Constellation: Programming decentralised social networks

Anne-Marie Kermarrec, François Taïani
Summary

- **Constellation**: DSL for self-organising social networks
  - **Rapid** experimentation
  - **Composable**, recursive
  - Still under development

- **Benefits**
  - Not much in terms of language research (?)
  - Dist. Sys.: organise **design space**, help **exploration**
  - Not limited to social networks (embryomorphic eng.)
Towards a User-Centred Web

- Privacy + Personalisation ➔ Decentralisation

Alice

Facebook

Twitter
Gossip-based Social Networks

Builds on T-Man [JMB09], Vicinity [VS05]

- Each user = 1 node = a profile + a set of neighbours
- Metrics: “similarity” measure between nodes
- Goal: find $n$ closest neighbours (decentralised)
Gossip-based Social Networks

- Uses randomising layer (RPS, Cyclon)
- Periodically: each node $n$
  - *merges* RPS neighbours with clustering neighbours
  - *sorts* nodes in list based on similarity
  - *keeps* $k$ closest neighbours
Example: Overlap Metrics

- User profile: list of shapes
  - Alice loves heart, Bob prefers diamonds over hearts

1. exchange of neighbors lists
2. neighborhood optimization
Exploitation: recommendation

- E.g. (geo)recommendations

Other uses: search, query extension, news filtering
Can we build a (simple, elegant) language to rapidly experiment with these self-stabilising social networks?
Challenges

- Large design space (profile, metrics)
- Recursive behaviours (neighbourhood comp)
- Monolithic approaches / low reuse
Our Take: Constellation

- Simple, declarative, (composable)
- Predicting twitter subscriptions using past subs (:s)

```ruby
node User { 
  data :name 
  data :s 
  clusters :cosSu { |u| cos_sim(self->:s, u->:s) } 
  clusters :overSu { |u| overlap(self->:s, u->:s) } 
  clusters :rand { |u| rand } 
}
```

- Ruby-based prototype
- 2446 users crawled from twitter
Tackling more: Heterogeneity

- Different types of nodes, asymmetric similarities (*)

\[
\begin{align*}
\text{node Website} & \{ \text{data :} k \} \\
\text{node User} & \{ \\
& \quad \text{data :} i \\
& \quad \quad \text{clusters :} \text{sites} \text{ with Website} \{ \\
& \quad \quad \quad |s| \cos( \text{this->:i, s->:k} ) \} \\
\end{align*}
\]

(*) under development
Tackling more: Recursion

- Recursive clustering [BBGKL10, BFGKL10]
  - containment relationships
  - clustering applied to containee
  - neighbourhood computation
node Checkin {
  data : t
  clusters : similarSites { \(|c| \
    overlap(this->:t,c->:t) \}
}
node User {
  contains Checkin : c
  clusters : similarUsers { \(|u| \
    overlap(this->:c->:t,u->:c->:t) \}
  set_scope : similarUsers for : c
}
( under development)
Summary

- Constellation: DSL for self-organising social networks
  - Rapid experimentation
  - Composable, recursive
  - Still under development

- Benefits
  - Not much in terms of language research?
  - Dist. Sys.: organise **design space**, help **exploration**
  - Not limited to social networks (embryomorphemic eng.)

Some Refs: Gossip


Some Refs: Gossip


Some Refs: Gossip + SocNet


Some Refs: Frameworks


- [LTB08] Lin, S., Taiani, F., and Blair, G. S. (2008). Facilitating gossip programming with the gossipkit framework, DAIS’08, pages 238–252


T-Man in Action

(taken from [JMB09])

Result $\rightarrow$ structured overlay

Highly resilient against churn, partition (RPS)

Fast convergence (Swap)
Capturing Neighbourhood Comp

container node
(e.g. user, storage node)

primary container data
(user name etc.)

vnode data
(tags in query)

vertical data aggregation

aggregated container data

similarity computation
(container)

similarity computation
(vnodes)

F. Taiani
Capturing neighbourhood Comp

- **user** (container node)
- **query** (vnode)
- Alice’s queries (x & y)
- **similarity link**
- **random link**

**virtualised clustering** of Alice’s queries

**clustering layer of Alice’s queries, x & y**

**RPS of Alice’s queries, x and y (vnodes)**

**clustering layer of users** (container nodes)

**RPS layer of users** (container nodes)
Self-organising networks

To speed convergence each node \( n \)
- picks one neighbour \( i \) in clustering layer
- sends own list of neighbours, receives that of \( i \)
- \( i \) and \( n \) merge, sort, keep \( k \) closest neighbours

(1) peer list exchange
(2) merge and truncation