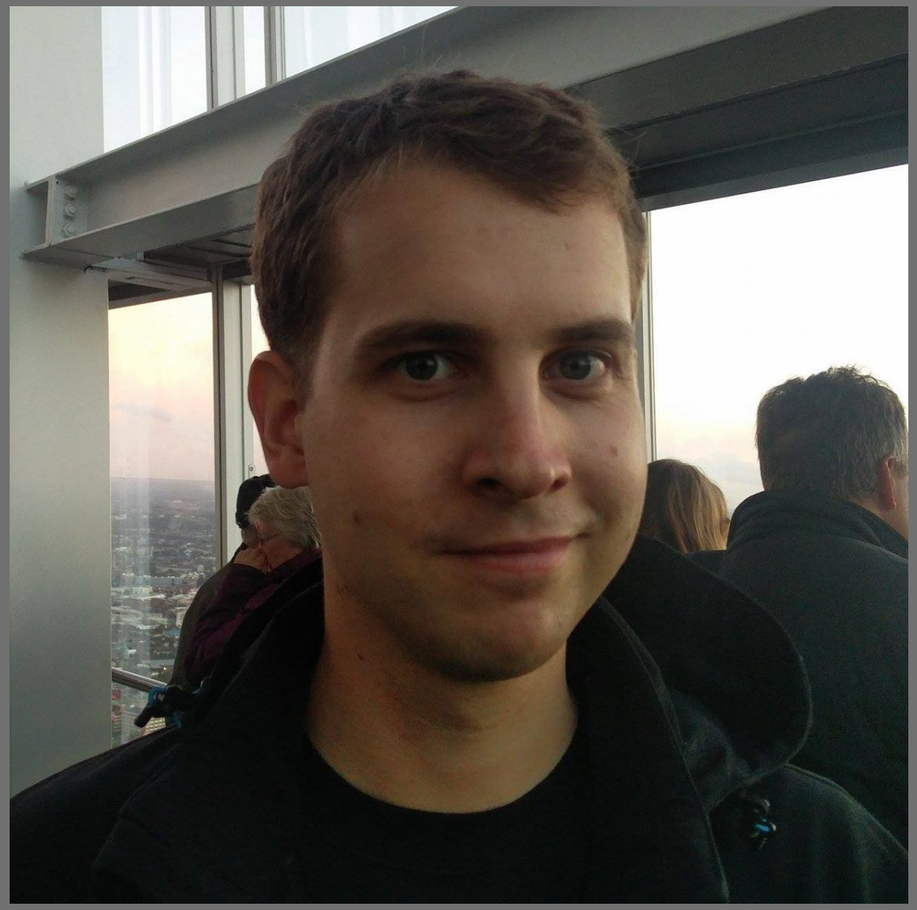
Redefining

Accessing graphs

About me

- Tomasz Pluskiewicz
- Zazuko GmbH
- Interests
  - Semantic Web
  - REST APIs
  - Hydra CG

/tpluscode
@tpluscode



The challenge

RDF is hard
RDF/JS is also hard
* or at least verbose
* contrary to what some will tell you[1]
* especially for novices

How do we get new devs into the RDF(/JS) space?

```
import * as Schema from '@rdfine/schema'

namespace Foaf {
  export interface Person {
    name: string
    avatar: Schema.Image
  }
}
```

[1] https://www.rubensworks.net/blog/2019/10/06/using-rdf-in-javascript/

Prior work

1. Romantic Web
2. JsonLD.Entities

— Rigid structures

— Destructive conversion

— Impendace mismatch

@tpluscode/rdfine[2]

