Service-orientierte Architekturen für Information Retrieval

Sven Meyer zu Eissen and **Benno Stein**

Fakultät Medien, Mediensysteme
Bauhaus-Universität Weimar, 99421 Weimar

sven.meyer-zu-eissen@medien.uni-weimar.de
benno.stein@medien.uni-weimar.de

Abstract

Dieses Papier gibt eine Einführung in TIRA[1], einer Software-Architektur für die Erstellung maßgeschneiderter Information-Retrieval-Werkzeuge. TIRA ermöglicht Anwendern, den Verarbeitungsprozess eines gewünschten IR-Werkzeugs interaktiv als Graph zu spezifizieren: die Knoten des Graphen bezeichnen so genannte „IR-Basisdienste“, Kanten modellieren Kontroll- und Datenflüsse. TIRA bietet die Funktionalität eines Laufzeit-Containers, um die spezifizierten Verarbeitungsprozesse in einer verteilten Umgebung auszuführen.

Motivation für unsere Forschung ist u. a. die Herausforderung der Personalisierung: Es gibt eine Diskrepanz zwischen der IR-Theorie und ihren Algorithmen und der – an persönlichen Wünschen angepassten – Implementierung, Verteilung und Ausführung entsprechender Programme. Diese Kluft kann mit adäquater Softwaretechnik verkleinert werden.

1 Einleitung

Information-Retrieval (IR) ist eine Schlüsseltechnologie im Umgang mit der Informationsüberflutung, die wiederum aus ubiquitärer Verfügbarkeit von Informationen und einer schnell wachsenden Zahl an Informationsquellen und -erzeugern entsteht. Dabei steht IR nicht als universelle Lösung für ein generisches Problem – vielmehr ist IR ein Sammelbegriff für unzählige Lösungen individueller Informationsbedürfnisse. Um wirksam und nützlich zu sein, muss IR-Technologie an persönliche Fragestellungen, an persönliche Vorlieben, an persönliche Fähigkeiten und an persönliche Daten angepasst werden. Diese Forderung wird von existierender IR-Technologie erst ansatzweise erfüllt: beispielsweise werden generische Suchmaschinen bei der Suche im World Wide Web benutzt, die nicht über problemspezifisches Wissen verfügen und den Erfolg einer Suche der Kreativität des Anwenders und seiner Erfahrung und Zeit überlassen.

Aus der Sicht der Softwaretechnik steht hinter jedem IR-Werkzeug ein bestimmter IR-Prozess. Die Implementierung eines IR-Prozesses sollte nicht monolithisch geschehen, sondern dem Paradigma der Service-Komposition folgen: Ein maßgeschneidertes IR-Werkzeug für einen individuellen IR-Prozess könnte durch die Kombination von IR-Basisdiensten spezifiziert und operationalisiert werden. Für eine Architektur, die so etwas ermöglicht, wünscht man sich folgende Eigenschaften:

- *Flexibilität.* Die Spezifikation von IR-Prozessen, ihre Anpassung an geänderte Informationsbedürfnisse und ihre Evaluation kann ad-hoc geschehen.
- *Offenheit.* Die Entwicklung und Integration neuer IR-Dienste ist unterstützt.
- *Modularität.* Das Speichern und die Wiederbenutzung von Prozessen als eigene Basisdienste ist möglich.
- *Skalierbarkeit.* Mehr Rechenleistung führt zu schnellerer Ausführung.

Das vorliegende Papier beschäftigt sich mit diesen Herausforderungen und stellt entsprechende Lösungen vor. Kapitel 2 beschreibt den Zusammenhang zwischen Retrieval-Theorie und IR-Software und motiviert einen Service-orientierten Lösungsansatz zur Implementierung von IR-Prozessen. Kapitel 3 diskutiert Formalismen, mit denen IR-Prozesse modelliert werden können, und Kapitel 4 erläutert die Konzepte von TIRA.

2 Von IR-Theorie zu IR-Software

Abhängig von einer gegebenen Retrieval-Aufgabe können verschiedene Aspekte eines Dokuments d wichtig sein, z. B. sein Layout, sein struktureller oder logischer Aufbau, oder seine Semantik. Eine Computerrepräsentation $\mathbf{d}$ von d muss die für die Retrieval-Aufgabe relevanten Aspekte widerspiegeln; bei der Konzipierung einer Repräsentation spielen linguistische Theorien, Algorithmen zur Textanalyse, Datenstrukturen zur Verwaltung großer Datenmengen und statistische Erkenntnisse eine Rolle. Eine optimale Repräsentation $\mathbf{d}$ ist sowohl auf die formalisierte Anfrage $\mathbf{q}$ gemäß der Retrieval-Aufgabe als auch auf das Retrieval-Modell $\mathcal{R}$ abgestimmt. Dabei umfasst $\mathcal{R}$ die linguistische Theorie, auf der die Abbildung $d \mapsto \mathbf{d}$ basiert, sowie die Funktion $\rho(\mathbf{q}, \mathbf{d})$, welche die Relevanz einer Abfrage $\mathbf{q}$ zu der Computerrepräsentation $\mathbf{d}$ eines Dokuments quantifiziert.

Abbildung 1 (unterhalb der gestrichelten Linie) illustriert diese Zusammenhänge; darüber ist die abstrakte Softwaretechniksicht dargestellt: der individuelle Informationsbedarf eines Anwenders wird durch einen IR-Prozess erfüllt.

Tatsächlich ist die gegenwärtige Praxis bei der Implementierung von IR-Prozessen *bibliotheksbasiert*: Funktionen, die mehr oder weniger komplexe Aufgaben lösen, werden mit generischen Schnittstellen versehen und in anderen Projekten wiederverwendet. Diese Praxis hat sich

[1] Akronym für „Text-based Information Retrieval Architecture“.

