ENTITY LINKING IN FOR FUN (CLOJURE)

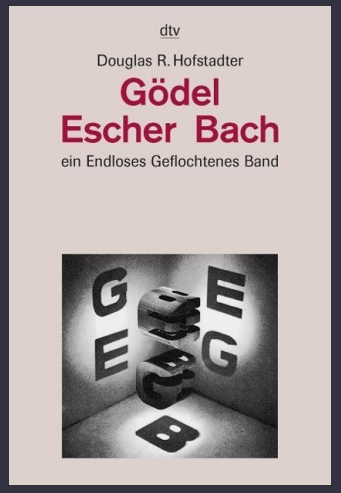
@Sojoner

AGENDA

- ★ Motivation
- ★ Entity linking
- ★ Reflector overview
- ★ Represent the data
- ★ A bit code
- ★ Learnings

MOTIVATION

- ★ practicing Clojure backend & frontend
- ★ using core.async
- ★ learning about FP
- ★ Gödel-Escher-Bach like information revealer
- ★ How will the Wikipedia reflect a given text when thrown against it ?



ENTITY LINKING

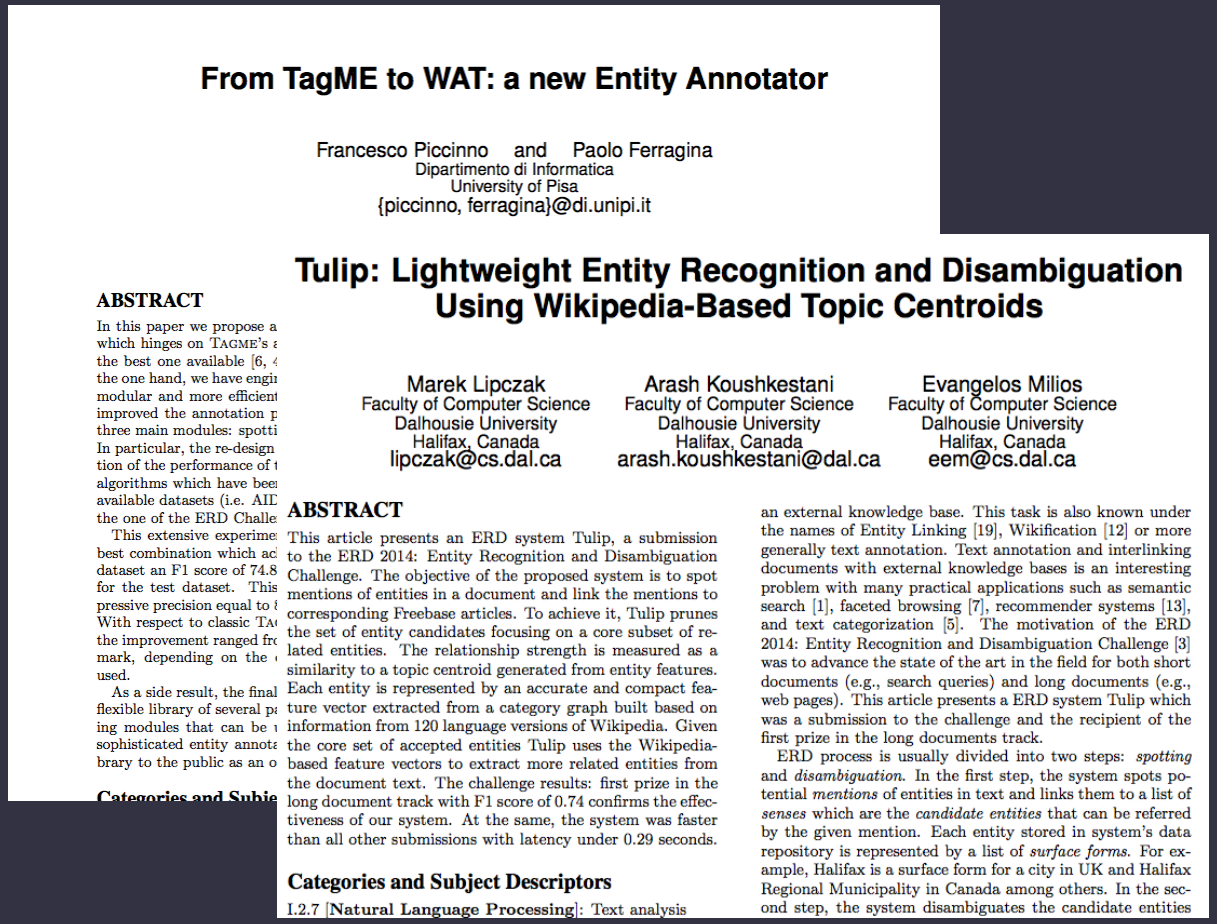
& SEARCH

- is the task of determining the identity of entities mentioned in text [...] Wikipedia

- You can improve search experience by enriching documents with entities e.g facet search for persons locations a.o

ENTITY LINKING

SYSTEMS & CHALLENGES



From TagME to WAT: a new Entity Annotator

Francesco Piccinno and Paolo Ferragina
Dipartimento di Informatica
University of Pisa
{piccinno, ferragina}@di.unipi.it

ABSTRACT

In this paper we propose a
which hinges on TAGME's a
the best one available [6, 4
the one hand, we have engi
modular and more efficient
improved the annotation p
three main modules: spotti
In particular, the re-design
tion of the performance of t
algorithms which have bee
available datasets (i.e. AID
the one of the ERD Challe
This extensive experime
best combination which ac
dataset an F1 score of 74.8
for the test dataset. This
pressive precision equal to
With respect to classic TA
the improvement ranged fr
mark, depending on the
used.
As a side result, the final
flexible library of several pa
ing modules that can be
sophisticated entity annota
brary to the public as an o

Categories and Subje

Tulip: Lightweight Entity Recognition and Disambiguation Using Wikipedia-Based Topic Centroids

Marek Lipczak
Faculty of Computer Science
Dalhousie University
Halifax, Canada
lipczak@cs.dal.ca

Arash Koushkestani
Faculty of Computer Science
Dalhousie University
Halifax, Canada
arash.koushkestani@dal.ca

Evangelos Milios
Faculty of Computer Science
Dalhousie University
Halifax, Canada
eem@cs.dal.ca

ABSTRACT

