Mehr finden: Hayoo! Haskell API Search

Eine Einführung in das Holumbus Framework

Timo B. Hübel

Fachhochschule Wedel / freiheit.com technologies gmbh

12. Juni 2009
HaL4 – Halle (Saale)

Einführung

Geschichte

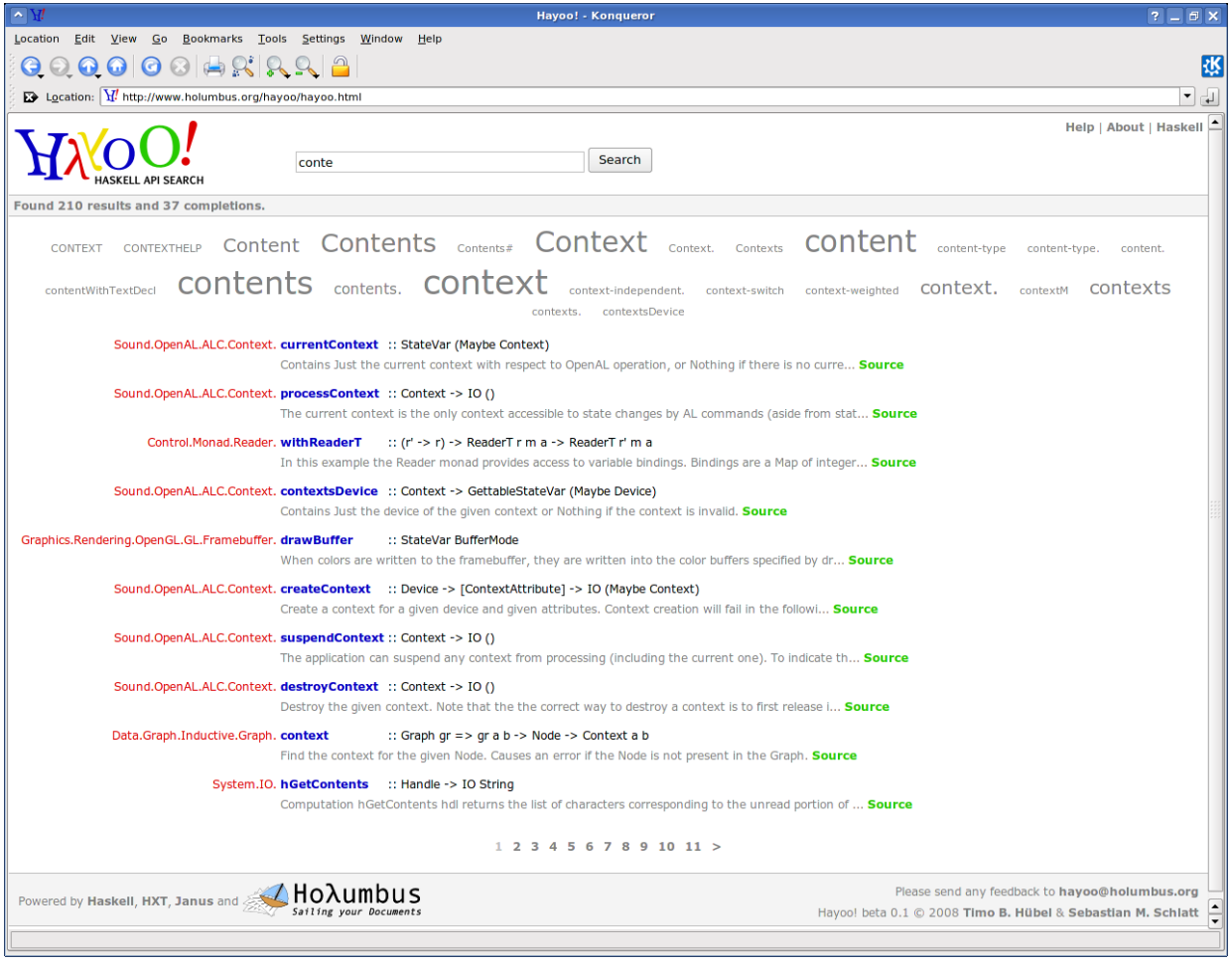
Idee

Flexibles Framework zur Erstellung von spezialisierten Suchmaschinen

Bisherige Arbeiten

- S. Schlatt: *Creating scalable and highly customized crawlers and indexers*
- T. Hübel: *Creating fast, flexible and highly customizable search engines with Haskell*
- S. Schmidt: *Distributed computing with MapReduce in Haskell*

Hayoo!



http://holumbus.fh-wedel.de/hayoo

Hayoo!