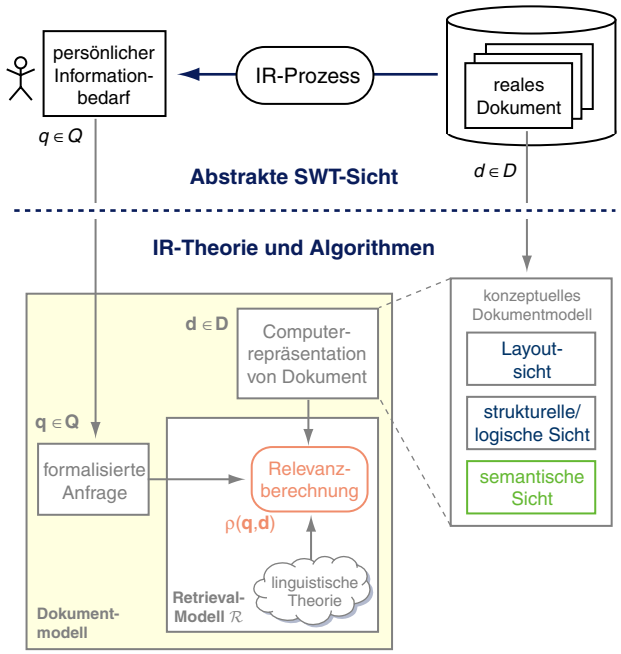

Abbildung 1: Als Ergebnis eines IR-Prozesses wird zu einem Informationsbedarf q ein passendes Dokument d geliefert (oberhalb der gestrichelten Linie). Die Realisierung dieses Prozesses bedingt die Abstraktion von q und d zu Computerrepräsentationen $\mathbf{q}$ bzw. $\mathbf{d}$ (unterhalb der gestrichelten Linie). Dieser Abstraktion liegt eine linguistische Theorie zugrunde, die in einem Retrieval-Modell $\mathcal{R}$ operationalisiert ist.

teilweise bewährt, sie berücksichtigt jedoch kaum die IR-spezifische Entwurfssituation:

- IR-Prozesse bestehen aus autonomen Software-Bausteinen, im folgenden als Module bezeichnet. Grundsätzlich stellt jedes Modul einen Dienst zur Verfügung, der eine Eingabedatenstruktur in eine Ausgabedatenstruktur überführt. Beispiele für solche Module sind Importfilter, Cluster-Algorithmen, Validitätsmaße, Ranking-Funktionen, Klassifizierer, POS-Tagger oder Visualisierungsalgorithmen.
- Typisch im Information Retrieval ist die Existenz alternativer Lösungen sowohl für ein und dieselbe Aufgabe als auch für verwandte Probleme.[2] Beispielsweise gibt es statistische und regelbasierte Stemming-Algorithmen [Porter 1980; Stein und Potthast 2006] oder interne, externe und korpusbasierte Schlüsselwortextraktionsverfahren.
- Unterschiedliche Aufgaben innerhalb eines IR-Prozesses können mit unterschiedlichen Parametrisierungen eines generischen Algorithmus gelöst werden.[3] Beispiele hierfür sind Stemming-Algorithmen oder Stopwort-Filter [Porter 2001], die abhängig von der gewünschten Sprache Regeln oder Wortlisten als Eingabeparameter erhalten.
- IR-Prozesse werden häufig modifiziert: sie werden optimiert, mit neuen Ideen erweitert und an sich ändernde Informationsbedürfnisse angepasst.
- Häufig lassen sich Teile von IR-Prozessen parallel ausführen, insbesondere wenn Dokumente bezüglich unterschiedlicher Fragestellung analysiert werden. Ein Beispiel hierfür ist die intrinsische Ähnlichkeitsanalyse einer Dokumentkollektion bzgl. Thema, Genre, oder Schreibstil [Ifrim *et al.* 2005; Stamatatos *et al.* 2000; Meyer zu Eissen und Stein 2006].
- Es gibt Standardmodule, die für fast jeden IR-Prozess von Nutzen sind. Hierzu zählen Module für das Stemming, Module für die Stopwortentfernung oder Konverter für Binärformate wie Adobe Acrobat (PDF) oder Microsoft Word.

Die dargestellten Punkte zeigen die modulare Natur von IR-Prozessen; dieser sollte bei einer Operationalisierung Rechnung getragen werden. In diesem Zusammenhang schlagen wir ein zweistufiges Konzept vor: Spezifikation eines IR-Prozesses als Diagramm (Schritt 1), das automatisch instanziiert und als verteiltes Softwaresystem implementiert wird (Schritt 2).

3 Spezifikation von IR-Prozessen

Als Beispiel für einen IR-Prozess betrachten wir die Aufgabe, ein Dokument sowohl nach Thema als auch nach Genre in eine Themen-Taxonomie bzw. Genre-Taxonomie einzuordnen [Meyer zu Eißen und Stein 2004]. Abbildung 2 zeigt eine Spezifikation des zugrunde liegenden IR-Prozesses in Pseudo-Code: aus einem Dokument mit der URL `u` werden Computerrepräsentationen für eine Themen-Kategorisierung und eine Genre-Kategorisierung erstellt, die als Eingabe für bereits konstruierte Klassifizierer dienen. Man beachte, dass mehrere Textrepräsentationen (HTML, Rohtext, gefilterter Text) notwendig sind, um die Repräsentationen zu generieren.