This article presents an ERD system Tulip, a submission to the ERD 2014: Entity Recognition and Disambiguation Challenge. The objective of the proposed system is to spot mentions of entities in a document and link the mentions to corresponding Freebase articles. To achieve it, Tulip prunes the set of entity candidates focusing on a core subset of related entities. The relationship strength is measured as a similarity to a topic centroid generated from entity features. Each entity is represented by an accurate and compact feature vector extracted from a category graph built based on information from 120 language versions of Wikipedia. Given the core set of accepted entities Tulip uses the Wikipedia-based feature vectors to extract more related entities from the document text. The challenge results: first prize in the long document track with F1 score of 0.74 confirms the effectiveness of our system. At the same, the system was faster than all other submissions with latency under 0.29 seconds.

Categories and Subject Descriptors

I.2.7 [Natural Language Processing]: Text analysis

an external knowledge base. This task is also known under the names of Entity Linking [19], Wikification [12] or more generally text annotation. Text annotation and interlinking documents with external knowledge bases is an interesting problem with many practical applications such as semantic search [1], faceted browsing [7], recommender systems [13], and text categorization [5]. The motivation of the ERD 2014: Entity Recognition and Disambiguation Challenge [3] was to advance the state of the art in the field for both short documents (e.g., search queries) and long documents (e.g., web pages). This article presents a ERD system Tulip which was a submission to the challenge and the recipient of the first prize in the long documents track.

ERD process is usually divided into two steps: *spotting* and *disambiguation*. In the first step, the system spots potential *mentions* of entities in text and links them to a list of *senses* which are the *candidate entities* that can be referred by the given mention. Each entity stored in system's data repository is represented by a list of *surface forms*. For example, Halifax is a surface form for a city in UK and Halifax Regional Municipality in Canada among others. In the second step, the system disambiguates the candidate entities

Microsoft Research

Entity Recognition and Disambiguation Challenge

Home | My Team | Related | Servers and Papers | Rules | Datasets | Short Track | Long Track | Discussion

Welcome to the 2014 Entity Recognition and Disambiguation Challenge!

