereits erwähnt wurde die Auswertung unter einer Variation der Parameter `distanceFactor` und `maxSuccessorLevel` durchgeführt. Der Wert von `maxPredecessorLevel` war dabei stets 0, vorhergehende Services wurden bei der Berechnung von $p(\cdot, \cdot)$ also ignoriert.
Im linken Teil der Tabelle 5.6 sind die obersten zehn Empfehlungen für variierende Einschränkungen der Nachfolgerebenen und einen Distanzfaktor von 0,9 angegeben, wohingegen im rechten Teil Empfehlungen bei einem Parameterwert von 0,5 dargestellt sind. (Untersucht wurden weiterhin Empfehlungen für einen Faktor von 0,1, sie unterschieden sich allerdings (in der Reihenfolge) nicht von jenen für den Wert 0,5.)

Zunächst lässt sich feststellen, dass bei der Einschränkung auf unmittelbare Nachfolger (`maxSuccessorLevel` $= 1$) unabhängig vom Distanzfaktor ausschließlich der Service `emma` empfohlen wird. Im ersten Beispiel-Workflow ist dies tatsächlich auch der einzige direkte Nachfolger, im zweiten Beispiel ist es der einzige, der einem Datensatz der Service-Bibliothek zugeordnet werden konnte.
Deutlich wird ebenfalls, dass eine geringe Abweichung vom Standardwert (1,0) die „räumliche“ Nähe der Services kaum berücksichtigt. Die Instanzen des Services `emma` befinden sich dort lediglich auf den Plätzen sieben und acht, während ihnen bei der alternativen Konfiguration die obersten Plätze zugewiesen werden.
Verschiedene Instanzen mit demselben Servicenamen ergeben sich etwa durch das Vorkommen eines lokalen (im EMBOSS-Archiv) sowie eines WSDL-Services (in BioCatalogue). Den Instanzen zu `emma`, `tmap` und `prettyplot` (hier nicht ersichtlich)

[10] MyExperiment-Workflows mit den IDs 158, 754, 821, 916, 997, 1365, 1652, 1653, 1794, 2213, 2846, 2847.
[11] MyExperiment-Workflows mit den IDs 159, 827, 996, 1364, 1792, 2226.

Tabelle 5.6: Empfehlungen für Distanzfaktoren von 0,9 (links) und 0,5 (rechts)

maxSuccessorLevel = 1:	maxSuccessorLevel = 1:
1. emma (BioCatalogue)	1. emma (BioCatalogue)
2. emma (EMBOSS)	2. emma (EMBOSS)
maxSuccessorLevel = 2:	**maxSuccessorLevel = 2:**
1. getorf (BioCatalogue)	1. emma (BioCatalogue)
2. prophecy (BioCatalogue)	2. emma (EMBOSS)
3. tmap (BioCatalogue)	3. getorf (BioCatalogue)
4. getorf (EMBOSS)	4. prophecy (BioCatalogue)
5. prophecy (EMBOSS)	5. getorf (EMBOSS)
6. tmap (EMBOSS)	6. prophecy (EMBOSS)
7. emma (BioCatalogue)	7. tmap (BioCatalogue)
8. emma (EMBOSS)	8. tmap (EMBOSS)
9. seqret (BioCatalogue)	9. seqret (BioCatalogue)
10. prettyplot (BioCatalogue)	10. prettyplot (BioCatalogue)
maxSuccessorLevel = 3:	**maxSuccessorLevel = 3:**
1. tmap (BioCatalogue)	1. emma (BioCatalogue)
2. tmap (EMBOSS)	2. emma (EMBOSS)
3. getorf (BioCatalogue)	7. tmap (BioCatalogue)
4. prophecy (BioCatalogue)	8. tmap (EMBOSS)
5. getorf (EMBOSS)	3. getorf (BioCatalogue)
6. prophecy (EMBOSS)	4. prophecy (BioCatalogue)
7. emma (BioCatalogue)	5. getorf (EMBOSS)
8. emma (EMBOSS)	6. prophecy (EMBOSS)
9. seqret (BioCatalogue)	9. seqret (BioCatalogue)
10. prettyplot (BioCatalogue)	10. prettyplot (BioCatalogue)

werden stets dieselben Werte zugewiesen, da die entsprechenden Einträge in der Workflowdatenbank mit Aliasen zu beiden Objekten der Servicedatenbank versehen sind.

Unterscheiden sich die Ergebnisse für die Instanzen zweier verschiedener Services nicht, so kann es zu einer Verschachtelung kommen, wie im Falle von `getorf` und `prophecy` (in beiden Konfigurationen).[12]

Die verhältnismäßig guten Bewertungen für den Service `tmap` haben ihren Ursprung in der geringen Häufigkeit des Auftretens dieses Services. Während Services wie `emma` insgesamt häufiger zusammen mit dem Ausgangsservice `seqret` auftreten, kommen sie auch in Workflows ohne `seqret` vor. Im Unterschied dazu ist `tmap` einzig in solchen enthalten, in denen auch `seqret` referenziert wird. In dieser Hinsicht besteht

[12]Zur Vermeidung solcher Resultate könnte das Recommender-System als sekundäres Sortierkriterium den Namen des Services nutzen!

– zumindest nach dem Prinzip der „Mutual Information“ – in der Tat ein engerer Zusammenhang zwischen ihnen!

5.1.4 Qualitätstest

Mithilfe der Korrektheitstests wurde überprüft, ob die Implementierungen der Recommender-Algorithmen tatsächlich die gewünschten Funktionalitäten aufweisen. Auch wenn diese funktionale Korrektheit sichergestellt wurde, ist jedoch noch kein Nachweis darüber erbracht, ob der Recommender auch tatsächlich im Praxisbetrieb gute Ergebnisse erzielen würde, da nicht klar ist, ob die entworfenen Empfehlungsstrategien tatsächlich die gewünschte Effektivität und die von betrachteten Auswahlkriterien wirklich die entsprechende Relevanz besitzen. Aufgrund dieser Tatsache soll zusätzlich zu den Korrektheitstests ein sogenannter Qualitätstest durchgeführt werden, welcher das Ziel hat, die semantische Sinnhaftigkeit der Empfehlungen und die Nützlichkeit des Recommenders beim Praxisbetrieb in dem für ihn vorgesehenen Einsatzbereich der Bioinformatikworkflows zu evaluieren.
Die Konzeption eines solchen Qualitätstests stellt jedoch ein nichttriviales Unterfangen dar, denn der Begriff der semantischen Qualität ist letztendlich nicht vollständig objektiv zu erfassen und entzieht sich somit erst recht einer mathematischen Definition. Auch eine subjektive Beurteilung der Sinnhaftigkeit der empfohlenen Services ist für die Teilnehmer der Projektgruppe mangels entsprechender Fachkenntnisse im Bereich der Bioinformatik kaum möglich und das tiefgründige Verständnis der biologischen Hintergründe von den zahlreichen Services kann von den Software-Entwicklern auch nicht erwartet werden.
Einen Lösungsansatz liefern jedoch die bereits vorhandenen Workflows aus dem MyExperiment-Projekt, welche ohnehin als Grundlage für unsere Empfehlungen importieren worden sind. Die Tatsache, dass Informationen über diese Workflows als Datenquelle für den Recommender verwenden werden, zeigt bereits, dass den Workflows eine gewisse Qualität zugesprochen wird und somit ist letztendlich auch davon auszugehen, dass sie eine ähnlich große semantische Sinnhaftigkeit aufweisen können, wie die Workflows, die schlussendlich durch die Service-Empfehlungen des Recommenders entstehen sollen.
Daher liegt die Idee nahe, Teile dieser Workflows für den Qualitätstest durch die Recommender nachbauen zu lassen. So könnte beispielsweise der Anfang eines MyExperiment-Workflows vorgegeben werden und anschließend der Recommender aktiviert werden, um einen Nachfolge-Service für die betrachtete offene Stelle zu empfehlen. Selbstverständlich müsste für diesen Test der entsprechende Workflow aus unserer MyExperiment-Datenbank entfernt werden, da die Aufgabe für den Recommender zu einfach wäre, wenn ein Workflow in der Datenbank exakt den gleichen Beginn aufweisen würde wie das vorgelegte Testobjekt. Der Recommender sollte bei mehr als 1000 ihm bekannten Bioinformatikworkflows in der Lage sein, auch ohne Kenntnis des tatsächlich nachzubauenden Workflows, eine sinnvolle Empfehlung für die offene Position abzugeben. Doch auch bei einem solchen Ablauf des Qualitätstests ist es, aufgrund der fehlenden Fachkenntnisse im Bereich der Bioinformatik, nur möglich

die Sinnhaftigkeit der Empfehlung auf Basis des zugrundeliegenden Workflows zu bestimmen. Somit ist die Aussagekraft des Qualitätstest immer noch abhänig von der Qualität der verwendeten Workflows.
Bei einem solchen Qualitätstest ist es insbesondere möglich, dass der aufgrund der Datenbasis empfohlene Workflow tatsächlich der erwartete Service ist, es jedoch weitere Services gibt, die tatsächlich eine ähnliche oder sogar bessere semanische Sinnhaftigkeit an der entsprechenden Stelle im Workflow aufweisen würden. Diese Eigenschaft kann jedoch wiederum aufgrund der mangelnden Bioinformatik-Expertise nicht bewertet werden kann. Es muss jedoch bei jeder Bewertung einer Empfehlung bedacht werden, dass der geschilderte Fall eintreten kann. Es muss also sorgfältig abgewogen werden, wann eine Empfehlung tatsächlich als am besten passend betrachtet werde sollte.