Diese Art der Spezifikation folgt dem eingangs beschriebenen bibliotheksbasierten Paradigma, und sie besitzt mehrere Schwächen: (*i*) der Austausch eines Moduls zieht fehleranfällige Datenstruktur- und Code-Ersetzungen nach sich, (*ii*) Expertenwissen bezüglich der zugrunde liegenden Softwarebibliothek ist notwendig, (*iii*) das Ausnutzen der Parallelität zwischen Teilaufgaben führt zu einem unflexiblen Design, da die Parallelität im Programm fest verdrahtet werden muss, u. a. in der Gestalt von Threads oder Remote-Function-Calls, (*iv*) auch die Verteilungsstrategie muss fest verdrahtet werden.

Einen Ausweg stellt die Spezifikation von IR-Prozessen auf einer konzeptuellen Ebene dar, beispielsweise mittels einer grafischen Modellierungssprache. In der Vergangenheit wurden verschiedene Modellierungstechniken für ähnliche Fragestellungen eingesetzt; sie lassen sich nach dem folgenden Schema einteilen [Teich 1997]:

1. kontrollflussdominant oder zustandsorientiert: endliche Automaten, UML Zustandsdiagramme
2. datenflussdominant oder aktivitätsorientiert: Datenflussgraphen, Petri-Netze, markierte Graphen, UML Aktivitätsdiagramme
3. strukturorientiert: Komponentenzusammenhangsdiagramme, UML Klassendiagramme, UML Verteilungsdiagramme
4. zeitorientiert: UML Zeitdiagramme
5. datenorientiert: ER-Diagramme
6. hybrid: Kombinationen der oben genannten Prinzipien, z. B. Kontroll/Datenflussgraphen

[2] Man beachte die Anwendbarkeit des in [Gamma *et al.* 1998] beschriebenen Strategy-Design-Patterns.

[3] Man beachte den Zusammenhang zum Factory-Design-Pattern und zum Decorator-Pattern aus [Gamma *et al.* 1998].

```
Input:  URL u, dictionary dict, stopword list stl.
Output: genre and topic class for the document at URL u.

Text ht=download(u);
Text plainText=removeHTMLTags(ht);
Text filteredText=removeStopwords(plainText, stl);
FeatureVector topicModel=buildTopicModel(filteredText, dict);

Language lang=detectLanguage(plainText);
FeatureVector presentationFeatures=buildPresentationFeatures(ht);
FeatureVector posFeatures=buildPOSFeatures(plainText, language);
FeatureVector genreModel=union(presentationFeatures, posFeatures);

int topicClass=classifyTopic(topicModel);
int genreClass=classifyGenre(genreModel);

return(topicClass, genreClass);
```

Abbildung 2: IR-Prozess für eine Kategorisierungsaufgabe, spezifiziert in Pseudo-Code

Ein Großteil der IR-Prozesse kann als datenflussdominant angesehen werden, da sie von Anwendern mittels einer Anfrage gestartet werden und kein involviertes Modul ohne die Ausgabedaten seiner Vorgängermodule ausführbar ist.

Neben der Möglichkeit, Datenabhängigkeiten zu spezifizieren, muss ein Modellierungsansatz für IR-Prozesse es auch ermöglichen, Parallelität (Verzweigungen und Synchronisation) zu definieren. Weiterhin sollte ein Modellierungsansatz die Typisierung von Daten unterstützen, um leistungsfähige Constraints für die Menge möglicher Modulverbindungen definieren zu können. Abhängig von der Modellierungsgranularität kann es sinnvoll sein, Iterationen auf Teilprozessen mit den damit verbundenen Bedingungen zu formulieren.

Der folgende Abschnitt diskutiert gängige Modellierungswerkzeuge in Hinblick auf ihre Eignung zur Spezifikation von IR-Prozessen.

3.1 Petri-Netze

Ein Petri-Netz [Petri 1962; Teich 1997] ist ein Tupel $N = \langle S, T, F, c, w, m_0 \rangle$; dabei ist

- $S = \{s_1, \ldots, s_m\}$ eine Menge von Stellen,
- $T = \{t_1, \ldots, t_n\}$ eine Menge von Transitionen,
- $S \cap T = \emptyset$,
- $F \subseteq (S \times T) \cup (T \times S)$ eine Flussrelation; die Elemente in F werden auch als Kanten bezeichnet,
- $k : S \to \mathbf{N} \cup \{\infty\}$ eine Funktion, die eine Kapazitätsbeschränkung für jede Stelle definiert,
- $w : F \to \mathbf{N}$ eine Funktion, die Kantengewichte definiert,
- $m_0 : S \to \mathbf{N}_0$ eine Anfangsmarkierung mit der Eigenschaft $\forall s \in S : m_0(s) \leq k(s)$.

Ein Petri-Netz ist ein bipartiter Graph, dessen Knotenmenge aus einer Stellenmenge S und einer Transitionsmenge T besteht, wobei jede Stelle $s \in S$ bis zu $k(s)$ Marken aufnehmen kann. Stellen werden als Kreise notiert, Marken stellen Punkte in den entsprechenden Kreisen dar, das Symbol für eine Transition ist ein Balken, und gerichtete Kanten aus F werden als Pfeile zwischen Transitionen und Stellen notiert. Die Markierung m_0 definiert eine initiale Verteilung der Marken auf die Stellen, und eine beliebige Markierung ist mit $m : S \to \mathbf{N}_0$ bezeichnet. Abhängig davon, welche Transition feuert, ändert sich auch die Markierung. Abbildung 3 zeigt ein Petri-Netz, das die Kategorisierungsaufgabe modelliert.