We are excited to announce the 2014 Entity Recognition and Disambiguation (ERD) Challenge! The participating teams will have the opportunity to not only win cash prizes in the total amount of US$1,500, but also be invited to publish and present their results at a SIGIR 2014 workshop in Gold Coast, Australia, co-sponsored by Google and Microsoft.

NIST TAC Knowledge Base Population (KBP2014) Entity Linking Track

Entity Linking and Wikification Reading List

- collected and recommended by Heng Ji, highly biased and incomplete, suggestions are welcome

Overview Papers and Tutorials:

- Overview
- Task Definition
- Timeline
- Annotation Guideline
- Scorer
- Tools
- Data
- Reading List
- Mailing List
- Registration
- Organizing Committee
- TAC 2014
- Related Events

[9] Heng Ji, Joel Nothman and Ben Hachey. 2014. Overview of TAC-KBP2014 Entity Discovery and Linking Tas

[8] Dan Roth, Heng Ji, Ming-Wei Chang and Taylor Cassidy. 2014. Wikification and Beyond: The Challenges of En Computational Linguistics (ACL2014). description. slides pdf. slides pptx. book.

[7] Edgar Meij, Krisztian Balog and Dann Odijk. 2014. Entity Linking and Retrieval. Tutorial at WSDM2014, SIG

[6] Roberto Navigli and Andrea Moro. 2014. Multilingual Word Sense Disambiguation and Entity Linking. Tuto

[5] Heng Ji, Ralph Grishman and Hoa Trang Dang. 2011. Overview of the TAC 2011 Knowledge Base Populatio

[4] Heng Ji, Ralph Grishman, Hoa Trang Dang, Kira Griffitt and Joe Ellis. 2010. Overview of the TAC 2010 Know

[3] Heng Ji and Ralph Grishman. 2011. Knowledge Base Population: Successful Approaches and Challenges.

[2] P. McNamee and H.T. Dang. 2009. Overview of the tac 2009 knowledge base population track. Proc. Text An

[1] Wei Shen, Jianyong Wang, Jiawei Han. 2014. Entity Linking with a Knowledge Base: Issues, Techniques, and Solut

Simple stuff ;-)