Umfang

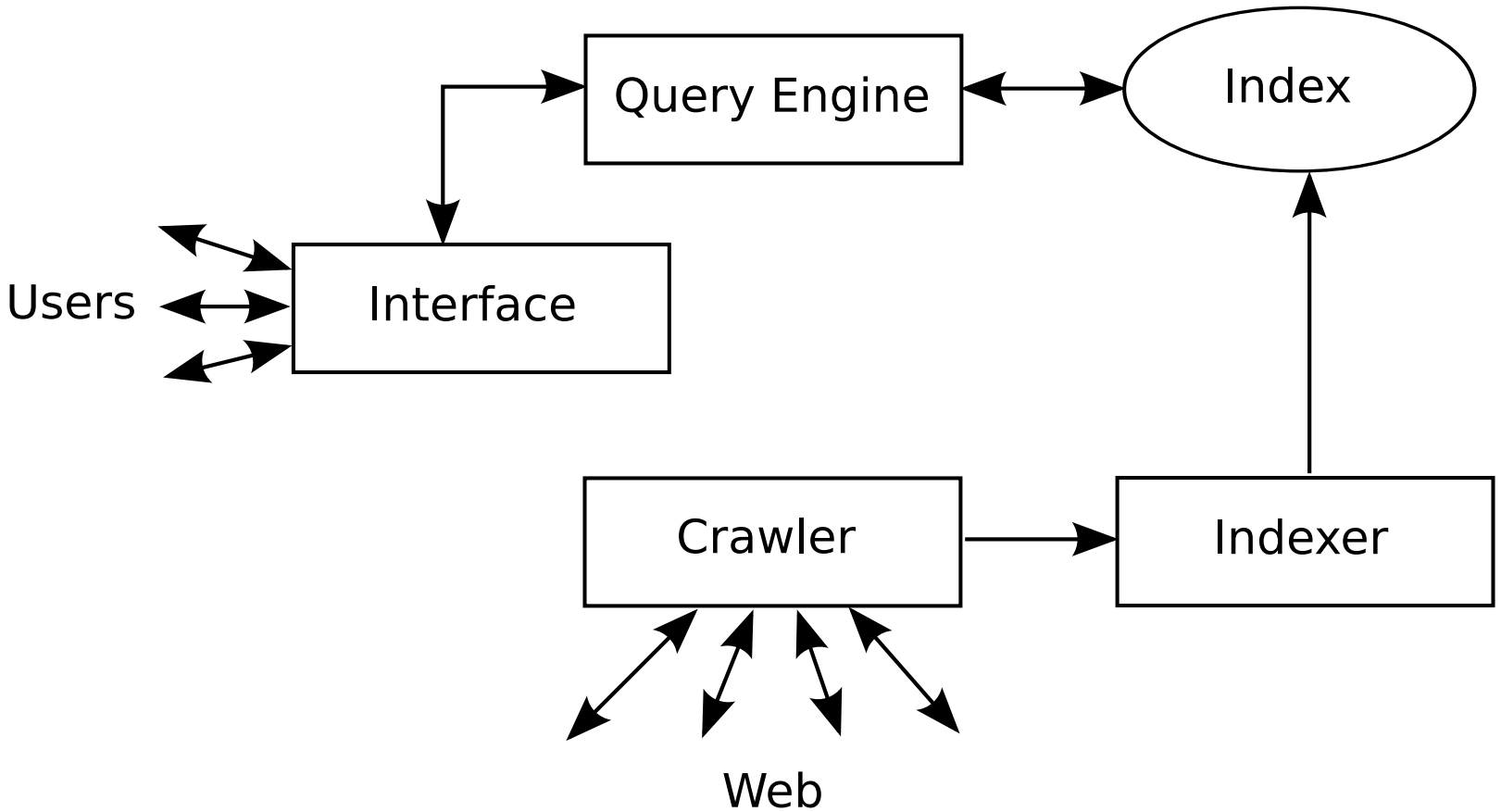
Sämtliche Haskell Pakete von hackage.haskell.org