Semantik von Petri-Netzen Für $x \in S \cup T$ bezeichne $\bullet x = \{y \mid (y, x) \in F\}$ den Vorbereich von x und $x\bullet = \{y \mid (x, y) \in F\}$ den Nachbereich von x. Eine Transition $t \in T$ heißt aktiviert unter einer gegebenen Markierung m genau dann, wenn gilt:

1. $\forall s \in \bullet t \setminus t\bullet : m(s) \geq w(s, t)$
2. $\forall s \in t\bullet \setminus \bullet t : m(p) \leq k(s) - w(t, s)$
3. $\forall s \in t\bullet \cap \bullet t : m(s) \leq k(s) - w(t, s) + w(s, t)$

Eine aktivierte Transition t kann feuern. Dadurch werden an jeder Stelle $s \in \bullet t$ genau $w(s, t)$ Marken konsumiert und für jedes $s \in t\bullet$ genau $w(t, s)$ Marken produziert.

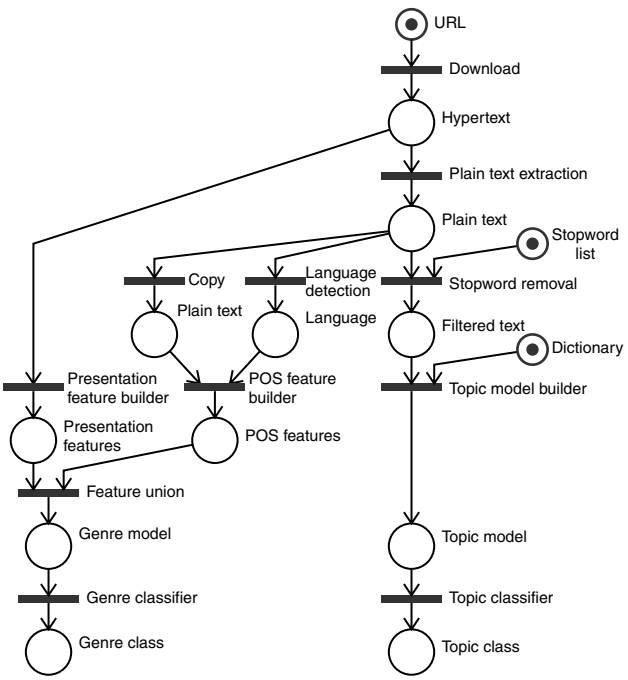

Abbildung 3: IR-Prozess für die Kategorisierungsaufgabe, spezifiziert als Petri-Netz.

Diskussion In unserem Szenario werden Module durch Transitionen modelliert, und die Marken entsprechen den Daten, die in Modulen verarbeitet und zwischen Modulen ausgetauscht werden. Petri-Netze können sowohl sequentielle als auch nebenläufige Prozesse modellieren: Abbildung 3 zeigt ein Petri-Netz für die Kategorisierungsaufgabe. Die beiden Klassifikationen werden parallel ausgeführt und lassen sich synchronisieren, um das Ergebnis einem Anwender anzuzeigen.

Petri-Netze sind gut erforscht; in den vergangenen vierzig Jahren wurden zahlreiche Werkzeuge zu ihrer Analyse und Simulation entwickelt, wie Algorithmen zur Ermittlung der Erreichbarkeit oder zur Feststellung von Deadlocks. Allerdings sind die Marken in Petri-Netzen nicht unterscheidbar und daher nicht zur Modellierung unterschiedlicher Datentypen geeignet. Des weiteren sehen Petri-Netze nicht vor, eine Verarbeitungsreihenfolge der Marken innerhalb der Stellen zu definieren, und es ist fraglich, ob eine Standardstrategie (z. B. FIFO) immer hinreichend für IR-Prozesse ist.

Die Beschränkung bezüglich der Datentypen lässt sich mit gefärbten Petri-Netzen aufheben [Jensen 1997]. Aber auch mit dieser Erweiterung bleibt das Modellieren von Kontrollflüssen stark eingeschränkt: Da Petri-Netze nicht in eine Marke hinein sehen können, lassen sich Kontrollflüsse, die von den Werten der Daten abhängen, nur umständlich modellieren.

3.2 Datenflussgraphen und Kontroll/Datenflussgraphen

Ein Datenflussgraph $G = \langle V, E \rangle$ ist ein gerichteter Graph, in dem jeder Knoten eine Aufgabe repräsentiert und jede gerichtete Kante einen Datenfluss zwischen ihren inzidenten Knoten darstellt. Die Semantik eines solchen Graphen ist, dass eine Aufgabe $v \in V$ nur dann ausgeführt werden kann, falls alle Aufgaben $u \in V$ mit $(u, v) \in E$ schon ausgeführt worden sind. Abbildung 4 zeigt einen Datenflussgraphen für die Kategorisierungsaufgabe.

Wenn man in dem abgebildeten Datenflussgraphen Knoten und Kanten durch Petri-Netz-Transitionen und -Stellen ersetzt, erhält man ein Petri-Netz, das isomorph zu Abbildung 3 ist. Folglich treffen die oben diskutierten Schwächen auch auf Datenflussgraphen zu. Die Schwäche, dass Kontrollstrukturen wie Iterationen schlecht modellierbar sind, trifft nicht mehr auf die ausdrucksstärkeren Kontroll/Datenflussgraphen (CDFG) zu. Diese nämlich ergänzen Datenflussgraphen um Kontrollflusskanten, die zur Modellierung von Kontrollflussalternativen sowie zur Modellierung von Iterationen benutzt werden können. Üblicherweise definieren Kontrollflusskanten alternative Pfade, von denen genau einer gemäß einer Bedingung begehbar ist.