Auswahl eines Evaluationsverfahrens

Für die Evaluation der Arbeit des Recommenders müssen Methoden entwickelt werden, die die Ergebnisse nach ihrer Nützlichkeit und „Arbeitstauglichkeit“ bewerten. Hierfür können selbstverständlich nicht die Algorithmen genutzt werden, die gerade auch der Recommender verwendet. Stattdessen müssen Verfahren entwickelt werden, die die Algorithmen nutzen, aber die Ergebnisse unabhängig auswerten.
Da einige der Algorithmen abhängig von den aus den Daten extrahierten Informationen arbeiten, müssen sie diese zur Verfügung haben. Würden die Algorithmen aber alle bekannten Workflows zur Berechnung verwenden, blieben keine mehr als Grundlage für Testfälle übrig.
Es muss also eine Einteilung der bestehenden Daten in disjunkte Trainings- und Testmengen vorgenommen werden, um danach die Evaluationsverfahren auf diesen Mengen durchzuführen.

Test- und Trainingsmenge

Ein weiteres Problem ist die Tatsache, dass einige der MyExperiment-Workflows möglicherweise einzigartig sein können, so dass der Recommender sie unmöglich auf Basis der anderen Workflows nachvollziehen kann. Dies kann beispielsweise der Fall sein, wenn ein Workflow nur aus sehr neuen Services besteht, die noch von keinem anderen Workflow verwendet werden oder ausschließlich aus lokalen Services besteht, die nur vom Ersteller verwendet werden. Daher muss darauf geachtet werden, eine große Menge der beschriebenen Vergleichs-Tests durchzuführen, damit ein statistisch aussagekräftiges Ergebnis erzeugt werden kann. Es ist daher vorgeschlagen worden, die MyExperiment-Workflows in zwei unterschiedlich große Mengen aufzuteilen und die erste Menge per Update-Prozess der Recommender-Datenbasis hinzuzufügen, während auf der zweiten Menge anschließend die Tests durchgeführt werden, so dass also ein Bruchstück von jedem Workflow aus dieser zweiten Menge dem Recommender zur Vervollständigung vorgelegt wird. Dieser Vorgang des Teilens und Testens wird dann entsprechend häufig mit unterschiedlichen Mengen durchgeführt. Diese Methode ist auch unter dem Namen *Kreuzvalidierung* bekannt [AL08].

Je größer die Trainingsmenge, desto besser sind die zu erwartenden Ergebnisse, weil der Algorithmus eine größere Entscheidungsgrundlage hat. Je größer die Testmenge, desto differenzierter kann die Arbeit des Algorithmus bewertet werden.
Es gibt hier verschiedene „übliche“ Verhältnisse, die je nach Forschungsgebiet und untersuchtem Verfahren variieren. Aufteilungen der Datenmenge im Verhältnis von 1:10 bis zu 1:1 sind dabei vertreten.
Um bessere Ergebnisse zu erzielen und mehr Tests durchführen zu können, wird oft eine k-Kreuzvalidierung durchgeführt. Dazu werden die Daten zunächst in k möglichst gleich große Teilmengen eingeteilt. Jeweils $k-1$ dieser Teilmengen werden als Trainingsmenge verwendet, die übrig gebliebene ist die Testmenge. Bei einer gewünschten Aufteilung im Verhältnis 1:10 würden also 11 gleichgroße Teilmengen gebildet, eine dieser Mengen bildet dann die Testmenge. Da jede der 11 Teilmengen dafür ausgewählt werden kann, ist es auch möglich, 11 Durchläufe des Verfahrens zu starten; das Ergebnis der Evaluation wird dann über den Durchschnitt dieser Testläufe gebildet.
Eine spezielle Variante der k-Kreuzvalidierung ist das Leave-One-Out-Verfahren, bei dem die Überprüfung für sämtliche Elemente der Datenmenge erfolgt: Das zu überprüfende Element ist die gesamte Testmenge, die übrigen Elemente stellen die Trainingsmenge. Hier wäre k also gleich n.
Je größer k gewählt wird, desto weniger Elemente sind in einer Testmenge. Beim Leave-One-Out-Verfahren ist nur eine geringe Feineinstellung durch die Angabe der Zahl der Durchläufe möglich. Dabei kann eine beliebig große Teilmenge der Elemente gegen alle übrigen getestet werden. In beiden Verfahren ist es möglich, sämtliche Elemente der zur Verfügung stehenden Datenmenge jeweils auf Grundlage der übrigen zu testen; es unterscheidet sich nur je nach Wahl von k die Größe der Trainingsmenge, die zur Generierung eines Ergebnisses verwendet wird.

Bewertungsverfahren

Eine Bewertung der Algorithmen sollte einerseits intuitiv einsichtig sein, um über die Qualität der Ergebnisse sprechen zu können, und andererseits möglichst genau die Anforderungen an den Recommender abbilden. Diese Anforderungen ergeben sich für ein Programm unter anderem aus den Nutzungsgewohnheiten von Usern, die bisher noch nicht bekannt sind; dementsprechend wird auch hier auf verallgemeinerte Annahmen zurückgegriffen. Diese Annahmen gehen davon aus, dass Services häufig in einer im Datenbestand vorkommenden Reihenfolge in einem Workflow auftreten. Somit ist es zum Einen möglich Services aufgrund des Datenbestandes zu empfehlen und zum Anderen die Qualität aller Empfehlungen anhand des Datenbestandes zu bestimmen.
Hier stellt sich weiterhin die Frage, wie groß der Anteil der für die Empfehlung vorgegebenen Workflows sein soll. Es kann als sinnvoll angenommen werden, einen gewissen Mindestanteil an Services aus dem vollständigen Workflow anzugeben, um eine ausreichende Aussagekraft des Workflow-Beginns sicherzustellen. Eine gute Möglichkeit, welche die zuvor genannte Schwierigkeit berücksichtigt, wären mehrere Tests mit steigender Zahl von bereits in dem Workflow vorhandenen Services, wobei die

Strafpunkte mit einer von dieser Zahl abhängigen exponentiellen Funktion muliplizierte werden. Anhand der durchschnittlichen Größe der Workflows lässt sich eine geeignete Funktion bestimmen. Denkbar wäre beispielsweise $(1{,}1^x)$, wobei x die Anzahl der bereits vorhandenen Services ist. Denn auf diese Weise werden Recommender bestraft, die selbst mit vielen Informationen nur unzureichende Empfehlungen liefern. Es wird jedoch auch berücksichtigt, dass eine gewisse Anzahl an Services vorhanden sein muss, um eine vernünftige Empfehlung zu erzeugen.

Die Größe der beiden Workflow-Mengen muss ebenso festgelegt werden wie die Anzahl der Wiederholungen des gesamten Testverfahrens mit unterschiedlichen Aufteilungen der Workflows in die beiden Mengen.

Eine mögliche fortgeschrittene Variante des Qualitätstests beinhaltet den Test von unterschiedlichen Recommender-Konfigurationen. Es könnten also in diesem Fall verschiedene Empfehlungs-Algorithmen mit unterschiedlichen Gewichtungen für die Bewertung verwendet und die Ergebnisse dieser einzelnen Konfigurationen miteinander verglichen werden. Auf diese Weise wäre es letztendlich sogar denkbar, mit Hilfe des Qualitätstests eine optimierte Konfiguration bezüglich der Algorithmen-Auswahl für unseren Recommender zu ermitteln, wenn er für die betrachteten Services und Workflows aus dem Bereich der Bioinformatik verwendet wird.