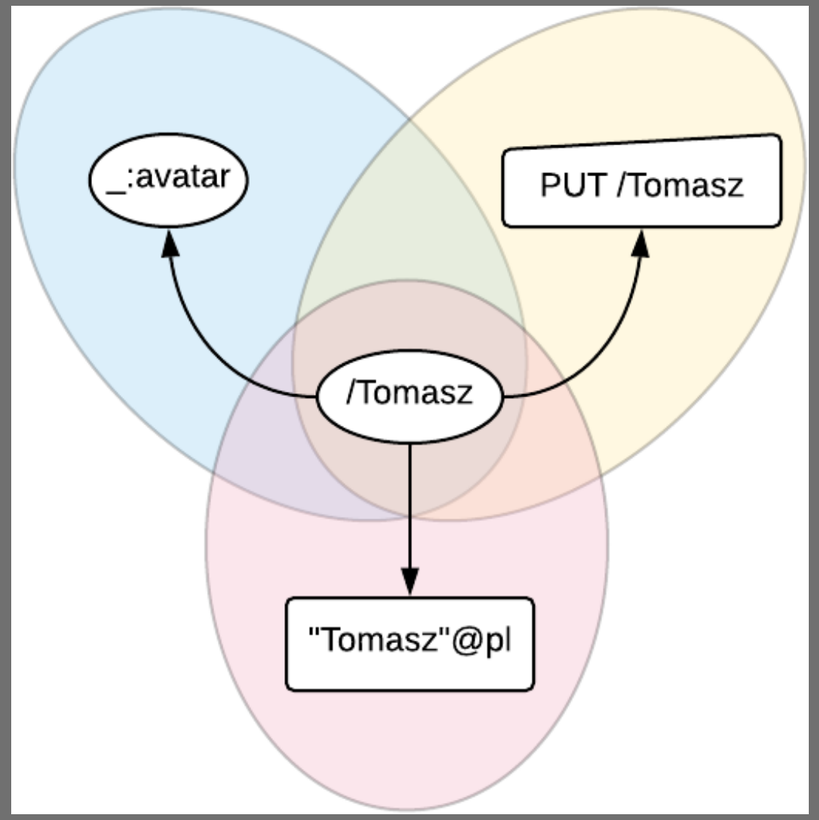
- Familiar JS objects
- Not detached from underlying RDF/JS model
  - A `DatasetCore` behind the scenes
- Uses clownface[3] by Thomas Bergwinkl
- TypeScript w/decorators
  - but usable with Babel

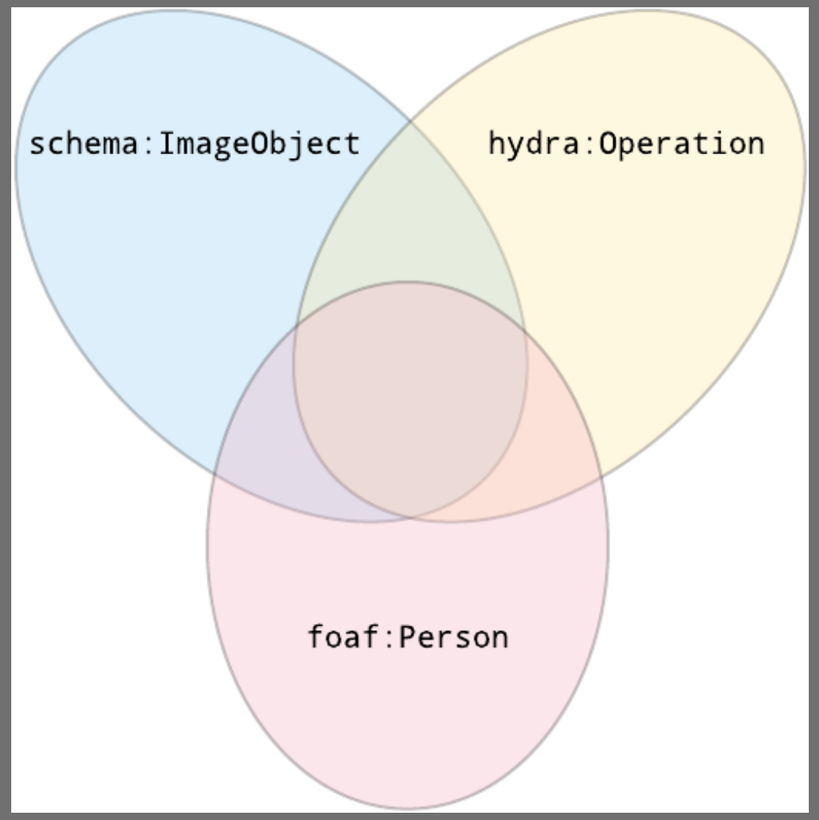
```
import rdf from '@rdfjs/dataset'
import namedNode from '@rdfjs/data-model'
import { RdfResourceImpl } from '@tpluscode/rdfine'
import * as Foaf from '@rdfine/foaf'
import { turtle } from '@tpluscode/rdf-string'

const dataset = rdf.dataset()
const person = RdfResourceImpl._create<Foaf.Person>({
  dataset,
  term: namedNode('Tomasz'),
})

// immediately reflected in dataset
person.name = "Tomasz Pluskiewicz"

/*
  </Tomasz> foaf:name "Tomasz Pluskiewicz" .
*/
turtle`${dataset}`
```