Durchschnittliche Dokumentgröße: 16 Wörter

Statistik

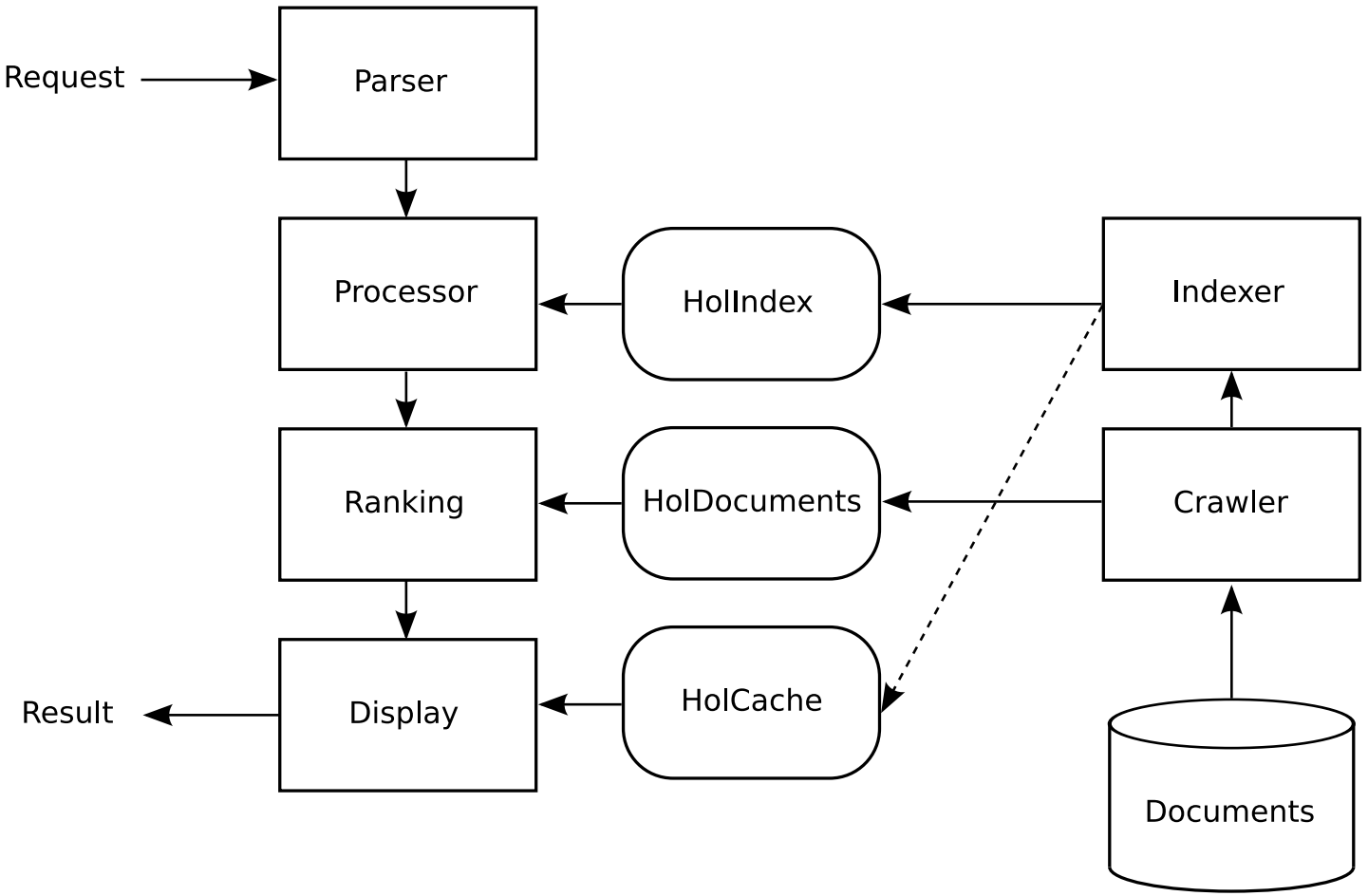
Dokumente	79.590
Wörter	129.742 / 1.258.662
Größe	35,1 MB
Speicher	548 MB
Wachstum	10.901 -> 47.833 -> 49.888 -> 79.590