Auswertungsmöglichkeiten

Es ist eine Auswertung der Ergebnisse nach der Platzierung des „gewünschten“ Services vorgenommen worden und dabei seine Position in der Liste der Vorschläge berücksichtigt worden. Die genauere Auswertung hängt nun auch davon ab, wie diese Positionen bewertet werden: Hierbei stellt sich die Frage, wie eine Position ungleich der ersten zu bestrafen ist. Ist der Unterschied zwischen einem Auftauchen des „gewünschten“ Services an 112. und 113. Stelle genau so gravierend wie der Unterschied zwischen Position 5 und 6? Die Bewertung der einzelnen Positionen hat sehr große Auswirkungen auf die Evaluation, da die hierdurch die Qualität definiert wird.

Um diese Frage zu beantworten, sind verschiedene Möglichkeiten betrachtet worden, die Positionen zu bewerten. Einerseits mit der linearen Zuordnung von Platzierung zu Bewertung (Position x entspricht dabei x Strafpunkten), aber auch nicht-linearen Verfahren.

Außerdem schien es sinnvoll, Ergebnisse, einerseits der Übersichtlichkeit wegen, andererseits der Verwendung im Programm geschuldet, gruppiert zu beurteilen. Im fertigen Programm wird, ähnlich wie bei populären Suchmaschinen, immer eine gewisse Anzahl an Empfehlungen gleichzeitig in einer Liste zu sehen sein. Diese Art der Darstellung muss bei der Bewertung von Empfehlungen berücksichtig werden, denn hierdurch ist es deutlich schlechter, wenn eine Empfehlung auf der zweiten Seite auftaucht, als wenn sie an der letzten Position der ersten Seite auftaucht.

Zur Darstellung der Ergebnisse sind einige der üblichen Methoden der deskriptiven Statistik implementiert worden, so sollen Durchschnitt, (Modus,) Median und eventuell zugehörige Quartile berechnet werden. Mithilfe dieser auch optisch leicht

zu vergleichenden und in Grafiken (wie z. B. Boxplots) darstellbaren Werte soll die intuitive Auswertung in einer einfachen Weise gewährleistet sein.

Linear vergebene Strafpunkte

Bei linear vergebenen Strafpunkten ist die Anzahl der Strafpunkte linear abhänig von der Position des „gewünschten“ Service in der Menge der Empfehlungen. Diese Methode hat den großen **Vorteil** eines intuitiv sehr leicht nachvollziehbaren Ergebnisses, dessen Auswertung durch die üblichen Methoden der deskriptiven Statistik weiterhin leicht einsichtig bleiben: Verändert sich der Durchschnittswert der Ergebnisse um 1, so ist das Ergebnis um eine Platzierung besser oder schlechter geworden. Bei nicht-linearen Funktionen muss der betragsmäßige Unterschied immer auch im Kontext der Funktion betrachtet und ausgewertet werden.

Der **Nachteil** einer linearen Auswertung liegt in der Nichtberücksichtigung der graphischen Darstellung im Programm und der Verarbeitung durch den Menschen. Werden allen Ergebnissen einfach Bewertungen entsprechend der Position zugewiesen, bleibt die Tatsache außen vor, dass ein Ergebnis auf Platz 112 einfach überhaupt keine Relevanz mehr hat, wenn es um die tatsächliche Nutzung geht.

Gerade in der Benutzung eines Programms, das einem Benutzer die Arbeit erleichtern und sie beschleunigen soll, muss berücksichtigt werden, dass er kaum die vollständige Liste von häufig über 1000 möglichen Vorschlägen durchgehen möchte.

Auch weil der Nutzer selbst davon ausgeht, dass die relevantesten Vorschläge am Kopf der Liste stehen werden, sieht er sich eher nur die ersten x Vorschläge an, bevor er einen gewünschten Service dann eventuell doch selbstständig und gezielt sucht. Das heißt, es ist nicht unbedingt 5 Mal schlechter, wenn ein Service auf Position 5 gelistet wird (denn diese Position befindet sich vermutlich noch im leicht auszuwählenden Bereich der Liste, wird also gleichzeitig mit Platz 1 angezeigt), als wenn er auf Platz 1 landet. Hingegen wird ein Service auf Platz 25, der im relativen Verhältnis auch „nur“ 5 Mal schlechter ist als der auf Platz 5, wohl kaum jemals vom Nutzer berücksichtigt. Bei der Anzeige der 5 ersten Services in einer Empfehlungsliste, ist die Position auf Platz 5 etwas, aber nicht wesentlich schlechter als die auf Platz 4, die auf Platz 6 aber wesentlich schlechter als die auf Platz 5.

Hier tritt bereits die Frage aus, ob Platzierungen auf bestimmten Positionen vielleicht einfach unberücksichtigt bleiben sollten. Es ist aber abhängig von der Repräsentation und dem Nutzungsverhalten eines Anwenders, ab wann ein Service in der Empfehlung als irrelevant betrachtet werden kann, da er an dieser Stelle wahrscheinlich nie vom Nutzer gefunden wird.

Gruppenweise Auswertung

Ein erster Ansatz, um das erwähnte Problem des Unterschieds zwischen zwei Ergebnissen, die dennoch auf einer Seite angezeigt werden und solchen die in ganz unterschiedliche Bereiche fallen, zu lösen, ist die einfache Gruppierung.

Abhängig von der Bedienoberfläche[13] könnten also Raster zur Auswertung der Ergebnisse gebildet werden. Auch ohne die Informationen zur letztlichen Implementierung können diese Raster intuitiv aufgeteilt werden: Eine Auswertung darüber, wie oft der „gewünschte“ Service unter den ersten drei, fünf, zehn, zwanzig... auftaucht, liefert ebenfalls eine intuitiv einsichtige Auswertung der Arbeit des Recommenders und wird häufig verwendet.
Die Auswertung in graphischer Form bietet sich über Histogramme an, die genau solche Verteilungen besonders gut darstellen können und auch einen etwas weniger komplexen Überblick darüber bieten, wie genau die Ergebnisse verteilt sind.
In jedem Fall ist die einfachste Gruppierung der Ergebnisse die in „relevante“ und „irrelevante“ Positionen in der Empfehlungsliste. Geht man davon aus, dass Services, die später als auf dem 20. Listenplatz empfohlen werden, gar keine Berücksichtigung mehr finden, lässt sich einfach binär auswerten: Wie viele der Tests haben überhaupt ein positives, das heißt „praxistaugliches“ Ergebnis, und wie viele nicht?
Für eine detaillierte Auswertung haben ist aber für eine etwas feinere Einstellung vorgesehen.

Nichti-lineare Straffunktionen

Wie bereits erwähnt, wäre es wünschenswert, dass die Straffunktion die Unterschiede zwischen einzelnen Positionen berücksichtigt, aber auch mit einbezieht, dass eine Platzierung ab einem gewissen Wert irrelevant ist und sich etwaiges Blättern in einer Liste von Anzeigen auf die Relevanz einer Empfehlung auswirken kann. Sollte sich beispielsweise herausstellen, dass die Ergebnisse so gestreut sind, dass in jedem Fall geblättert werden muss, kann die letztliche Realisierung dies entweder durch ein größeres Fenster für die Vorschläge umsetzen oder durch eine Liste, in der in beliebigen Schritten gescrollt, statt geblättert werden kann.
Einfache nicht-lineare Funktionen, die man üblicherweise zu einer solchen Auswertung heranziehen könnte, sind im folgenden Plot dargestellt. Im Vergleich zur roten, linearen Funktion sind zwar besonders hohe Werte für Platzierungen jenseits einer bestimmten Grenze zu erkennen, der Unterschied zwischen zwei recht hohen Positionen (z.B. 21 und 22) kann beispielsweise bei $(x \cdot 1{,}1^x)$ als enorm bezeichnet werden. Diese starke Unterscheidung ist nicht beabsichtigt und bildet kein gewünschstes Ergebnis.

Kombination linearer und nicht-linearer Eigenschaften

Gesucht war also eine Funktion, die einerseits die Gruppierung der Ergebnisse berücksichtigt, und andererseits innerhalb dieser Gruppen noch eine differenziertere Betrachtung möglich macht, statt alle x Ergebnisse einer „Seite“ zusammenzufassen.
Hierzu wurde von der PG eine Funktion entwickelt, die die Gruppierung berücksichtigt und innerhalb der Seiten dann noch eine weitere Unterscheidung macht.

[13] Die Implementierung könnte natürlich auch in ständiger Wechselbeziehung zur Evaluation entwickelt werden.