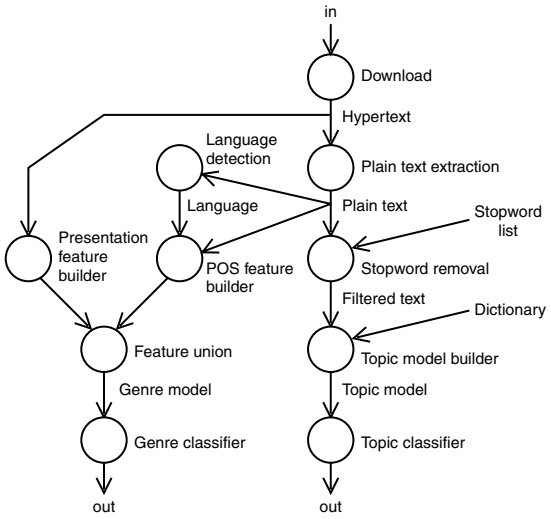

Abbildung 4: IR-Prozess für die Kategorisierungsaufgabe, spezifiziert als Datenflussgraph.

Diskussion CDFGs sind ausdrucksstark genug, um komplexe IR-Prozesse mit Verzweigungen und Iterationen zu modellieren. Allerdings zeigt die Datenflusskomponente in Abbildung 4, dass sich Datentypen und Synchronisation nur implizit durch Kantenbeschriftungen modellieren lassen. Dieses Defizit wird von dem intuitiven UML Modellierungsansatz behoben.

3.3 UML Aktivitätsdiagramme

UML Aktivitätsdiagramme vereinen neue Ideen, die Flusssprachen zur Spezifikation von Web-Service-Kompositionen (z. B. BPEL [Andrews *et al.* 2003]) zugrunde liegen, mit traditionellen Konzepten wie dem Markenkonzept von Petri-Netzen, um Kontroll- und Datenflüsse zwischen so genannten Aktionen zu modellieren. Insbesondere werden Aktionsknoten, Objektknoten und Kontrollknoten mit gerichteten Kanten verbunden, die sowohl Datenflüsse als auch Kontrollflüsse modellieren können [Hitz *et al.* 2005]. Abbildung 5 zeigt ein Aktivitätsdiagramm, das die Kategorisierungsaufgabe modelliert.

Ähnlich wie bei CDFGs werden IR-Basisdienste mit Knoten beschrieben, hier als Aktionsknoten bezeichnet. Objektknoten können zwischen Aktionsknoten platziert werden und stellen Datenobjekte dar, die zwischen den Aktionsknoten übertragen werden. Alternativ können auch Konnektoren, so genannte Pins, direkt mit Aktionsknoten

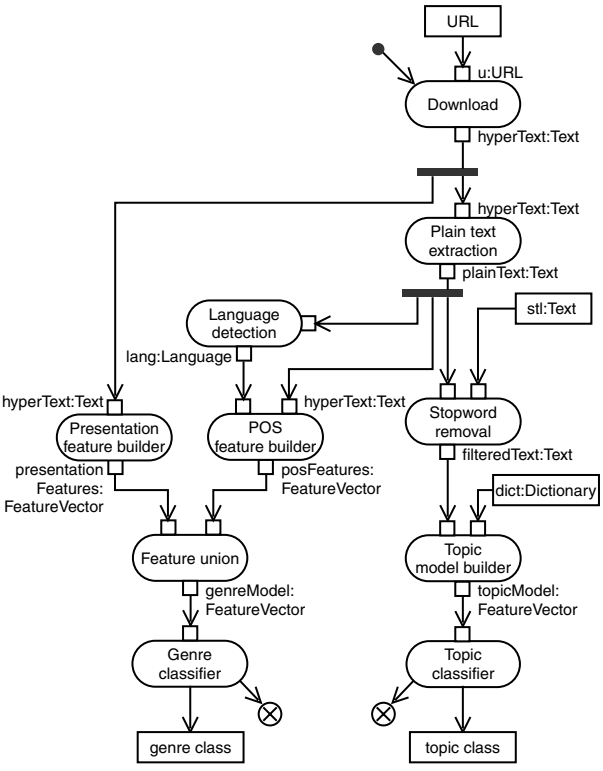

Abbildung 5: IR-Prozess für die Kategorisierungsaufgabe, spezifiziert als UML Aktivitätsdiagramm.

verknüpft werden, um Eingabe- und Ausgabedatentypen zu spezifizieren.

Kontrollknoten unterteilen sich weiter in Verzweigungsknoten (decision nodes), Verschmelzungsknoten (merge nodes), Parallelisierungsknoten (fork nodes) und Synchronisierungsknoten (join nodes). Verzweigungsknoten leiten den Kontrollfluss exklusiv über einen von mehreren möglichen Zweigen, abhängig von der Bedingung, die an einen Verzweigungsknoten gebunden sind; ihr Gegenstück sind die Verschmelzungsknoten. Der Beginn einer nebenläufigen Verarbeitung wird mittels Parallelisierungsknoten modelliert; Synchronisierungsknoten synchronisieren sowohl Daten- als auch Kontrollflüsse.

Ein Aktivitätsdiagramm kann in „Schwimmbahnen“ (swim lanes) unterteilt werden, wobei eine Schwimmbahn dazu dient, Knoten und Kanten hinsichtlich gemeinsamer Eigenschaften zu gruppieren. Solche logischen Einheiten (zum Beispiel in sich geschlossene Teile einer Retrieval-Aufgabe) werden vom Anwender definiert und erlauben es, einen komplexen IR-Prozess zu strukturieren.

