END-TO-END GRAPH MAPPER

Benjamin Billet, Mickaël Jurret, Didier Parigot and Patrick Valduriez
Startup which develop/market mobile social networks for micro-communities

Cross-platform (Android, iOS, web), specific-purpose (event, activities)
Triton Goals

ぽ Current state: A software factory for building customized social networking servers and databases, according to customers’ requirements

ぽ Configuration/specialization over development

ぽ Rapid and adaptable production

ぽ Scalability
Triton Goals

- Current state: A software factory for building customized social networking servers and databases, according to customers’ requirements
  - Configuration/specialization over development
  - Rapid and adaptable production
  - Scalability

- Next step: Building mobile clients as well and generalize the approach to various applications that manage linked data
Current Infrastructure

Object Graph Mapper
JSON objects <=> Java Entities <=> Graph DB

Graph Abstraction Layer
Type hierarchy, constraints and behaviors

Graph-specific languages
OrientQL, Cipher, Gremlin

OrientDB, Neo4j, Titan DB

React Web
Platform-specific code

Common code

react-native
Platform-specific code

Serveur Triton (v2)
Graph DB

Serveur Beepeers (v1)
Relational DB

FRONT-END

BACK-END
Typical Web and Mobile applications

⇒ Client-server

⇒ CRUD on a set of connected data
Typical Web and Mobile applications

- Client-server
- CRUD on a set of connected data
Typical Web and Mobile applications

- Client-server
- CRUD on a set of connected data

\( O_{GUI} \) UI Tree | Client Graph Mapping Operator
\( O_{CNM} \) Client Entity Graph Queries | Web Services Call Mapping Operator
\( O_{SNM} \) Server Entity Graph | Web Services Response Mapping Operator
\( O_{OGM} \) Server Entity Graph | GDBMS Mapping Operator

Data-driven
Graphs everywhere

Graph Database

**Event**
- **attrs**: {name, string}, {begin, date}, {end, date}, {location, string}
- **links**: {hasTalk, Talk, one-to-many}

**Talk**
- **attrs**: {name, string}, {begin, date}, {end, date}
- **links**: {hasSpeaker, User, one-to-many}

**User**
- **attrs**: {name, string}
- **links**: {followEvent, Event}
Graphs everywhere

Event event = new Event();
event.setId("zenithDay");
event.setName("Zenith Day");
event.setLocation("Saint Gely");
event.setBeginDate(new Date(2016, 9, 13));
event.setEndDate(new Date(2016, 9, 13));

User user = ...;
user.getFollowedEvents().add(user.getId());

Talk talk = new Talk();
talk.getSpeakers().add(user.getId());
talk.setBegin(new Date(0, 0, 0, 11, 0));
talk.setEnd(new Date(0, 0, 0, 11, 0));

event.getTalks().add(talk);
...

Server Code (Java)
Graphs everywhere

```java
Event event = new Event();
event.setId("zenithDay");
event.setName("Zenith Day");
event.setLocation("Saint Gely");
event.setBeginDate(new Date(2016, 9, 13));
event.setEndDate(new Date(2016, 9, 13));

User user = ...;
user.getFollowedEvents().add(user.getId());

Talk talk = new Talk();
talk.getSpeakers().add(user.getId());
talk.setBegin(new Date(0, 0, 0, 11, 0));
talk.setEnd(new Date(0, 0, 0, 11, 0));

event.getTalks().add(talk);
...```
Graphs everywhere

Event

User

Talk

Server Code (Java)

```java
Event event = new Event();
event.setId("zenithDay");
event.setName("Zenith Day");
event.setLocation("Saint Gely");
event.setBeginDate(new Date(2016, 9, 13));
event.setEndDate(new Date(2016, 9, 13));
User user = ...
user.getFollowedEvents().add(user.getId());
Talk talk = new Talk();
talk.getSpeakers().add(user.getId());
talk.setBegin(new Date(0, 0, 0, 11, 0));
talk.setEnd(new Date(0, 0, 0, 11, 0));
event.getTalks().add(talk);
...
```

Client State

```json
let store = {
  users: {
    bbillet: {
      id: 'bbillet',
      firstname: 'Benjamin',
      lastname: 'Billet',
      followEvents: [ 'zenithDay' ],
    },
  },
  event: {
    zenithDay: {
      id: 'zenithDay',
      name: 'Zenith Day',
      location: 'Saint Gely',
      beginDate: new Date(...),
      endDate: new Date(...),
      talks: [ {
        speakers: [ 'bbillet', ... ],
        begin: new Date(...),
        end: new Date(...),
      }, ... ],
    }, ...
  },
}
```
Graphs everywhere

Client State

```javascript
let store = {
  users: {
    bbillet: {
      id: 'bbillet',
      firstname: 'Benjamin',
      lastname: 'Billet',
      followEvents: [ 'zenithDay' ],
    },
  },
  event: {
    zenithDay: {
      id: 'zenithDay',
      name: 'Zenith Day',
      location: 'Saint Gely',
      beginDate: new Date(...),
      endDate: new Date(...),
      talks: [{
        speakers: [ 'bbillet', ... ],
        begin: new Date(...),
        end: new Date(...),
      }, ... ],
    },
  },
};
```

JSON Serialization