Suchmaschinen



Crawler-Indexer Architektur nach [BYRN99]

Komponenten



Struktur des *Holumbus* Framework

Schnittstellen

Typklassen

HolIndex Zentrale Index-Schnittstelle

HolDocuments Bidirektionale Abbildung zwischen URI und Id

HolCache Volltext-Cache zum Speichern von Dokumenten

Typparameter

Beliebige Zusatzinformationen zu einem Dokument

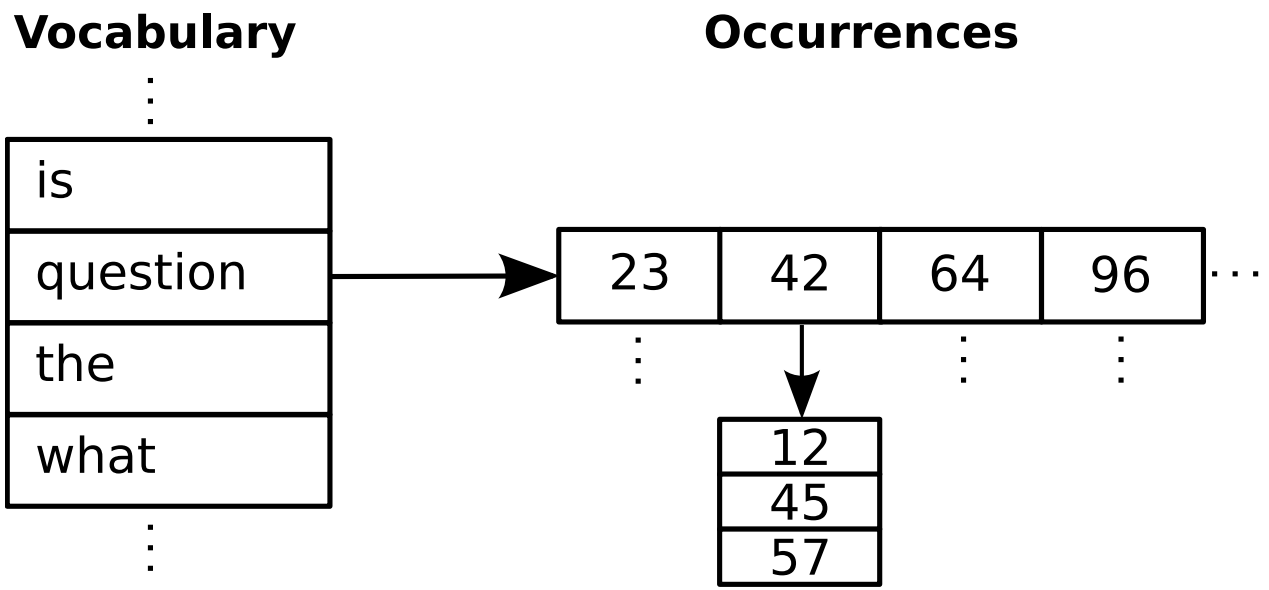
```
data Document a = Document { title   :: Title
                           , uri     :: URI
                           , custom  :: Maybe a
                           }
```

Inverted File

Komponenten

Vocabulary Verzeichnis aller Wörter

Occurrences Positionsangaben zu den Wörtern



Inverted File Datenstruktur

Index

Definition

```
newtype Inverted = Inverted { indexParts :: Parts }

type Parts      = Map Context Part
type Part       = StrMap Occurrences

type Occurrences = IntMap IntSet
```

Details

- Modellierung der Dokumentstruktur durch mehrere Indexe
- Speicherung der Wortpositionen (*Fully Inverted File*)