Diskussion Nicht nur wegen ihrer Intuitivität sind UML Aktivitätsdiagramme weitläufig akzeptiert. Weiterführende Konzepte umfassen die Modellierung von Streams, Parametermengen, Stereotypen, aktions- und zeitgesteuerten Ereignissen sowie Ausnahmebehandlung. Diese Konzepte sind bereits in der aktuellen Version von UML standardisiert und machen die Diagramme zu einem idealen Modellierungswerkzeug für IR-Prozesse. Für die kommende UML 2.1-Spezifikation sind an Blockdiagramme erinnernde Bedingungs- und Iterationsknoten geplant; sie sollen helfen, die Modellierung von Kontrollflüssen noch weiter zu vereinfachen.

4 Operationalisierung von IR-Prozessen mit TIRA

Unter dem MDA-Paradigma[4] stellt eine UML-Spezifikation eines IR-Prozesses ein plattformunabhängiges Modell (PIM) dar [Object Management Group (OMG) 2003a], da Aktivitätsdiagramme nicht an Programmiersprachen, Betriebssysteme, Middleware oder Systemarchitekturen gebunden sind. Um einen als Aktivitätsdiagramm spezifizierten IR-Prozess zu operationalisieren, ist eine Zielplattform zu wählen und das PIM in ein ausführbares, plattformabhängiges Modell (PSM) zu übersetzen.

Der Begriff Plattform bezeichnet hier die nächste (= tiefere) Abstraktionsebene, auf der ein bestimmtes Modell konkreter beschrieben wird. Beispielsweise sind J2EE und CORBA mögliche Plattformen für die Implementierung von Geschäftsprozessen, und die Java-Entwicklungsumgebung ist eine mögliche Plattform für die CORBA-Implementierung. Das Beispiel verdeutlicht, wie durch eine Folge von Transformationen auf jeweils eine tiefere Ebene ein PIM ausführbar gemacht wird. Die OMG bezeichnet in diesem Zusammenhang alle Plattformen, die sich zwischen PIM und ausführbarem Code befinden, als Middleware-Plattform [Object Management Group (OMG) 2003a].

Wie in anderen MDA-basierten Anwendungsszenarien ist es unser Ziel, ausgehend vom PIM Transformationen auf tiefere Plattform-Ebenen hinsichtlich ihrer Semantik zu definieren und zu implementieren. Im Unterschied zu typischen MDA-basierten Anwendungsszenarien sind wir allerdings nicht daran interessiert, das PIM auf eine große Anzahl verschiedener Middleware-Plattformen abzubilden, sondern konzentrieren uns auf *eine* Zielplattform, die besonders geeignet ist, personalisierte IR-Prozesse auszuführen. Kurz gefasst: unser Fokus liegt auf schnellem Entwurf, kurzen Entwicklungszyklen und einem minimiertem Aufwand für Implementierung und Test.

Abbildung 7 zeigt unseren Vorschlag einer Schichtenarchitektur für TIRA: Eingabe ist ein PIM, das einen IR-Prozess in Form eines UML Aktivitätsdiagramms spezifiziert. Das Diagramm wird, wie im nächsten Abschnitt erläutert, in ein PSM überführt und in einer verteilten Umgebung ausgeführt. Die Module für die IR-Basisdienste stammen aus einer Modulbibliothek; sie sind als Web-Services gekapselt und über Rechnergrenzen hinweg transparent aufrufbar. Sowohl Eingabe als auch Ausgabe der Module sind Datenobjekte, serialisiert in der Form von XML-Datenströmen.

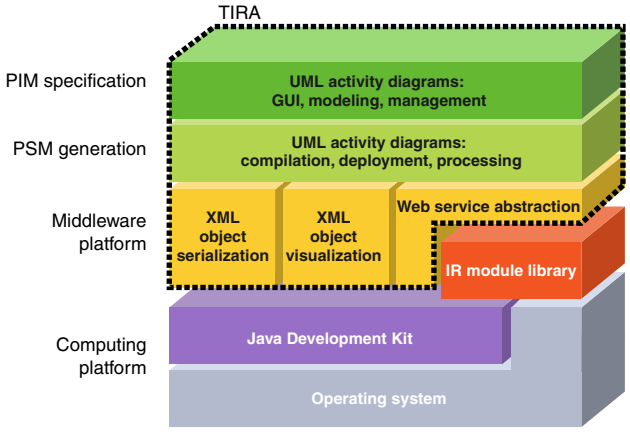

Abbildung 7: Die Schichtenarchitektur von TIRA setzt auf einer Rechnerplattform auf. Die IR-Modul-Bibliothek ist nicht Teil von TIRA, sondern steht für eine offene und erweiterbare Menge von IR-bezogenen Algorithmen und Datenstrukturen.

4.1 Von PIM zu PSM

Ein Aktivitätsdiagramm – sei es aus einer Datei geladen oder interaktiv mit der TIRA-GUI modelliert – wird speicherintern als Objektstruktur dargestellt. Diese Struktur ist am UML Metamodell der Object Management Group (OMG) [2003b] orientiert: Die Objekte der Struktur sind Instanzen von Aktionsknoten, Parallelisierungsknoten usw. und mittels Datenknoteninstanzen verbunden. Diese Struktur wird zu einem Kontroll-/Datenflussgraphen G übersetzt und mit einer Petri-Netz-artigen Markensemantik ausgeführt. Die Aktionsknoten in G werden an die entsprechenden Web-Services gebunden, die Datenknoten in G auf XML-Objekte abgebildet. Zur Abarbeitung von G werden zunächst alle ausführbaren Knoten bestimmt und die assoziierten Web-Services asynchron aufgerufen. Sobald ein Web-Service sein Ergebnis liefert, wird die entsprechende Marke in G propagiert, und die Menge der ausführbaren Aktionsknoten aktualisiert.