1. spotting
2. searching
3. disambiguation
4. prune irrelevant stuff

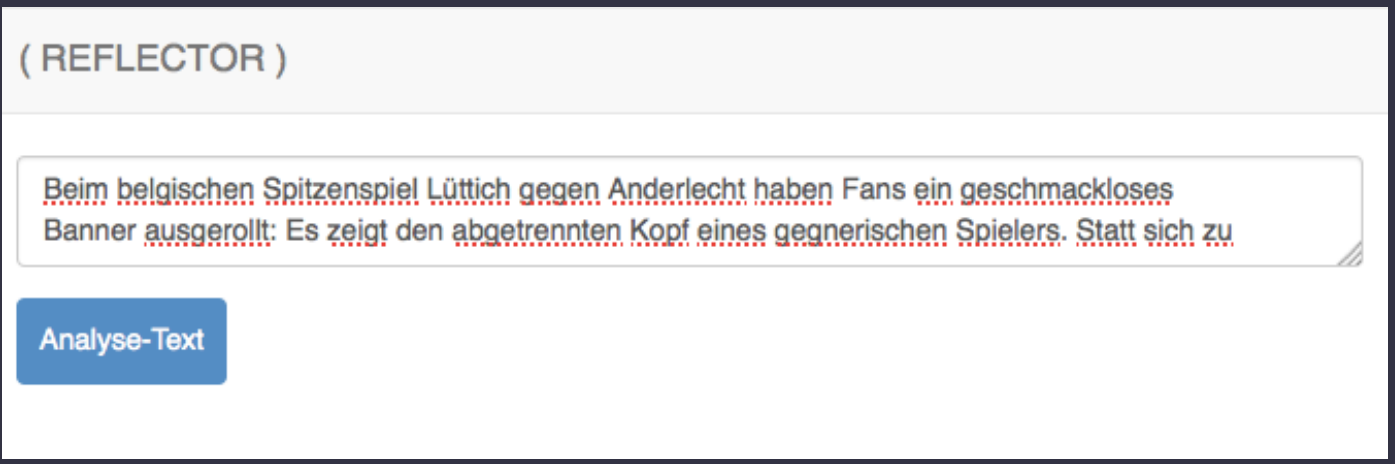
REFLECTOR OVERVIEW

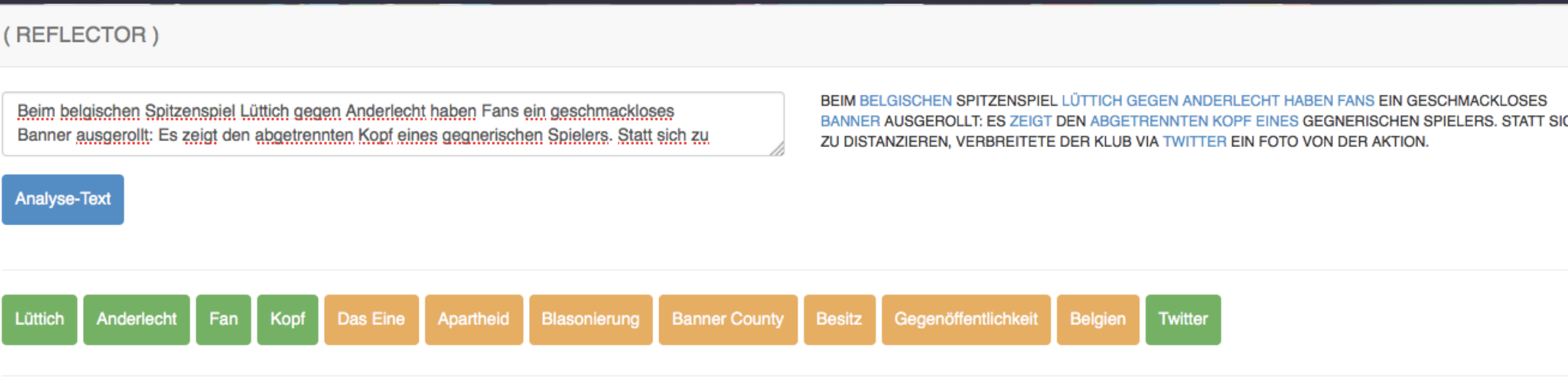
& TECHNOLOGIES


```
; the force
[org.clojure/clojure "1.6.0"]
[com.stuartsierra/component "0.2.2"]
[org.clojure/core.match "0.2.1"]
[org.clojure/clojurescript "0.0-2322"]
[org.clojure/core.async "0.1.346.0-17112a-alpha"]
; graph
[aysylu/loom "0.5.0"]
; IR
[clojure-opennlp "0.3.3"]
[clojurewerkz/elastisch "2.1.0"]
; Frontend
[om "0.7.3"]
[racehub/om-bootstrap "0.3.2"]
[prismatic/om-tools "0.3.6"]
;; Websockets
[com.taoensso/sente "1.2.0" :exclusions [org.clojure/clojure]]
;; Backend
[compojure "1.1.9"]
[http-kit "2.1.19"]
[clj-http "1.0.1"]
[ring "1.3.1"]
[ring/ring-defaults "0.1.1"]
;; data on the wire
[com.cognitect/transit-clj "0.8.247"]
[com.cognitect/transit-cljs "0.8.188"]
; logging
[org.clojure/tools.logging "0.3.0"]
[ch.qos.logback/logback-classic "1.1.1"]
```

REFLECTOR OVERVIEW

& TECHNOLOGIES





REFLECTOR OVERVIEW

& TECHNOLOGIES

PERSONS
JOHANN WOLFGANG VON GOETHE
LEOPOLD I. (BELGIEN)
LEOPOLD II. (BELGIEN)
LEOPOLD III. (BELGIEN)
ALBERT I. (BELGIEN)
FRANZ JOSEPH I.
MAXIMILIAN VON MONTGELAS
ALBERT II. (BELGIEN)