Patricia-Trie

Eigenschaften

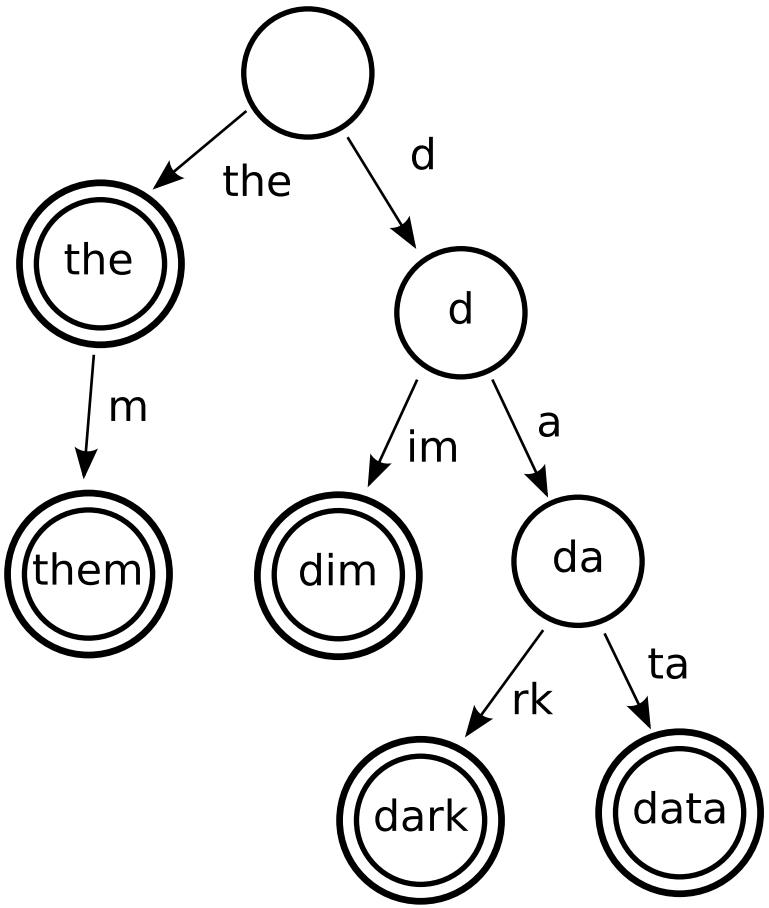
Präfix-Baum mit zusammengefassten Knoten.

Vorteile

- Suche in *O(l)*
- Präfix-Suche möglich
- Speichereffizienz

Nachteile

- Teure Einfügeoperation



StrMap

Definition

```
data Trie a = End Key a [Trie a]
            | Seq Key [Trie a]

type Key = [Word8]
```

Unicode?

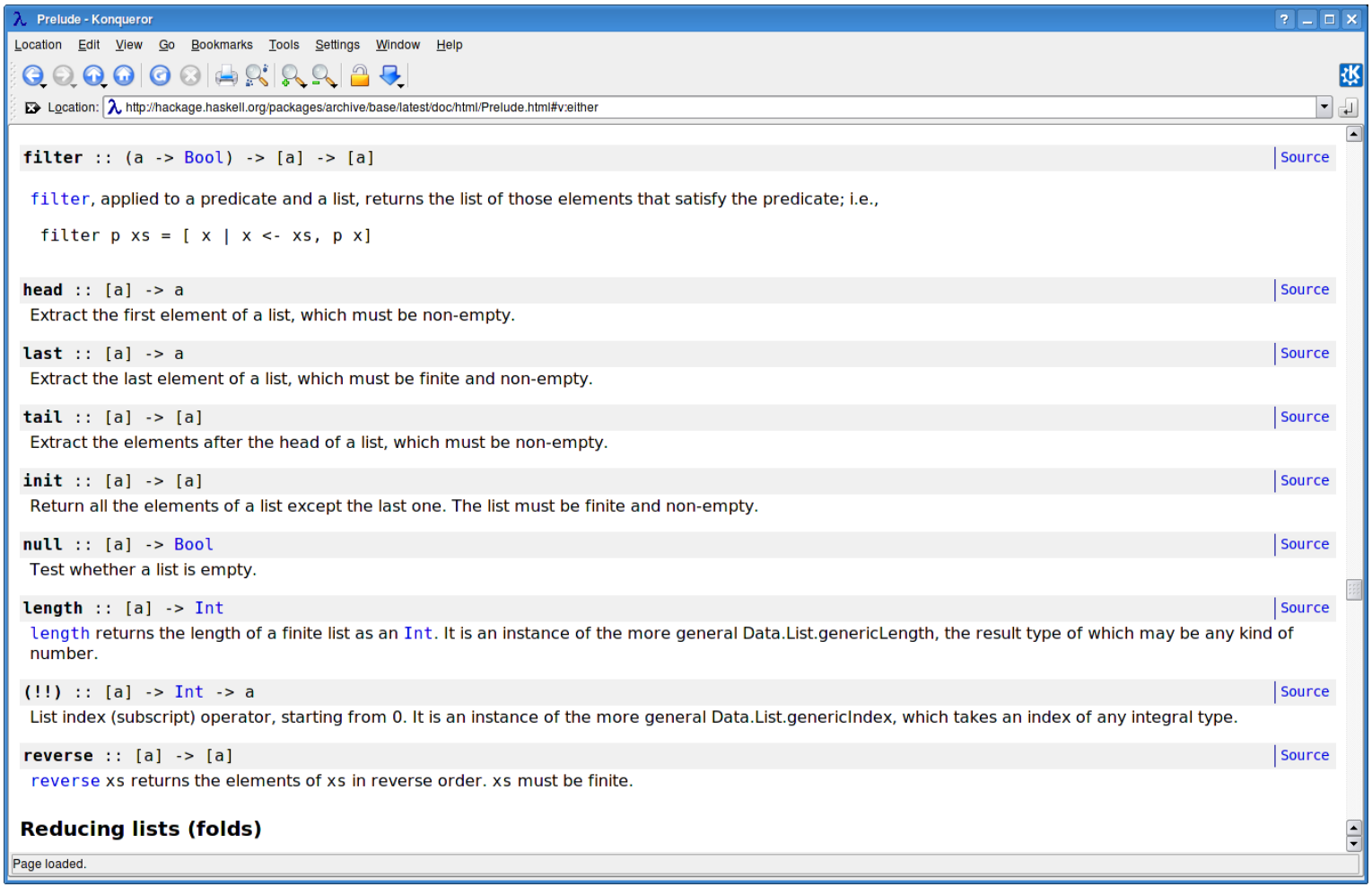
Darstellung von Unicode-Zeichen als Word8 durch UTF8-Encoding