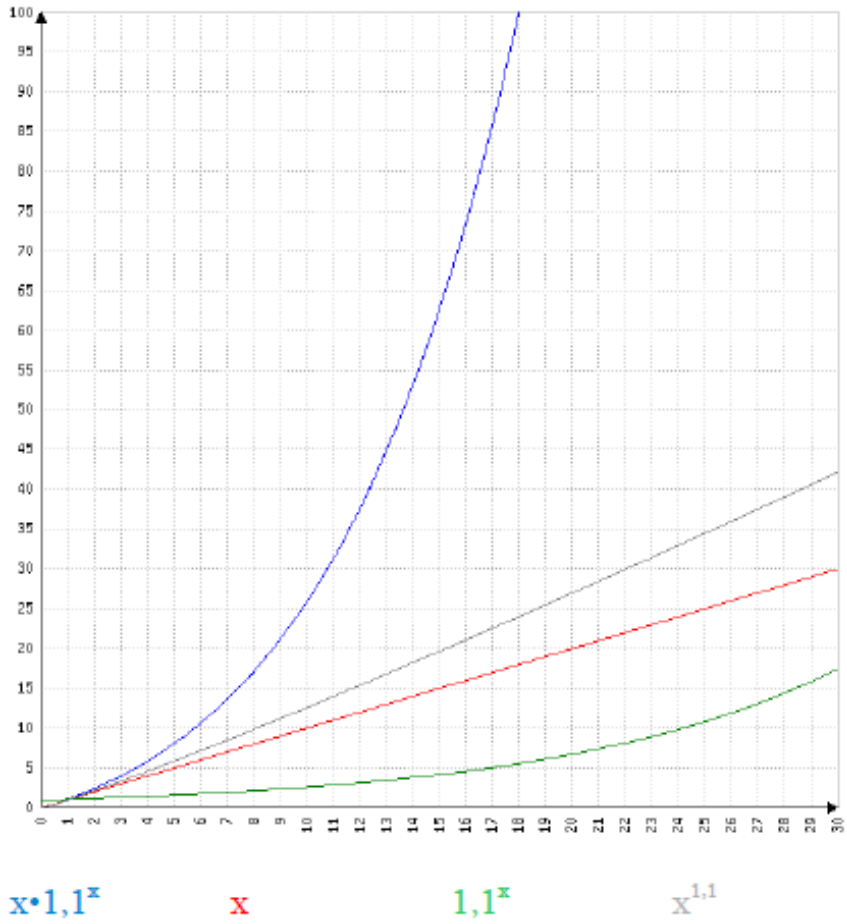


Abbildung 5.2: Mögliche Straffunktionen im Vergleich

Auch wenn einige Informationen zur bevorzugten Partitionierung von Informationen beim Menschen vorliegen[14], ist hierbei die letztliche Gruppengröße offen gehalten worden und es wurde eine allgemeine Formel für eine passende Funktion f entworfen:

$$f(x) = x \div g + \sqrt[c]{1/2} \cdot \frac{x \bmod g}{g-1}.$$

Hierbei bezeichnet g die gewählte Gruppengröße und der Parameter c beeinflusst die Steigung innerhalb der Gruppen.
In jedem Fall wird durch diese Formel erreicht, dass der Unterschied zwischen dem kleinsten und größten Wert innerhalb einer Gruppe, genauso bewertet wird, wie der zwischen dem letzten/größten innerhalb einer und dem ersten/kleinsten der nächsten Gruppe.
Somit bezieht die Funktion sowohl die Steigung innerhalb der Gruppe als auch Unterschiede zwischen den Gruppen ein.

[14] Baddley und Hitch, Modell des Arbeitsgedächtnisses – Chunks/Gedächtnis

Semantische Beurteilung

Wie bereits erwähnt, ergibt sich ein großes Problem dadurch, dass sich die tatsächliche Zufriedenheit eines Kunden nicht einfach mathematisch abbilden lässt. Ließen sich die Wünsche eines Nutzers als Funktion darstellen, könnten diese Funktionen schon zur Berechnung der Empfehlungen verwendet werden und müssten kaum noch über ihre angedachte Funktionalität hinaus evaluiert werden.
Stattdessen drängt sich, wie immer in Bereichen mit eher „weichen“ Kriterien eine Auswertung „per Hand“, oder vielmehr „per Mensch“ auf. Ideal wäre ein Test durch regelmäßige Nutzung der in dieser Anwendung vom Recommender bearbeiteten Datengrundlage. Durch die Nutzung könnte dann nicht nur die Nutzbarkeit der Empfehlungen in der Anwendung beurteilt werden, sondern auch die Erfahrungen der Nutzer bei der Beurteilung unerwarteter Ergebnisse mit einfließen lassen.
In Ermangelung der Möglichkeit, diese sämtlichen Ergebnisse überprüfen zu lassen, wurde überlegt, als „nicht erfolgreich“ beurteilte Empfehlungen des Recommenders noch einmal einzeln zu überprüfen. Eine zumindest grobe Einteilung der tatsächlich an erster Stelle empfohlenen Services dürfte anhand der textuellen Beschreibungen möglich sein, sodass auch hier mit einer einfachen Bewertungsfunktion Fragebögen erstellt und umgesetzt werden können, mit deren Hilfe die entsprechenden Empfehlungen manuell evaluiert werden. Damit wäre eine differenziertere Beurteilung der Stärken und Schwächen des Recommenders möglich.
Zunächst wurden hierzu die ausgelesenen Daten in eine leicht lesbare Form gebracht; somit müssen Tester nicht sämtliche Services tatsächlich kennen, sondern nur anhand ihrer Beschreibung beurteilen, ob eine entsprechende Verarbeitung der Daten sinnvoll wäre.
Im Laufe der Entwicklung ist jedoch von der Möglichkeit die Empfehlungen manuell von Bioinformatikern oder Biologen überprüfen zu lassen Abstand genommen worden. Der Aufwand auf beiden Seiten ist hierbei zu groß. Die Erstellung der Fragebögen mit deren Hilfe eine manuelle Evaluation stattfinden kann und die Einarbeitung der Testpersonen rechtfertig den Aufwand nicht.

Implementierung

Bei der Implementierung der Evaluation ist auf die Erweiterbarkeit geachtet worden. Wie zuvor beschrieben gibt es sehr viele Möglichkeiten die Sinnhaftigkeit von Empfehlungen zu beurteilen. Des Weiteren gibt es überhaupt viele Möglichkeiten zu einer Empfehlung zu gelangen. Denn ein Workflow kann beliebig viele Informationen enthalten bevor eine Empfehlung in der Evaluation verlangt wird. Alle diese Ansätze sollen in der Evaluation berücksichtigt werden.
Der Kern der Evaluation ist die Klasse `EvaluationSystem`. In dieser Klasse werden die verschiedenen Bewertungsfunktionen, Evaluationsmethoden und Trainings- und Testmengengeneratoren zusammengeführt. Hierbei werden alle Kombinationen durchlaufen und in jeder dieser Kombinationen werden Emfehlungen generiert und diese anschließend mit der Bewertungsfunktion entsprechend bewertet.
Bei den Trainings- und Testmengengeneratoren sind ist die k-fache Kreuzvalidierung sowie das Leave-one-out-Verfahren implementiert worden. Da zu Beginn der Imple-

mentierung noch nicht abzusehen war, auf wie vielen Workflows die Evaluation die Recommender bewertet ist wurde es als sinnvoll erachtet worden eine Methode zu haben, welche auf großen Datenmengen repräsentative Ergebnisse liefert und eine welche auf kleineren Datenmengen sehr genaue Ergebnisse liefert.
Auch bei den Bewertungsfunktionen werden zwei der zuvor beschriebenen Ansätze umgesetzt. Zum einen existiert eine lineare Bewertungsfunktion zum anderen eine Kombination aus linearer und nicht-linearer Bewertungsfunktion. Diese Bewertungsfunktionen werden auf jede Empfehlung angewandt. Hierbei ist zu beachten, dass es pro Trainings- und Testmenge mehrere Empfehlungen gibt.
Bei der Implementierung werden mithilfe der Trainings- und Testmengengeneratoren unterschiedliche Szenarien erzeugt. Diese Szenarien bestehen immer aus Workflows, die mithilfe von Empfehlungen rekonstruiert werden sollen und eine Menge von Workflows, welche als Datengrundlage dienen. Um sicherzustellen, dass die Workflows welche in der Testmenge enthalten sind nicht zum Generieren von Empfehlungen genutzt werden, wurden die Services und Workflows um ein Attribut *hidden* erweitert. Dieses Attribut kann entweder den Wert wahr oder falsch annehmen. Die Recommender erhalten nur Zugriff auf Services und Workflows, bei denen das Attibut auf false gesetzt ist. Hierdurch wird schon auf Datenbankebene sichergestellt, dass die Trainings- und Testmengen tatsächlich voneinander getrennt sind. Eine solche Trennung ist sehr wichtig, da sonst ein Workflow, welcher rekonstruiert werden soll in der Datenbank enthalten sein könnte. Somit wäre die Empfehlungen sehr einfach und das Ergebnis der Evaluation verfälscht. Ein weiterer Punkt, an dem das Attribut eingesetzt wird ist die Änderung von Profilen. In den Profilen kann die Menge der zu verwendenden Importmodule bestimmt werden. Damit die Recommender immer das gleiche Verhalten bei einer bestimmten Auswahl solcher Importmodule vorweisen, werden alle Services und Workflows aus nicht verwendeten Importmodulen auf *hidden* gesetzt. Sonst könnte es passieren, dass die Recommender auf Services und Workflows zurückgreifen, die sie bei der gewählten Konfiguration von Importmodulen gar nicht kennen dürften. Insbesondere wird hierdurch verhindert, dass diese Services und Workflows aus der Datenbank gelöscht werden müssen und somit bei der erneuten Hinzunahme des Importmoduls neu importiert werden müssten.
Die Ergebnisse werden durch Instanzen der Klasse `EvaluationResult` repräsentiert. In dieser Klasse sind die Ergebnisse der Bewertungsfunktionen gespeichert sowie weitere relevante Informationen wie: Verwendeter Trainings- und Testmengengenerator, verwendete Bewertungsfunktion, verwendetes Profil sowie Informationen über die Services und Workflows. Mithilfe dieser Ergebnisse lassen sich alle Relevanten Informationen herausarbeiten und präsentieren.
Wie schon zuvor erwähnt ist bei der Implementierung auf die Erweiterbarkeit geachtet worden, so stehen für die Trainings- und Testmengengeneratoren sowie die Bewertungsfunktionen abstrakte Klassen zur Verfügung. Klassen, die hiervon erben können einfach in das `EvaluationSystem` eingebunden werden und somit in der Evaluation genutzt werden.