LOCATIONS
LÜTTICH
PARIS
BRÜSSEL
PRAG
GENT
MAASTRICHT
HUY (BELGIEN)
KERKRADE

ORGANISATIONS
SOFTM
COMARCH SOFTWARE UND BERATUNG
KBC BANK DEUTSCHLAND
ERGO VERSICHERUNG
MUSÉE DES TRANSPORTS EN COMMUN DU PAYS DE LIÈGE
BURGBAD
KAMPA (UNTERNEHMEN)
GROHE

UNDIRECTED WIKIPEDIA PATHS

Belgien Belgien_nach_Gemeinde Lüttich

Gegenöffentlichkeit Kommunikationswissenschaft Association_for_Education_in_Journalism_and_Mass_Communication Internationale_Organisation International_Committee_of_Military_Medicine Lüttich

Besitz Volkswirtschaftslehre Entwicklungsökonomik Desertec Internationale_Organisation International_Committee_of_Military_Medicine Lüttich

Blasonierung Heraldik Court_of_the_Lord_Lyon Edinburgh Europa_nach_Ort Lüttich

REFLECTOR OVERVIEW

& SYSTEM

storage

back

front

REPRESENT THE DATA

& DATA EXAMPLES


```
"_source": {
  "title": "Tiefurt",
  "text": "Tiefurt ist ein Ortsteil der Thüringen|
  "wiki_text": "{{Infobox Ortsteil einer Gemeinde
  "redirect": false,
  "redirect_page": null,
  "special": false,
  "stub": false,
  "disambiguation": false,
  "category": [
    "Stadtteil von Weimar",
    "Ehemalige Gemeinde (Weimar)",
    "Ort an der Ilm (Saale)",
    "Straßendorf"
  ],
  "link": [
    "Thüringen",
    "Weimar",
    ...
  ],
  "redirects": [],
  "disambiguations": [],
  "incoming_links": [
    "Ingeborg Stein",
    "Liste der Stadtteile von Weimar",
    "Georg Bleyer",
    ...
  ]
}
```

REPRESENT THE DATA

& DATA EXAMPLES

AMBIGUOUS

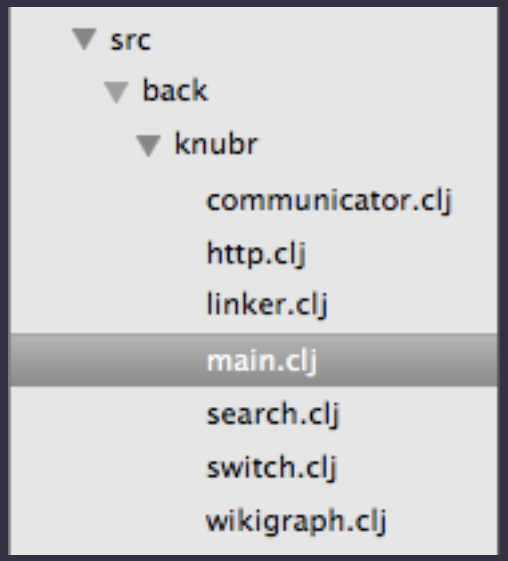

```
"docs": [
      {
        "id": "Opposition (Politik)",
        "text": [
          "Oppositionssprecherin",
          "Oppositionsjahre",
          "Oppositionskreisen",
          "Oppositionskraft",
          "oppositionellen Haltung",
          "Oppositionssprecher",
          "Oppositionen",
          "politischer Gegner",
          "Oppositionszeit",
          "Oppositionspolitiker",
          "Oppositionsorganisation",
          "Fundamentalopposition",
          "systemoppositionellen",
          "politische Gegner",
          "oppositionell",
```