4.2 Die TIRA Middleware-Plattform

Die Funktionen in der Modulbibliothek erhalten Objekte als Eingabe und liefern neue Objekte zurück. Anstatt die

[4] MDA steht für Model Driven Architecture.

Web-Services mit Serialisierungen dieser Objekte aufzurufen, ist die Parameterübergabe mit dem Call-By-Name-Paradigma realisiert: Ein Parameter ist eine URL, die auf den Ort einer serialisierten XML-Repräsentation des entsprechenden Objekts zeigt. Dieser Ansatz hat folgende Vorteile:

1. Während der Ausführung eines IR-Prozesses muss ein Client nicht die Daten zwischen zwei Web-Service-Aufrufen übertragen. Stattdessen greift ein Web-Service unmittelbar auf die angegebenen URLs zu. So werden Datentransportkosten reduziert – insbesondere, wenn die Web-Services zweier aufeinander folgender Module auf demselben Server gehostet sind.
2. Der Transfer von URL-Referenzen anstelle von Datenobjekten ermöglicht es, auch Rechner mit geringer Bandbreite zu voll funktionstüchtigen Clients zu machen.
3. Die Verwendung von URLs schafft die Voraussetzung, das World Wide Web zur Datenspeicherung und -verteilung zu nutzen.

XML ist als Standard zum Datenaustausch und zur Serialisierung von Datenobjekten weit verbreitet. In TIRA werden zum Lesen und Schreiben von XML-Daten-Streams moderne Parser-Generatoren der Java-XML-Sprachbindung (JAXB) eingesetzt [Sun Microsystems 2003].

Die zwischen den Modulen ausgetauschten Daten lassen sich für eine visuelle Analyse oder zu Debugging-Zwecken angezeigen. Hierbei kommt die Technik der XSL-Transformationen zum Einsatz, mittels der serialisierte XML-Objekte auf Basis von XSL-Stylesheets in Formate wie XHTML oder PostScript on-the-fly umgewandelt werden können.

4.3 Arbeiten mit TIRA

Abbildung 6 (links) zeigt einen Snapshot unserer Meta-Suchmaschine AIsearch als TIRA-Anwendung. AIsearch sucht zu eingegebenen Schlüsselworten passende Web-Dokumente mit Hilfe kommerzieller Suchmaschinen und ordnet die gefundenen Dokumente nach inhaltlicher Ähnlichkeit [Meyer zu Eißen und Stein 2002]. Der zugrunde liegende IR-Prozess extrahiert die von den Suchmaschinen gelieferten Dokumentausschnitte, entfernt Stopworte, führt eine Wortstammreduktion durch und erzeugt eine komprimierte Term-Vektor-Darstellung. Auf Grundlage der Term-Vektoren wird eine Cluster-Analyse mit dem MajorClust-Algorithmus durchgeführt [Stein und Niggemann 1999] und für die gefundenen Themenkategorien aussagekräftige Bezeichnungen mittels statistischer Textüberdeckungsalgorithmen generiert.

Abbildung 6 (rechts) zeigt einen Snapshot des TIRA Aktivitätsdiagramm-Editors, der in einem Java Applet ausgeführt wird. Die linke Seite zeigt eine Auswahl der instanziierbaren Module und Datenknotentypen. Die Instanzen sind auf der rechten Seite des Applets zu sehen und können mit Pfeilen, die die Richtung des Datenflusses definieren, verbunden werden. Ein Klick auf einen Datenknoten startet die assoziierte XSL-Transformation, die die Daten in ein anzeigbares Format umwandelt, das dann im Browser dargestellt wird.

5 Zusammenfassung

IR-Prozesse haben eine ubiquitäre Präsenz erreicht, sei es in Form von Suchmaschinen im privaten oder professionellen Kontext, auf mobilen Endgeräten oder auf stationären Rechnern, oder als Retrieval-Komponenten in Dateisystemen, Dokumentkollektionen, Datenbanken oder Wissensmanagement-Werkzeugen. Der Grund dieser Durchdringung ist ein wachsender Informationsbedarf, die Vielfältigkeit der IR-Aufgaben, und ein gewünschter Grad an Personalisierung. Obgleich viele spezialisierte Retrieval-Algorithmen in der Vergangenheit entwickelt wurden, sind wenig Anstrengungen unternommen worden, IR-Prozesse aus der Sicht der Softwaretechnik zu modellieren und zu operationalisieren.

Das vorliegende Papier soll an dieser Stelle einen Beitrag leisten: Ausgehend von einer Diskussion einschlägiger Modellierungstechniken bezüglich ihrer Eignung für die Abstraktion von IR-Prozessen haben wir TIRA als eine flexible MDA-Lösung für den schnellen Entwurf maßgeschneiderter IR-Werkzeuge vorgestellt. Mit TIRA ist es möglich, IR-Prozesse als UML Aktivitätsdiagramme zu modellieren, die per Knopfdruck in plattformspezifische Modelle transformiert und in einer verteilten Umgebung ausgeführt werden können.