Ergebnisse der Evaluation

Bei der Durchführung der Evaluation ist ausschließlich das Leave-one-out-Verfahren verwendet worden. Aufgrund der Anzahl von Workflows in der Datenbank ist die Laufzeit mit einem durchschnittlichen Wert von neun Minuten annehmbar. Die 10-fache-Kreuzvalidierung hätte nur zu ungenaueren Ergebnissen geführt. Sie bietet sich jedoch für die Evaluation des Systems auf größeren Datenbeständen an.
Insgesamt sind bei der Evaluation 30384 Empfehlungen durchgeführt worden. Eine Empfehlung wird als Sinnvoll beachtet, wenn der durch die Bewertungsfunktion gegebene Wert einer Empfehlung unter einem gewissen Grenzwert liegt. Dieser Grenzwert liegt bei dieser Evaluation bei 100. Dieser Wert erscheint auf den ersten Blick als zu groß, da kaum ein Nutzer eine Menge in dieser Größenordnung nach den passenden Service durchsuchen möchte. Aus diesem Grund ist zudem die Anzahl der Empfehlungen angegeben, die der Nutzer in der Oberfläche sofort angezeigt bekommt. In diesem Fall handelt es sich um jeweils sieben Services. Die Anzahl der Empfehlungen, die den „gewünschten“ Service unter den ersten sieben Services der Empfehlung enthalten sind somit auch in der Praxis als sehr gut anzusehen.
Im weiteren Verlauf werden zwei Ergebnisse vorgestellt. Das erste Ergebnis stellt die Kombination linearer und nicht-linearer Straffunktionen dar. Das zweite Ergebnis ist die lineare Straffunktion. Es ist leicht zu sehen, dass das Ergebnis der linearen Straffunktion besser erscheint, dies liegt daran, dass „gewünschte“ Services, die nicht unter den ersten sieben Services der Empfehlung auftauchen nicht so stark bestraft werden. Bei der linearen Funktion wird zudem immer das Minimum betrachtet. Es zählt also die kleinste Position auf der ein „gewünschter“ Service auftaucht.

Ergebnis kombinierte Straffunktion. Die Evaluation wurde für die kombinierte Straffunktion mehrmals durchgeführt. Jedes Mal ist eine andere Kombination von aktiven Recommenderalgorithmen gewählt worden. Hierdurch soll auch das Verhalten und die Auswirkungen einzelner Recommenderalgorithmen auf das Gesamtergebnis verdeutlicht werden. In Tabelle 5.7 sind die Ergebnisse der Evaluation dargestellt, wenn alle Recommenderalgorithmen aktiv und gleichgewichtet sind.

Durchschnittliche Bewertung	$18,876687569803217$
$Q_{0,25}$	$3,0$
$Q_{0,75}$	$25,0$
Median	7.666666666666667

Tabelle 5.7: Kombinierte Straffunktion - Alle Algorithmen gleichgewichtet

Das Ergebnis verdeutlicht was schon bei der Nutzung des Recommenders aufgefallen ist: Die Empfehlungen sind häufig sehr gut und passend. Immerhin finden sich bei einem Viertel aller Empfehlungen die „gewünschten“ Services unter den ersten drei Empfehlungen. Das der Durchschnitt der Empfehlungen so hoch liegt, auch im Vergleich zum Median lässt darauf schließen, dass es einige Empfehlungen gibt, die sehr schlecht bezüglich der Straffunktion abgeschnitten haben. Dies kann vorkommen, wenn Workflows rekonstruiert werden, welche nur einmal in der Datenbank

enthalten sind und die Empfehlungsalgorithmen auf andere Workflows zurückgreifen müssen. Auch die Tatsache, dass das obere Quantil bei 25 liegt verdeutlicht dies. Anschließend ist die Evaluation für alle Algorithmen einzeln durchgeführt worden. Hierbei ist aufgefallen, dass das Mutual Information-Verfahren mit Abstand die besten Ergebnisse liefert. In Tabelle 5.8 sind diese dargestellt.

Durchschnittliche Bewertung	$17,259634776392904$
$Q_{0,25}$	$3,0$
$Q_{0,75}$	$23,151515151515152$
Median	$8,0$

Tabelle 5.8: Kombinierte Straffunktion - Mutual Information

Es fällt hierbei besonders auf, dass der Algorithmus in fast jeden Punkt besser oder genauso gut ist wie die Kombination aller Algorithmen zusammen. Der Median ist jedoch höher, was darauf schließen lässt, dass die anderen Algorithmen einige qualitativ hochwertige Vorschläge zur Empfehlung beigesteuert haben.
Dieses Resultat führt direkt zur dritten Auswertung. Bei dieser Evaluation gehen alle Algorithmen bis auf Mutual Information nur noch mit dem halben Gewicht ein. Die Ergebnisse der Evaluation sind in Tabelle 5.9 dargestellt.

Durchschnittliche Bewertung	$19,22417777890612$
$Q_{0,25}$	$2,5$
$Q_{0,75}$	$24,5$
Median	$7,666666666666667$

Tabelle 5.9: Kombinierte Straffunktion - Alle Algorithmen unterschiedlich gewichtet

Hierbei Fällt auf, dass die Ergebnisse im Durchschnitt schlechter sind als zuvor, der Median jedoch wieder kleiner ist. Was darauf schließen lässt, dass einige schlechte Ergebnisse hinzugekommen sind. Diese haben jedoch nur eine geringe Auswirkung für den Nutzer. Das untere Quantil ist kleiner als bei den beiden Szenarien zuvor, was insbesondere für diese Konfiguration spricht.

Lineare Straffunktion. Wie bei der kombinierten Straffunktion sind auch bei der linearen Straffunktion mehrere Evaluationen für unterschiedliche Konfigurationen der Algorithmen durchgeführt worden. Die lineare Funktion ist hierbei noch einfacher zu interpretieren als die kombinierte Funktion, da der Funktionswert genau der Position in der Empfehlung entspricht. Auch hier gilt: Je kleiner desto besser. Insgesamt bietet die implementierte Straffunktion drei Modi: *Minimum*, *Maximum* und *Durchschnitt*. Je nach Modus ist der Beste, Schlechteste bzw. der Durchschnitt aller Werte der Funktionswert. Im Folgenden wird die lineare Straffunktion im Modus *Minimum* betrachtet.
Zunächst ist das Verhalten des Recommenders untersucht worden, wenn alle Algorithmen aktiv und gleichgewichtet sind. Die Ergebnisse sind in Tabelle 5.10 dargestellt.

Durchschnittliche Bewertung	21,927922590837284
$Q_{0,25}$	1,0
$Q_{0,75}$	18,0
Median	3,0
Unter den besten sieben	20191

Tabelle 5.10: Lineare Straffunktion - Alle Algorithmen gleichgewichtet

Hierbei fällt auf, dass das untere Quantil 1 ist. Bei 25% der Empfehlungen befindet sich das gewünschte Ergebnis somit an der ersten Stelle der Liste aller Empfehlungen. Auch hier ist wie zu erwarten der Durchschnitt deutlich höher als der Median. Somit gibt es einige sehr schlecht Empfehlungen. Doch 75 % aller Empfehlungen sind unter den ersten 18. Somit ist die Qualität der Empfehlungen als sehr hoch einzuschätzen. Wie zu erwarten ist auch bei der linearen Straffunktion das Mutual Inforation-Verfahren am besten. In Tabelle 5.11 sind die Ergebnisse der Evaluation dieses Verfahrens dargestellt.