UNIQUE


```
"response": {
    "numFound": 1,
    "start": 0,
    "docs": [
      {
        "id": "Opposition (Politik)",
        "text": [
          "Opposition (Politik)"
        ],
        "_version_": 1490128805088460800
      }
    ]
  }
}
```

A BIT CODE

& BACK



A BIT CODE

& BACK

- **main** starts components
- **http** is the external interface
- **communicator** sends massages
- **linker** does entity linking with solr
- **search** searches pages in ES
- **wikigraph** computes paths
- **switch** glues the core.async channels together

A BIT CODE

& BACK

```
16  (defn get-system [conf]
17    "Create system by wiring individual components so that component/start
18    will bring up the individual components in the correct order."
19    (component/system-map
20      :communicator-channels (comm/new-communicator-channels)
21      :communicator  (component/using (comm/new-communicator) {:channels :communicator-channels})
22      :linker-channels (linker/new-linker-channels)
23      :linker (component/using (linker/new-linker conf) {:channels :linker-channels})
24      :search-channels (search/new-search-channels)
25      :search   (component/using (search/new-search conf) {:channels :search-channels})
26      :graph-channels (graph/new-graph-channels)
27      :graph   (component/using (graph/new-graph conf) {:channels :graph-channels})
28      :http  (component/using (http/new-http-server conf) {:communicator :communicator})
29      :switchboard   (component/using (sw/new-switch) {:comm-chans :communicator-channels
30                                                       :search-chans :search-channels
31                                                       :linker-chans :linker-channels
32                                                       :graph-chans :graph-channels})))
33
34  (def system (get-system conf))
35
36  (defn -main [& args]
37    (log/info "Application starting")
38    (alter-var-root #'system component/start)
39    (log/info "Application started"))
```

A BIT CODE

& BACK

```clojure
(ns back.knubr.switch
  (:gen-class)
  (:require
    [clojure.tools.logging :as log]
    [com.stuartsierra.component :as component]
    [clojure.core.async :as async :refer [chan mult tap pipe]]))

(defrecord Switch [comm-chans search-chans linker-chans graph-chans]
  component/Lifecycle
  (start [component] (log/info "Starting Switchboard Component")
    (pipe (:query comm-chans) (:query search-chans))
    (pipe (:entity-details comm-chans) (:entity-details search-chans))
    (pipe (:graph comm-chans) (:graph graph-chans))
    (pipe (:graph-results graph-chans) (:graph-results comm-chans))
    (pipe (:linker comm-chans) (:linker linker-chans))
    (pipe (:linker-results linker-chans) (:linker-results comm-chans))
    (pipe (:query-results search-chans) (:query-results comm-chans))
    (pipe (:entity-details-results search-chans) (:entity-details-results comm-chans)))
  (stop [component] (log/info "Stop Switchboard Component")))

(defn new-switch [] (map->Switch {}))
```

A BIT CODE

& BACK

```
(def ring-defaults-config (assoc-in ring.middleware.defaults/site-defaults [:security :anti-forgery]
                                    {:read-token (fn [req] (-> req :params :csrf-token))}))
(defn- static-html [file-name] (content-type (resource-response file-name {:root "public"}) "text/html"

(defrecord Httpserver [conf communicator server]
  component/Lifecycle
  (start [component] (log/info "Starting HTTP Component")
    (defroutes my-routes  ; created during start so that the correct communicator instance is used
               (GET  "/"     [] (static-html "index.html"))
               (GET  "/chsk" req ((:ajax-get-or-ws-handshake-fn communicator) req))
               (POST "/chsk" req ((:ajax-post-fn communicator) req))
               (route/resources "/") ; Static files, notably public/main.js (our cljs target)
               (route/not-found "Page not found"))
    (let [my-ring-handler   (ring.middleware.defaults/wrap-defaults my-routes ring-defaults-config)
          server (http-kit-server/run-server my-ring-handler {:port (:port conf)})
          uri (format "http://localhost:%s/" (:local-port (meta server)))]
      (log/info "Http-kit server is running at" uri)
      (assoc component :server server)))
  (stop [component] (log/info "Stopping HTTP Server")
    (server :timeout 100)
    (assoc component :server nil)))

(defn new-http-server [conf] (map->Httpserver {:conf conf}))
```