TIRA hat zurzeit den Status eines Forschungsprototyps und wird in unserer Arbeitsgruppe weiterentwickelt. Der

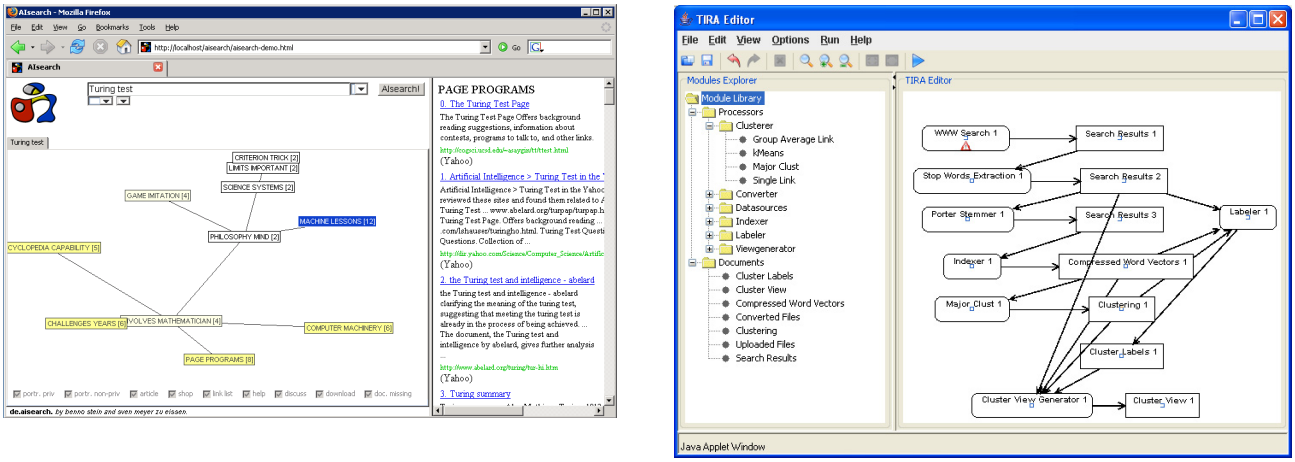

Abbildung 6: Der linke Snapshot zeigt unsere Meta-Suchmaschine AIsearch, deren IR-Prozess mit TIRA modelliert ist. Der rechte Snapshot zeigt den TIRA-Editor für die Spezifikation von IR-Prozessen mittels Aktivitätsdiagrammen.

TIRA-Ansatz ist unabhängig von IR-Algorithmen und Datenstrukturen; unser Ansatz sieht die Einbindung existierender IR-Bibliotheken (und das hierin codierte Know-How) explizit vor.

Literatur

Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic und Sanjiva Weerawarana. Business process execution language for web services (bpel4ws) version 1.1. `http://www-128.ibm.com/developerworks/library/specification/ws-bpel/`, Mai 2003.

Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., 1998.

Martin Hitz, Gerti Kappel, Elisabeth Kapsammer und Werner Retschitzegger. *UML @ Work*. dpunkt.verlag, 2005.

Georgiana Ifrim, Martin Theobald und Gerhard Weikum. Learning Word-to-Concept Mappings for Automatic Text Classification. In *Proceedings ICML, Learning in Web Search Workshop*, 2005.

Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use.*, Volume 1 der *Monographs in Theoretical Computer Science*. Springer, 1997.

Sven Meyer zu Eißen und Benno Stein. The AIsearch Meta Search Engine Prototype. In Amit Basu und Soumitra Dutta, Eds., *Proceedings 12th Workshop on Information Technology and Systems (WITS 02), Barcelona Spanien*. Technische Universität Barcelona, Dezember 2002.

Sven Meyer zu Eißen und Benno Stein. Genre Classification of Web Pages: User Study and Feasibility Analysis. In Susanne Biundo, Thom Frühwirth und Günther Palm, Eds., *KI 2004: Advances in Artificial Intelligence*, Volume 3228 *Lecture Notes in Artificial Intelligence*, S. 256-269, Berlin Heidelberg New York, September 2004. Springer.

Sven Meyer zu Eissen und Benno Stein. Intrinsic plagiarism detection. In Mounia Lalmas, Andy MacFarlane, Stefan M. Rüger, Anastasios Tombros, Theodora Tsikrika und Alexei Yavlinsky, Eds., *Proceedings European Conference on Information Retrieval (ECIR 2006)*, Volume 3936 *Lecture Notes in Computer Science*, S. 565-569. Springer, 2006.

Object Management Group (OMG). Model driven architecture (mda) guide. `http://www.omg.org/docs/omg/03-06-01.pdf`, 2003.

Object Management Group (OMG). The UML Metamodel. `http://www.omg.org/cgi-bin/doc?ptc/2004-10-05`, 2003.

Carl Adam Petri. *Kommunikation mit Automaten*. Dissertation, Universität Bonn, 1962.

M. F. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130-137, 1980.

Martin Porter. Snowball. `http://snowball.tartarus.org/`, 2001.

E. Stamatatos, N. Fakotakis und G. Kokkinakis. Text genre detection using common word frequencies. In *Proceedings 18th Int. Conference on Computational Linguistics*, Saarbrücken, 2000.

Benno Stein und Oliver Niggemann. On the Nature of Structure and its Identification. In Peter Widmayer, Gabriele Neyer und Stefan Eidenbenz, Eds., *Graph-Theoretic Concepts in Computer Science*, Volume 1665 *Lecture Notes in Computer Science*, S. 122-134. Springer, Juni 1999.

Benno Stein und Martin Potthast. Putting Successor Variety Stemming to Work. In *30th Annual Conference of the German Classification Society (GfKl) 2006 (erscheint in Kürze)*, 2006.

Sun Microsystems. Java Architecture for XML Binding (JAXB Specification). `http://java.sun.com/xml/downloads/jaxb.html`, 2003.

Jürgen Teich. *Digitale Hardware/Software-Systeme*. Springer, 1997.