Durchschnittliche Bewertung	20,868253027909425
$Q_{0,25}$	1,0
$Q_{0,75}$	17,0
Median	3,0
Unter den besten sieben	19974

Tabelle 5.11: Lineare Straffunktion - Mutual Information

Es fällt jedoch besonders auf, das sich nicht so viele Ergebnisse unter den ersten sieben Empfehlungen befinden wie bei der Konfiguration wenn alle Algorithmen gleich gewichtet sind. Dieses Ergebnis führt zu dem Schluss, dass die anderen Algorithmen noch Empfehlungen sehr hoher Qualität geben, jedoch auch viele schlechte.
Um diese Eigenschaften zu verbinden ist eine Evaluation mit einer Konfiguration durchgeführt worden, bei der alle Algorithmen aktiv sind jedoch bis auf das Mutual Information-Verfahren nur mit halben Gewicht eingehen. Die Ergebnisse dieser Evaluation sind in Tabelle 5.12 dargestellt.

Durchschnittliche Bewertung	21,071649552395996
$Q_{0,25}$	1,0
$Q_{0,75}$	17,0
Median	2,0
Unter den besten sieben	20520

Tabelle 5.12: Lineare Straffunktion - Alle Algorithmen unterschiedlich gewichtet

Im Durchschnitt sind die Ergebnisse zwar ein wenig schlechter als die vorherigen Ergebnisse, es finden sich jedoch 546 „gewünschte“ Services unter den ersten sieben Empfehlungen. Dies ist besonders relevant, da der Nutzer diese zuerst zu sehen bekommt. Auch der Median ist um eine Position besser.

Interpretation. Die Ergebnisse der Evaluation zeigen, dass die Empfehlungen des Recommender-Systems in vielen Fällen sehr gut sind. Es lässt sich nicht abstreiten, dass es Ausnahmen gibt, bei denen die Empfehlungen sehr schlecht sind. Dies liegt aber zum Teil an der Datengrundlage für die Empfehlungen. Alles in allem haben die Empfehlungen jedoch eine sehr gute Qualität, wenn auch einige Verfahren, auf den vorliegenden Daten, besser funktionieren als andere.
Zusammenfassend kann die Evaluation als Bestätigung der praktischen Ergebnisse gesehen werden. Die Empfehlungen des Recommender-Systems sind sehr häufig sinnvoll und zielführend, es gibt jedoch Situationen, in denen das System kaum oder keine guten Empfehlungen ausgibt. Insbesondere muss auf die Konfiguration der Algorithmen geachtet werden, da diese die Qualität sehr stark beeinflussen können.

6 Schluss

6.1 Fazit

Im Rahmen der Projektgruppe *Situation-Aware Semantic Service Discovery* im Sommersemester 2012 und Wintersemester 2012/2013 wurde ein Recommender-System für Services entwickelt und als Erweiterung für die Workflowmanagementumgebung *jABC* implementiert.

EMF-Metamodell. Es wurde ein EMF-Metamodell für Services entwickelt, das Informationen zur Untersuchung semantischer Zusammenhänge zwischen Services enthält. Zu diesen Informationen gehören syntaktische Informationen (Ein- und Ausgabeparameter), semantische Informationen (Beschreibung, Zuordnungen zu Kategorien und Tags) sowie kollaborative Informationen (Benutzung in Workflows, Benutzerbewertung, Popularität).
Angesichts des einfachen Service-Modells und der geringen gespeicherten Datenmenge erwies sich die Wahl von *EMF* zur Modellierung und *Hibernate* zur Wahrung der Persistenz als unangemessen. Die automatische Code-Erzeugung zur Instanziierung von Modellen, die zu den Vorteilen von *EMF* gehört, führte an einigen Stellen zu unnötig hoher Komplexität. Die Persistenz von EMF-Modell-Instanzen mittels *Hibernate* führte ebenfalls zu unnötigen Kompatibilitätsproblemen.

Plug-in-Architektur. Ausgehend vom Metamodell wurden verschiedene Methoden der Theorie der Empfehlungssysteme sowie eigene Ansätze verfolgt, um für jeden verfolgten Ansatz einen Recommender zu entwickeln. Von einer zentralen Kernkomponente ausgehend wurde ein möglichst modularer Ansatz verfolgt. Die einzelnen Recommender wurden als unabhängige Plug-ins für die Kernkomponente konzipiert, sodass sich in der Implementierung gewichtete Kombinationen der einzelnen Recom-

mender auswählen lassen. Der modulare Aufbau erlaubt die geplante Nutzung des *OSGi*-Frameworks zur dynamischen Einbindung einzelner Plug-ins.
Unter anderem aufgrund der im Zusammenhang mit dem Metamodell entstandenen Probleme bei der Realisierung des Systems als *jABC*-Erweiterung musste letztlich auf die Verwendung von *OSGi* verzichtet werden.

Ein Nachteil des modularen Aufbaus des Recommender-Systems ist die völlige Unabhängigkeit einzelner Recommender untereinander bezüglich der zusammengesetzten Empfehlung. Einzelne Recommender arbeiten mit unterschiedlichen Informationen über vorhandene Services. So kann es passieren, dass aufgrund mangelhafter Informationen mehrere Recommender einander widersprechen. Aus der Sicht der Kernkomponente lassen sich in diesem Fall die Gründe nicht nachvollziehen. So bleibt ihr keine andere Wahl, als eine gewichtete Summe widersprüchlicher Ergebnisse als Empfehlung zurückzugeben.

Datenquellen. Zur Nutzung der entwickelten Recommender werden Daten aus verschiedenen Datenquellen importiert und durch die Kernkomponente verwaltet. Der Import und die Verwaltung der Daten geschieht wiederum durch voneinander unabhängige Plug-ins. Die verwendeten Datenquellen (*EMBOSS*, *BioCatalogue*, *MyExperiment*) haben sich als gut bezüglich syntaktischer, ausreichend bezüglich semantischer und mangelhaft bezüglich kollaborativer Informationen erwiesen. In den meisten Fällen kollaborieren einzelne Benutzer der Portale kaum und verwenden eigene Tags und Services innerhalb ihrer Workflows. Dies erschwert die Aufdeckung von Zusammenhängen zwischen Services.

Integration. Die Entscheidung gegen eine geplante Implementierung des Recommender-Systems in *Eclipse4Bio* zugunsten von *jABC* führte bis auf die beschriebene Problematik mit *OSGi* zu keinen Nachteilen. Die Abbildung von Services auf SIBs sowie die Nutzung der Plug-in-Schnittstelle war gut für die Integration geeignet. Dass *jABC* Workflows kontrollflussorientiert verarbeitet, aber vorhandene Datenquellen datenflussorientierte Workflows zur Verfügung stellen, führte zu keinen besonderen Schwierigkeiten.
Da die Gewichtung einzelner Recommender bedeutend ist, wurde eine grafische Benutzeroberfläche zur Konfiguration der Gewichtung und benutzter Datenquellen eingerichtet.

Evaluation. Eine Evaluation des Recommender-Systems im Kontext vorhandener Workflows aus *MyExperiment* mit dem *Leave-One-Out*-Verfahren zeigte, dass auf der Grundlage gegebener Daten erzeugte Empfehlungen sehr häufig dem Benutzerverhalten entsprechen und zielführend sein können. Nichtsdestotrotz sind in machen Fällen erzeugte Empfehlungen aufgrund mangelhafter Daten sinnlos.

Insgesamt lässt sich sagen, dass die für die Projektgruppe gesetzten Ziele – bis auf Änderungen an Detailanforderungen – erreicht wurden, und dass es möglich ist, ein sinnvoll funktionierendes Recommender-System für Services der Bioinformatik mit zur Verfügung stehenden Daten zu entwickeln.

6.2 Ausblick

Verbesserung der Unifikation von Services. Die Qualität des Datenbestandes und damit auch der Empfehlungen ist stark vom Unifikationsvorgang von Services abhängig. Eine Verbesserung der Unifikation von Services würde einerseits zu weniger Empfehlungen redundanter Services und andererseits zu einer besseren Analyse von Workflows führen. Eine vielversprechende Möglichkeit besteht darin, eine Relation aufzubauen, die Informationen über neuere Versionen von Services enthält. So könnten nicht nur verschiedene Versionen eines Services unifiziert werden, sondern auch nur die neueste Version eines Services empfohlen werden.