Vorteil

Beliebige auf ByteString abbildbare Typen als Schlüssel

Indexerstellung

Haddock



Haskell Dokumentation generiert von Haddock

Dokumente

Hayoo!

```
type HayooDoc = Document FunctionInfo

data Document a = Document
  { title    :: Title    -- Funktionsname
  , uri      :: URI      -- URI (inkl. Textmarke)
  , custom   :: Maybe a  -- Zusatzinformation
  }

data FunctionInfo = FunctionInfo
  { moduleName  :: String       -- Modulname
  , signature   :: String       -- Signatur
  , package     :: String       -- Cabal-Paket
  , sourceURI   :: Maybe String -- Quellcode
  }
```

Anforderungen

Grundfunktionen

- Verwaltung bearbeiteter und zu bearbeitender Dokumentmengen
- Identifikation von Referenzen
- Ein-/Ausschluss von Dokumenten & -bäumen

Erweiterte Funktionalität

- Identifikation identischer Dokumente mit unterschiedlichen URIs
- Extraktion von Zusatzinformationen

Funktionsweise

Ablauf

1. Abrufen des Dokuments unter einer URI
2. Rekursiver Aufruf für alle im Dokument enthaltenen URIs

Interner Zustand (vereinfacht)

```
data CrawlerState a = CrawlerState
  { cs_toBeProcessed  :: Set URI
  , cs_wereProcessed  :: Set URI
  , cs_fCrawlFilter   :: (URI -> Bool)
  , cs_docs           :: Documents a
  , cs_fGetCustom     :: IOSArrow XmlTree (Maybe a)
  , cs_docHashes      :: Map MD5Hash URI
  }
```

Transformation

Mögliche Transformationen

- Berechnung Ranking-relevanter Werte
- Aufteilung von Dokumenten in logische Dokumente
- Änderung der URIs um auf lokalen Kopien zu arbeiten

Hayoo!

