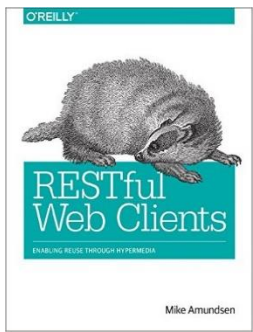
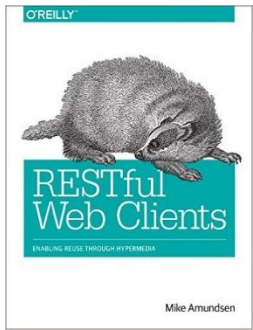
The Representor Pattern

Mike Amundsen
API Academy
@mamund

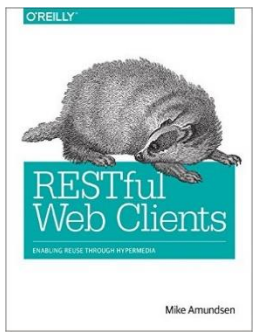


Highlights

- XML or JSON: Pick a Side!
- When Formats are Negotiable
- The Representor Pattern
- Server-Side Representor
- Summary

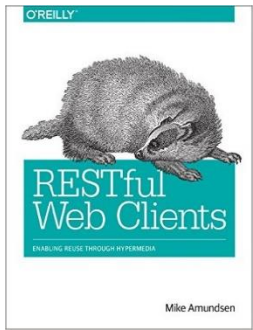


XML or JSON: Pick a Side!



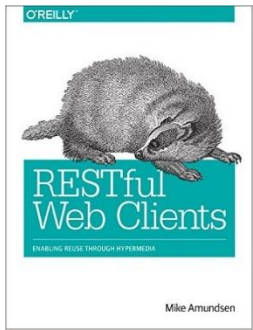
XML or JSON: Pick a Side!

- XML is Dead, long live XML?
- The New kids in town
  - Atom (2006)
  - HAL (2011)
  - Collection+JSON (2011)
  - Siren (2012)
  - UBER (2014)
  - Mason, JSON API, CPHL, etc.



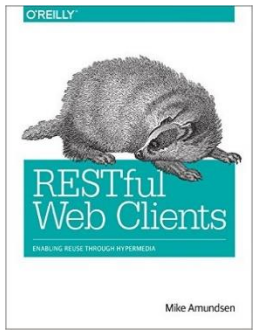
Negotiating

- The Fallacy of “The Right Format”
- Reasons for “Just One”
  - Cheaper
  - Easier
  - Controlled

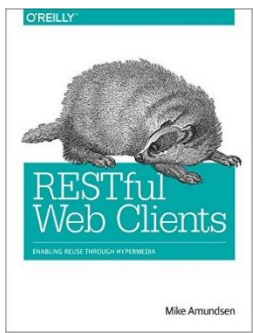


Negotiating

- The Fallacy of “The Right Format”
- Reasons for “Just One”
  - Cheaper
  - Easier
  - Controlled
- What if we could make supporting multiple formats:
  - Cheaper
  - Easier
  - Controlled

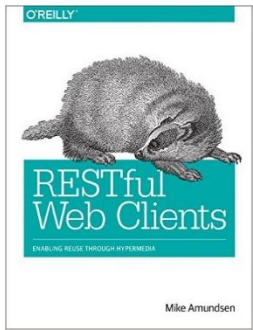


The Representor Pattern



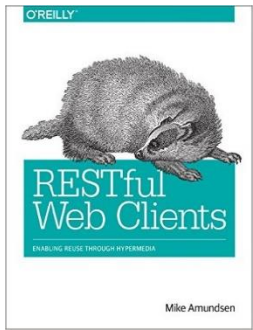
Implementing Representor Pattern

- Separate Form(at) from Functoin
- Use a Selecting Algorithm
- Implement a Dispatch Routine
- Create Transformers for Each Format
- Rely on Shared Model for All Transformers



Separate Format from Function

- DORR (again)
  - Data
  - Object
  - Resource
  - **Representation**
- What you send vs. How you “package” it