Neue Datenquellen. Kollaborative Daten werden bei konventionellen Empfehlungssystemen oft für sinnvolle Empfehlungen verwendet. Diese Daten müssen allerdings in gewisser Quantität vorhanden sein. Bei den verwendeten Datenquellen ist dies größtenteils nicht der Fall. Es wäre naheliegend, andere Datenquellen zu suchen oder womöglich neue Datenquellen zu generieren. So wäre es sinnvoll, Benutzerstatistiken einer Workflowmanagementumgebung für zukünftige lokale Empfehlungen zu sammeln und an eine zentrale Datenbank zu versenden, um global kollaborative Daten zu erzeugen und zu nutzen.

Verfeinerung vorhandener Datenquellen. Um der unzureichenden Qualität vorhandener Datenquellen entgegenzuwirken, könnten diese manuell für den Einsatz im Recommender-System vorbereitet werden. Zum einen kann die Gestaltung von Tags vereinheitlicht werden, sodass mehr semantische Beziehungen zwischen Services aufgedeckt werden können. Zum anderen würde eine größere Aktivität hinsichtlich Benutzerbewertungen von Services unmittelbar dazu führen, dass relevante Services eher empfohlen werden.

Neue Recommender. Die Implementierung anderer Recommender als Plug-ins für das Recommender-System würde die Qualität der Empfehlungen weiterhin steigern. Es ist zum Beispiel vorstellbar, Recommender auf Basis von Ontologien zu konzipieren.

Zusammenarbeit der Recommender. Eine weitere Verbesserung des Recommender-Systems besteht darin, dass einzelne Recommender feiner zusammenarbeiten. In Fall, dass mindestens ein Recommender aufgrund von „qualitätsvollen“ Daten davon überzeugt ist, einen relevanten Service gefunden zu haben, sollte seine Empfehlung nicht durch unvollständige Daten anderer Recommender relativiert werden. Dazu müssen der Begriff „qualitätsvoll“ näher definiert und insbesondere die Datenquellen analysiert werden. Eine genauere Analyse benutzter Datenquel-

len könnte durchaus dazu führen, dass oft genutzte lokale Skripte als eigenständige Elemente empfohlen werden können.

Ausführbarkeit. Die Ausführbarkeit der Workflows aus empfohlenen Services in *jABC* ist nicht gegeben. Da verschiedene Services aus verschiedenen Datenquellen unterschiedlich spezifiziert und lokalisiert sind, ist dies im Allgemeinen für beliebige Datenquellen nicht möglich. Es ist nicht unwahrscheinlich, dass bei geeigneter Einschränkung, etwa auf *EMBOSS*-Services, Skripte und Kontrollflusselemente die Ausführbarkeit eines erzeugten Workflows in *jABC* ohne sehr großen Zusatzaufwand verwirklicht werden könnte.

Vision. Ein verlässliches Recommender-System benötigt umfangreiche und gepflegte Datenquellen. Die folgenden Schritte würden genügen, um eine gute Datenqualität zu erreichen.

1. Den Grad an Dokumentation und Konsistenz der *EMBOSS*-Services auf *BioCatalogue*-Services übertragen.

2. Services aus *MyExperiment*-Workflows mit der so entstandenen Datenbank verknüpfen.

3. Eine aktive Community aufbauen, die Services benutzt und bewertet.

Abschließend lässt sich sagen, dass relevante Daten bereits in einem genügend großen Umfang vorliegen, jedoch ist deren Nutzbarkeit durch mangelnde Konsistenz und Pflege schwierig.