```javascript
[
  {
    "bbillet",
    "name": "Benjamin",
    "lastname": "Billet",
    "followEvents": [ "zenithDay" ]
  },
  {
    "zenithDay",
    "name": "Zenith Day",
    "location": "Saint Gely",
    "beginDate": 1473717600,
    "endDate": 1473717600,
    "talks": [{
      "speakers": [ "bbillet" ],
      "begin": 36000,
      "end": 37800
    }]
  }
]
```

Client UI (JSX)

```jsx
<View style={styles.container}>
  <View style={styles.row}>
    <Text>User name: </Text>
    {this.props.user.firstname}
    {this.props.user.lastname}
  </View>
  ...
</View>
```
Zoom in $\mathcal{O}_{OGM}$

Graph Database

Event
- **attrs**: `{name, string}, {begin, date}, {end, date}, {location, string}`
- **links**: `{hasTalk, Talk, one-to-many}`

Talk
- **attrs**: `{name, string}, {begin, date}, {end, date}`
- **links**: `{hasSpeaker, User, one-to-many}`

User
- **attrs**: `{name, string}`
- **links**: `{followEvent, Event}`
Zoom in $\mathcal{O}_{\text{OGM}}$: Mappings

**Event.class**
- **gType**: Event
- **attrs**: `{title, name}`, `{beginDate, begin}`, `{endDate, end}`, ...
- **links**: `{talks, out(hasTalk)}`

**Talk.class**
- **gType**: Talk
- **attrs**: `{title, name}`, `{beginDate, begin}`, `{endDate, end}
- **links**: `{speakers, out(hasSpeaker)}`, `{event, in(Event)}`

**User.class**
- **gType**: User
- **attrs**: `{name, name}
- **links**: `{followedEvents, out(followEvent)}`, `{followedTalks, out(hasTalk, out(followEvent))}`
Zoom in $\mathcal{O}_{\text{OGM}}$: Mappings

Event.class
gType: Event
attrs: {title, name}, {beginDate, begin}, {endDate, end}, ...
links: {talks, out(hasTalk)}

Talk.class
gType: Talk
attrs: {title, name}, {beginDate, begin}, {endDate, end}
links: {speakers, out(hasSpeaker)}, {event, in(Event)}

User.class
gType: User
attrs: {name, name}
links: {followedEvents, out(followEvent)},
       {followedTalks, out(hasTalk, out(followEvent))}

Server Code
(Java)

class User
{
    private String name;
    private Set<Event> followedEvents;
    private Set<Talk> followedTalks;
}

Current State & Problems

➡ Current state: Prototypal implementation based on OrientDB, GraphQL, redux, react-native
➡ A lot of custom code for the client-side
➡ Technical problems: graph databases heterogeneity and instability
Current State & Problems

- Current state: Prototypal implementation based on OrientDB, GraphQL, redux, react-native
  - A lot of custom code for the client-side
  - Technical problems: graph databases heterogeneity and instability

- Ongoing research problems:
  - Complex mappings, identifying common mapping patterns in actual applications
  - Horizontal concerns: access control, transactions
  - Common query language for each layer (SPARQL?)
  - Distributed systems? Non-graph databases?
Zoom in $\mathcal{O}_{\text{CNM}}$ and $\mathcal{O}_{\text{SNM}}$: Network

Client

```
GET /graph HTTP/1.1
...
{ type: Talk, id: 1 }
```

```json
{
    type: Talk, id: 1,
    title: My Lecture
    evaluated: [{
        type: Evaluation, id: 2,
        like: 1, comment: "...",
    }]
}
```

Server

```
select from Talk where id = 1
```

nodes {type=Talk, id=1, ...} and {type=Evaluation, id=2, ...} are added to the client graph
Zoom in $\mathcal{O}_{\text{CNM}}$ and $\mathcal{O}_{\text{SNM}}$: Network

**insert into** Evaluation
values (like = 1, comment = 'blah', ...)
links (evaluated = (select from Talk
where id = 5), (select from User
where id = 6))

**Client**

```
PUT /graph HTTP/1.1
...
{
    type: Evaluation,
    like: 1,
    comment: "blah",
    ...
    evaluated: [{
        type: Talk, id: 5
    }],
    evaluator: [{
        type: User, id: 6
    }],
}
```

**Server**

```
{
    type: Evaluation, id: 7
}
```

node `{type=Evaluation, id=4, ...}` is added to the client graph
Full Graph Schema: Root Types

SocialEntity
- contentID:STRING
- creationDate:DATE
- isActive:BOOLEAN
- updateDate:DATE
- version:LONG

Agent
- isAdministrable
- isAttachment
- isCommentable
- isDislikeable
- isFilterable
- isLikeable
- isRankable
- isReportable
- isShareable

Content
- isRankable
- isReportable

Interaction
- creationDate:DATE
- updateDate:DATE

SocialRelationship
- status:STRING

Report
- reason:STRING
- reported:Reportable
- reporter:User

Invite
- invited:User
- inviter:User
- whatToFollow:Agent

BelongsTo

Attachment

Full Graph Schema: Social Entity Types