A BIT CODE

& BACK

```
(def ring-defaults-config (assoc-in ring.middleware.defaults/site-defaults [:security :anti-forgery]
                                    {:read-token (fn [req] (-> req :params :csrf-token))}))
(defn- static-html [file-name] (content-type (resource-response file-name {:root "public"}) "text/html"

(defrecord Httpserver [conf communicator server]
  component/Lifecycle
  (start [component] (log/info "Starting HTTP Component")
    (defroutes my-routes  ; created during start so that the correct communicator instance is used
               (GET  "/"     [] (static-html "index.html"))
               (GET  "/chsk" req ((:ajax-get-or-ws-handshake-fn communicator) req))
               (POST "/chsk" req ((:ajax-post-fn communicator) req))
               (route/resources "/") ; Static files, notably public/main.js (our cljs target)
               (route/not-found "Page not found"))
    (let [my-ring-handler   (ring.middleware.defaults/wrap-defaults my-routes ring-defaults-config)
          server (http-kit-server/run-server my-ring-handler {:port (:port conf)})
          uri (format "http://localhost:%s/" (:local-port (meta server)))]
      (log/info "Http-kit server is running at" uri)
      (assoc component :server server)))
  (stop [component] (log/info "Stopping HTTP Server")
    (server :timeout 100)
    (assoc component :server nil)))

(defn new-http-server [conf] (map->Httpserver {:conf conf}))
```

A BIT CODE

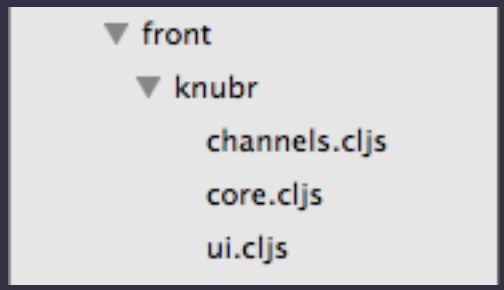
& BACK

```
(defrecord Search [conf channels conn native-conn]
  component/Lifecycle
  (start [component]
    (log/info "Starting Search Component")
    (let [conn (esr/connect (:es-address conf))]
      (run-query-loop (:query channels) (:query-results channels) conf conn)
      (run-entitydetails-loop (:entity-details channels) (:entity-details-results channels) conf conn)
      (assoc component :conn conn :native-conn native-conn)))
  (stop [component]
    (log/info "Stopping Search Component")
    (assoc component :conn nil :native-conn nil)))

(defn new-search [conf] (map->Search {:conf conf}))

(defrecord Search-Channels []
  component/Lifecycle
  (start [component] (log/info "Starting Search Channels Component")
    (assoc component
      :query (chan)
      :query-results (chan)
      :entity-details (chan)
      :entity-details-results (chan)
      ))
  (stop [component] (log/info "Stop Search Channels Component")
    (assoc component :query nil :query-results nil)))

(defn new-search-channels [] (map->Search-Channels {}))
```

A BIT CODE

& FRONT

- **core** handles communication and app state
- **ui** OM components
- **channels** just the definitions



A BIT CODE

& FRONT

```
(def app-state (atom {:query-string ""
                      :aggs         {:links {:buckets []}}
                      :abstractions (list)
                      :synonyms (list)
                      :links (list)
                      :ilinks (list)
                      :text    ""
                      :linkage (list)
                      :paths (list)}))

(def packer
  "Defines our packing (serialization) format for client<->server comms."
  (sente-transit/get-flexi-packer :json))
```

A BIT CODE

& FRONT

```
(defn- event-handler [{:keys [event]}]
  (match event
         [:chsk/state {:first-open? true}] (print "Socket established!")
         [:chsk/recv payload]
         (let [[msg-type msg] payload]
           (match [msg-type msg]
                  [:cmd/linker-result hits] (put! c/linker-result-channel hits)
                  [:cmd/entity-details-result hits] (put! c/entity-details-channel hits)
                  [:cmd/graph-result hits] (put! c/graph-result-channel hits)))
         :else (print "Unmatched event: %s" event)))
```