A Anhang

A.1 Copyright – Entscheidung für eine Lizenz

Bereits zu Beginn des ersten Semesters wurde in einer kurzen Diskussion die Frage zum Copyright aufgebracht. Lange herrschte dabei Uneinigkeit über die zu verwendenden Lizenzen, wobei ein Großteil der Gruppe für eine möglichst freigebige Lizenz stimmte. Nach einiger Recherche und der Vervollständigung des Projekts ist die Entscheidung auf die *BSD 2-Clause License* gefallen:


```
Copyright (c) 2013, PG-SASSD, TU Dortmund
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
```

```
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

Die Entscheidung für diese Lizenz ist aus zweierlei Gründen gefallen: Einerseits erfüllt sie die Anforderung nach minimalen Restriktionen, wie sie in der Gruppe aufgestellt wurde. Andererseits ist sie kompatibel zu den jeweiligen Copyrighterklärungen und Lizenzen der von uns verwendeten Komponenten, für deren direkte Integration im Projekt wir uns entschieden haben.
Es folgt eine Übersicht über die innerhalb des Projekts als Bibliotheken verwendeten Komponenten, jeweils mit einem Vermerk zur Lizenz der Komponente und ihrem Status im Projekt.

Nicht enthaltene Komponenten

- **Java Development Kit:**
  `http://www.oracle.com/technetwork/java/index.html`
  Lizenz: *Sun License*, kann aufgrund der Lizenz nicht im Projekt enthalten sein, sondern muss manuell installiert werden

- **EMBOSS:** `http://emboss.sourceforge.net/`
  Lizenz: *GPL*, kann optional manuell installiert werden und ist aufgrund der Lizenz nicht im Projekt enthalten

- **jABC:** `http://www.jabc.de/`
  Lizenz: Beinhaltet sowohl *Apache2*, *BSD*, *LGPL* als auch eine eigene Lizenzvereinbarung; muss manuell installiert werden

Variabel enthalten

- **Eclipse Modeling Framework (EMF):**
  Lizenz: *Eclipse Public License* (*EPL*), kompatibel zur verwendeten Lizenz; kann manuell installiert werden, sofern nicht im Projekt enthalten

Enthaltene Komponenten

- **Apache Commons Komponenten:** `http://commons.apache.org/`
  Lizenz: *Apache2*, kompatibel zur verwendeten Lizenz und ist im Projekt unverändert enthalten

- **jFreeChart:** `http://www.jfree.org/jfreechart/`
  Lizenz: *LGPL*, kompatibel zur verwendeten Lizenz und ist im Projekt unverändert enthalten

- **jGraph:** `http://www.jgraph.com/`
  Lizenz: *BSD*, kompatibel zur verwendeten Lizenz und ist im Projekt unverändert enthalten
- **HSQLDB:** `http://www.hsqldb.org/`
  Lizenz: *BSD*, kompatibel zur verwendeten Lizenz und ist im Projekt unverändert enthalten
- **Hibernate:** `http://www.hibernate.org/`
  Lizenz: *LGPL 2.1*, kompatibel zur verwendeten Lizenz und ist im Projekt unverändert enthalten
- **DesignGridLayout:** `http://designgridlayout.java.net/`
  Lizenz: *Apache2*, kompatibel zur verwendeten Lizenz und ist im Projekt unverändert enthalten
- **ACDFile parser:** `http://moby.ucalgary.ca/seahawk/`
  Lizenz: *LGPL 3*, kompatibel zur verwendeten Lizenz und ist im Projekt unverändert enthalten

A.2 CategoryFilter - Tests

EmbossACD_descseq
Categories:
edit

Recommended services:

EmbossACD_aligncopy
EmbossACD_aligncopypair
EmbossACD_biosed
EmbossACD_cutseq EmbossACD_degapseq
EmbossACD_domtesta
EmbossACD_domtestb
EmbossACD_domtestc
EmbossACD_domtestd
EmbossACD_entrygo
EmbossACD_entrytax
EmbossACD_extractseq
EmbossACD_featreport
EmbossACD_feattext
...

EmbossACD_maskseq
Categories:
edit

Recommended services:

EmbossACD_aligncopy
EmbossACD_aligncopypair
EmbossACD_biosed
EmbossACD_cutseq
EmbossACD_degapseq
EmbossACD_descseq
EmbossACD_domtesta
EmbossACD_domtestb
EmbossACD_domtestc
EmbossACD_domtestd
EmbossACD_entrygo
EmbossACD_entrytax
EmbossACD_extractseq
EmbossACD_featreport
EmbossACD_feattext
...

EmbossACD_wordcount
Categories:
composition

Recommended services:

EmbossACD_compseq
EmbossACD_freak
EmbossACD_chaos
EmbossACD_complex
EmbossACD_density
EmbossACD_emowse
EmbossACD_isochore
EmbossACD_mwcontam
EmbossACD_mwfilter
EmbossACD_oddcomp
EmbossACD_pepdigest
EmbossACD_backtranambig
EmbossACD_backtranseq
EmbossACD_dan
EmbossACD_pepstats

EmbossACD_banana
EmbossACD_btwisted
EmbossACD_pepinfo

EmbossACD_freak
Categories:
composition

Recommended services:

EmbossACD_compseq
EmbossACD_wordcount
EmbossACD_chaos
EmbossACD_complex
EmbossACD_density
EmbossACD_emowse
EmbossACD_isochore
EmbossACD_mwcontam
EmbossACD_mwfilter
EmbossACD_oddcomp
EmbossACD_pepdigest
EmbossACD_backtranambig
EmbossACD_backtranseq
EmbossACD_dan
EmbossACD_pepstats
EmbossACD_banana
EmbossACD_btwisted
EmbossACD_pepinfo

EmbossACD_digest
Categories:
motifs

Recommended services:

EmbossACD_antigenic
EmbossACD_epestfind
EmbossACD_fuzzpro
EmbossACD_patmatdb
EmbossACD_patmatmotifs

EmbossACD_preg
EmbossACD_pscan
EmbossACD_fuzztran
EmbossACD_sigcleave

BioCatalogue_EMBOSS_hmoment
Categories:
protein_secondary_structure

Recommended services:

BioCatalogue_EMBOSS_garnier
BioCatalogue_EMBOSS_pepcoil
BioCatalogue_GOR_III_protein_secondary_structure_prediction__CNRS_IBCP_
BioCatalogue_GOR_I_protein_secondary_structure_prediction__CNRS_IBCP_
BioCatalogue_INB_mmb_pcb_ub_es_runPHDFromBLASTText
BioCatalogue_passtaImplementationService
BioCatalogue_pepnetService
BioCatalogue_pepwheelService
BioCatalogue_plotFeatureAASequenceService_
BioCatalogue_EMBOSS_tmap
BioCatalogue_EMBOSS_topo
BioCatalogue_EMBOSS_helixturnhelix
BioCatalogue_Phobius__REST_
BioCatalogue_Phobius__SOAP_
BioCatalogue_GenesilicoMetaServerSOAPService
BioCatalogue_WSPhobius

EmbossACD_hmoment
Categories:
properties

Recommended services:

EmbossACD_charge
EmbossACD_octanol
EmbossACD_pepwindow
EmbossACD_pepwindowall
EmbossACD_iep
EmbossACD_pepstats
EmbossACD_pepinfo

EmbossACD_dottup
Categories:
dot_plots

Recommended services:

EmbossACD_dotmatcher
EmbossACD_dotpath
EmbossACD_polydot
EmbossACD_extractalign

BioCatalogue_TreeBASE
Categories:
phylogeny
phylogeny:evolutionary_distance_measurements
phylogeny:tree_inference
phylogeny:tree_display
phylogeny:statistical_robustness
phylogeny:molecular_sequence

Recommended services:

BioCatalogue_ClustalW2_Phylogeny__REST_
BioCatalogue_ClustalW2_Phylogeny__SOAP_
BioCatalogue_PhylogeneticnewService
BioCatalogue_rociImplementationService
BioCatalogue_roseImplementationService

EmbossACD_distmat
Categories:
phylogeny:molecular_sequence

Recommended services:

BioCatalogue_ClustalW2_Phylogeny__REST_
BioCatalogue_ClustalW2_Phylogeny__SOAP_
BioCatalogue_PhylogeneticnewService
BioCatalogue_TreeBASE
BioCatalogue_rociImplementationService

Literatur

[55812] Projekgruppe 558. *PG-Endbericht: Eclipse4Bio*. PDF-Dokument. Online verfügbar unter `http://hdl.handle.net/2003/29847` (besucht am 25. April 2013). 2012.

[AL08] E. Alpaydin und S. Linke. *Maschinelles Lernen*. Oldenbourg, 2008.

[Bad08] Youakim Badr. „Service-Oriented Workflow“. In: *JDIM* 6.1 (2008), S. 120–127.

[Bel08] Michael Bell. *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Wiley Publishing, 2008.

[Bioa] *BioCatalogue*. Webseite. Online verfügbar unter `www.biocatalogue.org` (besucht am 25. April 2013). 2012.

[Biob] *BioPortal*. Webseite. Online verfügbar unter `bioportal.bioontology.org` (besucht am 25. April 2013). 2012.

[Ecla] *Eclipse4Bio*. URL: `https://projekte.itmc.tu-dortmund.de/projects/eclipse4bio`.

[Eclb] *EclipseLink Project*. URL: `http://www.eclipse.org/eclipselink/`.

[Eda] *EDAM Ontology*. Webseite. 2012. URL: `edamontology.sourceforge.net`.

[Emb] *EMBOSS*. Webseite. Online verfügbar unter `emboss.sourceforge.net` (besucht am 25. April 2013). 2012.

[Emf] *Eclipse Modeling Framework Project (EMF)*. URL: `http://www.eclipse.org/modeling/emf/`.

[Emp] *Empfehlungsysteme aus informations- wissenschaftlicher Sicht- State of the Art*. Webseite. Online verfügbar unter `www.phil-fak.uni-duesseldorf.de/fileadmin/Redaktion/Institute/Informationswissenschaft/forschung/information_retrieval/1189509550empfehlung.pdf` (besucht am 25. April 2013). 2012.

[Equ] *Equinox*. URL: `http://www.eclipse.org/equinox/`.

[Erl07] Thomas Erl. *SOA: Principles of Service Design*. First. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007.

[FFB06] E. Freeman, E. Freeman und B. Bates. *Entwurfsmuster von Kopf bis Fuß*. Ein Buch zum Mitmachen und Verstehen. O'Reilly, 2006.

[Fie12] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architecture*. Webseite. Online verfügbar unter `http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm` (besucht am 25. April 2013). 2012.

[Fou12] Apache Software Foundation. *Java Configuration API*. Webseite. Online verfügbar unter `http://commons.apache.org/configuration/` (besucht am 25. April 2013). 2012.

[Fur12] J. Furthmüller. *Verfahren und Protokolle für energiebewusste, gemeinsame Ressourcenverwendung mit mobilen Geräten*. KIT Scientific Publishing, 2012. ISBN: 978-3-86644-877-3.

[Hib] *Hibernate*. URL: `http://www.hibernate.org/`.

[Ids] *IDS Korpusarchiv*. Webseite. Online verfügbar unter `http://www.ids-mannheim.de/kl/projekte/korpora/archiv.html` (besucht am 25. April 2013).

[Iso] *Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)*. Norm. 2011.

[Jan+10] D. Jannach u. a. *Recommender Systems: An Introduction*. first. Cambridge University Press, 2010.

[KI10] Prof. Dr. G. Kern-Isberner. *Darstellung, Verarbeitung und Erwerb von Wissen*. 2010. URL: `http://ls1-www.cs.uni-dortmund.de/ie/lehre/ws1011/dvew/folien/`.

[Kla09] André Klahold. *Recommender Systems - Grundlagen, Konzepte und Lösungen*. Vieweg Teubner, 2009.

[MS12] Prem Melville und Vikas Sindhwani. *Encyclopedia of Machine Learning – Recommender Systems*. PDF-Dokument. Online verfügbar unter `http://www.prem-melville.com/publications/recommender-systems-eml2010.pdf` (besucht am 25. April 2013). 2012.

[Mye] *myExperiment*. Webseite. Online verfügbar unter `www.myexperiment.org` (besucht am 25. April 2013). 2012.

[Nag] Ralf Nagel. *Technische Herausforderungen modellgetriebener Beherrschung von Prozesslebenszyklen aus der Fachperspektive: Von der Anforderungsanalyse zur Realisierung*.

[Orm] *Objektrelationale Abbildung*. URL: `http://de.wikipedia.org/wiki/Objektrelationale_Abbildung`.

[Osga] *OSGi*. URL: `http://www.osgi.org`.

[Osgb] *OSGi Framework*. URL: `http://www.osgi.org/Technology/WhatIsOSGi`.

[Osgc] *OSGi Security*. URL: http://sfelix.gforge.inria.fr/osgi-security/osgi.html.

[Owl] *OWL-API*. Webseite. Online verfügbar unter http://owlapi.sourceforge.net (besucht am 25. April 2013).

[Rsw] *Regeln für den Schlagwortkatalog (RSWK)*. Webseite. Online verfügbar unter www.dbi-berlin.de/dbi_pub/einzelpu/regelw/rswk/rswk_00.htm (besucht am 25. April 2013). 2012.

[Sto09] M. Stolzenberger. *Empfehlungssysteme: Transparente Visualisierung im mobilen Umfeld*. Diplomica Verlag, 2009.

[Tav] *Taverna Workflow Management System*. URL: http://www.taverna.org.uk/.

[TH01] L. Terveen und W. Hill. *Beyond Recommender Systems: Helping People Help Each Other*. Webseite. Online verfügbar unter http://irecvn.googlecode.com/svn-history/r12/trunk/Documents/rec-sys-overview.pdf (besucht am 25. April 2013). 2001.