Zerlegung der Haddock-Dokumentation in viele Einzeldokumente (eines pro Funktion)

Anforderungen

Grundfunktionen

- Konfiguration verschiedener Kontexte
- Zerlegung von Zeichenketten in Wörter
- Identifikation von Stop-Wörtern

Erweiterte Funktionalität

- Vorverarbeitung des Dokuments
- Erstellung eines Caches

Indexer

Konfiguration (vereinfacht)

```
data IndexerConfig = IndexerConfig
    { ic_startPages      :: [URI]
    , ic_tmpPath         :: Maybe String
    , ic_contextConfigs  :: [ContextConfig]
    }

data ContextConfig = ContextConfig
    { cc_name            :: String
    , cc_preFilter       :: ArrowXml a => a XmlTree XmlTree
    , cc_fExtract        :: ArrowXml a => a XmlTree XmlTree
    , cc_fTokenize       :: String -> [String]
    , cc_fIsStopWord     :: String -> Bool
    , cc_addToCache      :: Bool
    }
```

Hayoo! Indexer

Kontexte im Hayoo! Index

description	Funktionsbeschreibung (*Fold the values...*)
hierarchy	Vollqualifizierter Modulname (*Data.Map*)
module	Modulname (*Map*)
name	Funktionsname (*foldWithKey*)
package	Paketname (*containers*)
signature	Signatur der Funktion (*Int -> Int -> Bool*)
normalized	“Normalisierte” Signatur (*a -> a -> b*)

Suchmaschine

Boolean Model

Operatoren

AND Schnitt von Dokumentmengen

OR Vereinigung von Dokumentmengen

NOT Komplement einer Dokumentmenge

Eigenschaften

- Weit verbreitet und bekannt
- Einfacher und prägnanter Formalismus

Erweiterte Funktionalität

Präfix-Suche

Suche nach “Test” liefert z.B. auch “Testbericht”, “Testanalyse”, “Testergebnisse”, “Testen”, etc.

- Zusätzliche Wörter als Vorschläge (*suggestions*)
- Suchergebnisse ab dem ersten Buchstaben (*find as you type*)

Dokumentstruktur

Nutzung der Abbildung der Dokumentstruktur im Index.

- Standardmäßig Durchsuchen aller Indexe
- Konkrete Anfragen an einzelne Indexe

Erweiterte Funktionalität

Präfix-Suche

Suche nach “Test” liefert z.B. auch “Testbericht”, “Testanalyse”, “Testergebnisse”, “Testen”, etc.

- Zusätzliche Wörter als Vorschläge (*suggestions*)
- Suchergebnisse ab dem ersten Buchstaben (*find as you type*)

Dokumentstruktur

Nutzung der Abbildung der Dokumentstruktur im Index.

- Standardmäßig Durchsuchen aller Indexe
- Konkrete Anfragen an einzelne Indexe

Query

Anfragesprache

```
data Query = Word       String | Phrase       String
           | CaseWord   String | CasePhrase   String
           | FuzzyWord  String | Negation     Query
           | Specifier  [Context] Query
           | BinQuery   BinOp Query Query

data BinOp = And | Or | But
```

Verwendung

- Parser - Umwandlung beliebiger Syntax in Query-Struktur
- Prozessor - Rekursive Auswertung der Query-Struktur

Query

Anfragesprache

```
data Query = Word       String | Phrase       String
           | CaseWord   String | CasePhrase   String
           | FuzzyWord  String | Negation     Query
           | Specifier  [Context] Query
           | BinQuery   BinOp Query Query

data BinOp = And | Or | But
```