[2] https://npm.im/@tpluscode/rdfine

[3] https://npm.im/clownface

```
</Tomasz> a foaf:Person ;
  schema:photo [
    a schema:ImageObject ;
    schema:contentUrl
      "/profile_images/684777938785558529/iw_oqgCU_400x400.jpg"
  ] ;
  foaf:name "Tomasz Pluskiewicz" ;
  hydra:operation [
    hydra:method "PUT"
  ].
```


_:avatar
PUT /Tomasz
/Tomasz
"Tomasz"@pl


schema:ImageObject
hydra:Operation
foaf:Person

```
import { foaf, schema } from '@tpluscode/rdf-ns-builders'
import {
  Constructor, namespace, property, RdfResource
 } from '@tpluscode/rdfine'
import * as Schema from '@rdfine/schema'
import * as Foaf from '@rdfine/foaf'

export function Person<B extends Constructor>(resource: B) {
  @namespace(foaf)
  class P extends resource implements Foaf.Person {
    @property.literal()
    name: string

    @property.resource({
      path: schema.photo,
    })
    avatar: Schema.Image
  }

  return P
}

Person.shouldApply = (res: RdfResource) =>
  res.types.has(foaf.Person)
```

```
import {
  Constructor, namespace, property, RdfResource
} from '@tpluscode/rdfine'
import { schema } from '@tpluscode/rdf-ns-builders'
import * as Schema from '@rdfine/schema'

export function Person<B extends Constructor>(resource: B) {
  @namespace(schema)
  class SI extends resource implements Schema.ImageObject {
    @property.literal()
    contentUrl: string

    @property.resource()
    thumbnail: Schema.ImageObject
  }

  return SI
}

SchemaImage.shouldApply = (res: RdfResource) =>
  res.types.has(schema.ImageObject)
```

```
import { hydra } from '@tpluscode/rdf-ns-builders'
import {
  Constructor, namespace, property, RdfResource
} from '@tpluscode/rdfine'
import * as Hydra from '@rdfine/hydra'

export function Operation<B extends Constructor>(resource: B) {
  @namespace(hydra)
  class HO extends resource implements Hydra.Operation {
    @property.literal({
      initial: "GET",
    })
    method: string

    invoke() {
      // do the HTTP request
    }
  }

  return HO
}

Operation.shouldApply = (res: RdfResource) =>
  res._selfGraph.in(hydra.operation)
```

Unit test friendly

- Unit tests without RDF dependency
  - Easier to understand
  - Easier to set up
  - Easier to manage
  - Self-contained

```
import Processor from '../src/PersonProcessor'

describe('Person processor', () => {
  it('does its thing', () => {
    // given
    const person = {
      name: 'John Doe',
      photo: {
        contentUrl: 'http://foo/bar',
      }
    }

    // when
    Processor.thing(person)

    // then
    expect(Processor).toHaveDoneIt()
  })
})
```

Next up

Reusable packages with pre-built classes

```
import Skos from '@rdfine/skos'
import Rdfs from '@rdfine/rdfs'
import Hydra from '@rdfine/hydra'
import { RdfResourceImpl } from '@tpluscode/rdfine'

RdfResourceImpl.factory.add(Skos)
RdfResourceImpl.factory.add(Rdfs)
RdfResourceImpl.factory.add(Hydra)
```

More tools

- Used by @tpluscode/rdfine
  - clownface by Thomas Bergwinkl
- Some more helpful libraries I build
  - @tpluscode/rdf-ns-builders
  - @tpluscode/rdf-string
  - @tpluscode/sparql-builder

Thank you