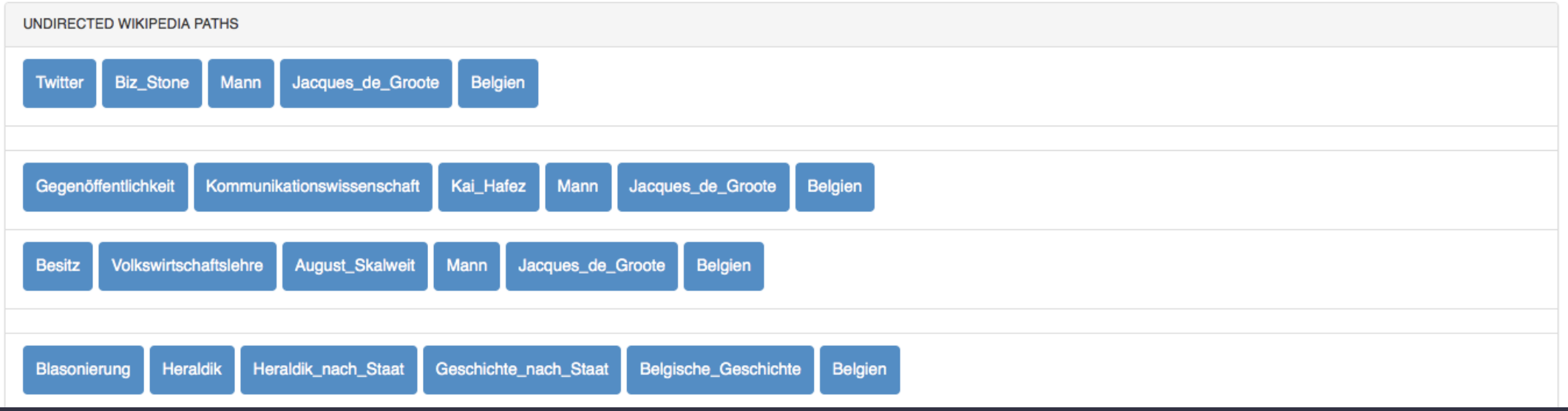
A BIT CODE

& FRONT

```
; Go baby
(go-loop []
         (let [linker-results (<! c/linker-result-channel)]
           (swap! app-state assoc :linker-results linker-results)
           (recur)))

(go-loop []
         (let [graph-results (<! c/graph-result-channel)]
           (swap! app-state update-in [:paths] conj (:path graph-results))
           (recur)))

(go-loop []
         (let [entity-details (<! c/entity-details-channel)
               document (:_source entity-details)]
           (swap! app-state assoc :abstractions (:sig_categories document))
           (swap! app-state assoc :synonyms (:like_title document))
           (swap! app-state assoc :links (:links document))
           (swap! app-state assoc :ilinks (:ilinks document))
           (swap! app-state assoc :persons (:persons entity-details))
           (swap! app-state assoc :locations (:locations entity-details))
           (swap! app-state assoc :companies (:companies entity-details))
           (recur)))
```

A BIT CODE

& FRONT




```
(defcomponent graph-result [state owner]
              (render [_]
                      (get-panel (map (fn [path-list]
                                        (d/li {:class "list-group-item"}
                                              (map (fn [x]
                                                     (b/button {:bs-style "primary"} x))
                                                   path-list)))
                                      (:paths state))
                                 "Undirected wikipedia paths")))
```

```
(om/root graph-result core/app-state {:target (. js/document (getElementById "graph-results"))})
```

LEARNINGS

& FIN

- **Structure** clojure code via components
- **Same** language full stack
- **Async** communication from front to back
- use favorite **datastores** with clojure
- learn about **wikipedia**

SHOULDERS

& FIN

https://github.com/elasticsearch/elasticsearch-river-wikipedia

https://github.com/OpenSextant/SolrTextTagger

https://github.com/stuartsierra/component

https://github.com/clojure/core.async

https://github.com/swannodette/om

https://github.com/matthiasn/BirdWatch