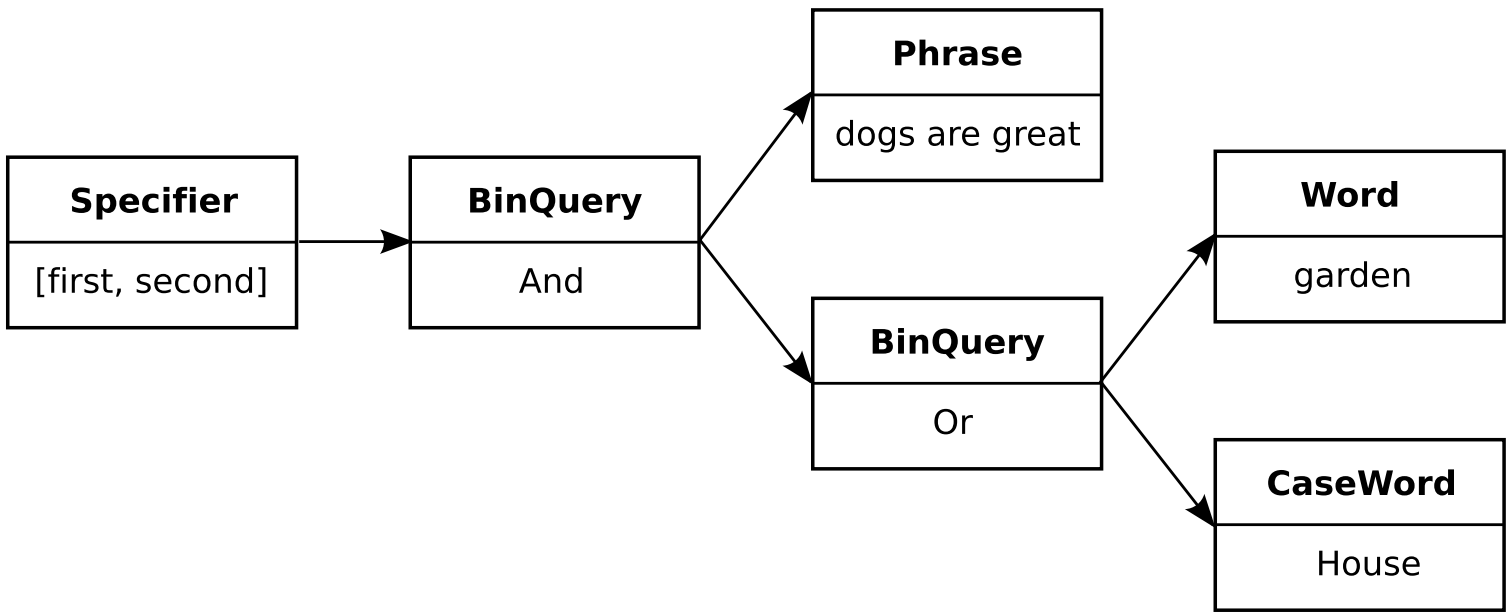
Verwendung

- Parser - Umwandlung beliebiger Syntax in Query-Struktur
- Prozessor - Rekursive Auswertung der Query-Struktur

Query Beispiel

Beispiel

first,second:(“dogs are great” AND (garden OR !House))



Resultierende Query-Strukur

Prozessor

Funktionsweise

1. Rekursiver Abstieg durch den Query-Baum
2. Auswertung der Blätter durch Indexanfragen
3. Anwendung entsprechender Operatoren

Operationen

Einschränkung der Kontexte und Abbildung der Boolschen Operatoren auf entsprechende Mengen-Operationen.

```
data ProcessState i = ProcessState
  { contexts :: [Context]
  , index    :: i
  }
```

Ergebnis

Inhalt

Ergebnis enthält Dokumente und Vervollständigungen

Datenstruktur

```
data Result a = Result
  { docHits  :: (DocHits a), wordHits :: WordHits }

data DocInfo a = DocInfo (Document a) Score
data WordInfo = WordInfo Terms Score

type DocHits a = IntMap (DocInfo a,
  Map Context (Map Word Positions))

type WordHits = Map Word (WordInfo,
  Map Context (IntMap Positions))
```

Ranking

Konfiguration

```
data RankConfig = RankConfig
  { docRanking  :: DocId -> DocHits -> Score
  , wordRanking :: Word -> WordHits -> Score
  }
```

Hayoo!

- Bevorzugung von Treffern in *Prelude* und *base*
- Gewichtung der Kontexte (z.B. Name vor Beschreibung)
- Bessere Bewertung exakter Treffer

Ausblick

Holumbus

- Alternative Implementierungen der Index-Datenstruktur
- Verteilung und Parallelisierung der Query-Auswertung
- Praktischer Einsatz des MapReduce Frameworks

Hayoo!

- Verbessertes Ranking durch “Dependency-Rank”
- Automatisierte Aktualisierung des Index

Ausblick

Holumbus

- Alternative Implementierungen der Index-Datenstruktur
- Verteilung und Parallelisierung der Query-Auswertung
- Praktischer Einsatz des MapReduce Frameworks

Hayoo!

- Verbessertes Ranking durch “Dependency-Rank”
- Automatisierte Aktualisierung des Index

Fragen?

Vielen Dank für die Aufmerksamkeit!

Weitere Infos, Quellcode etc. unter:

http://holumbus.fh-wedel.de

Hayoo! Haskell API Search unter:

http://holumbus.fh-wedel.de/hayoo

Fragen an:

`timo.huebel@freiheit.com`