Selection Algorithm (context)

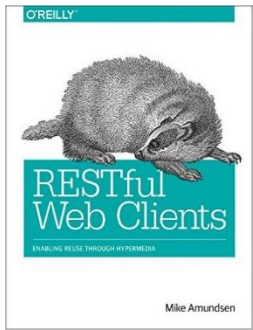
- HTTP Accept
- HTTP Content-Type

A typical client request might be:

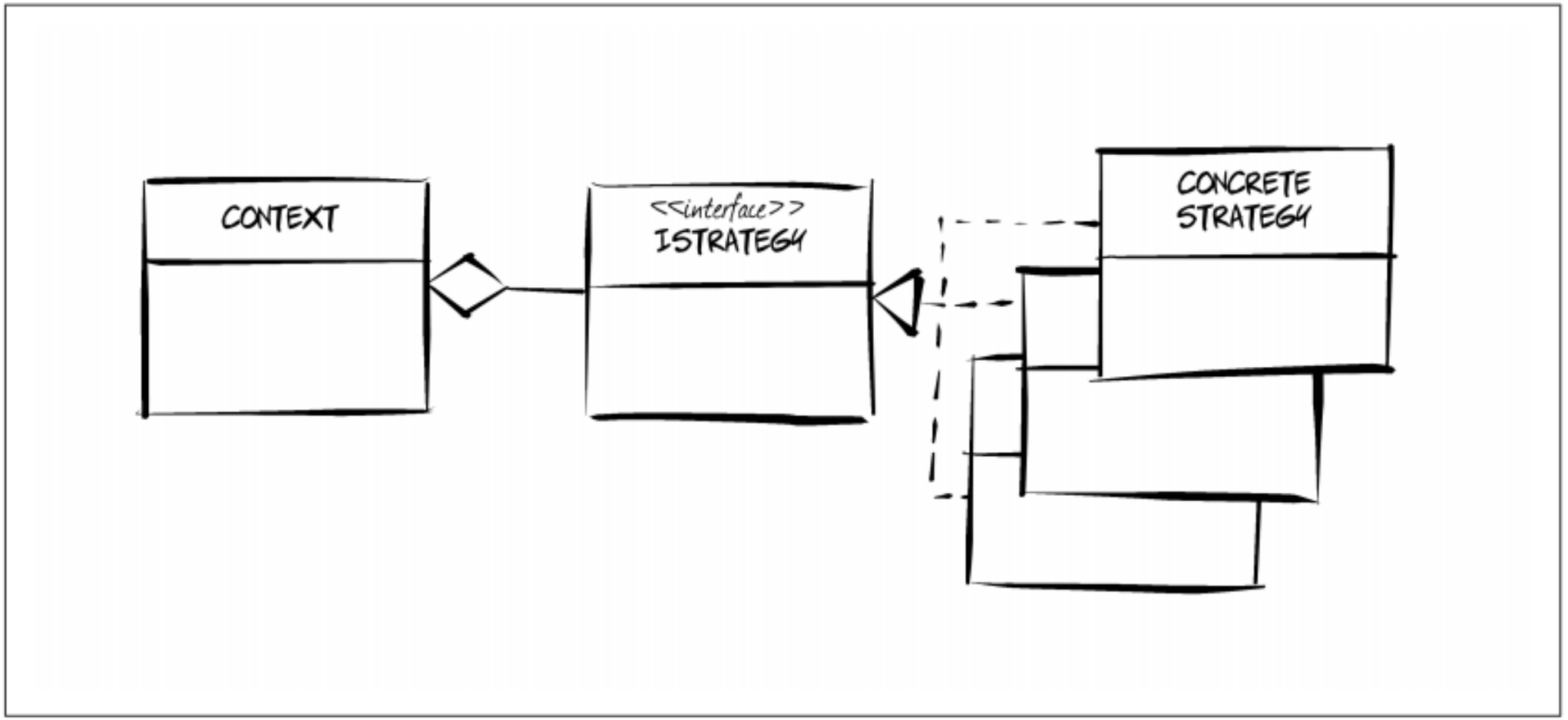
```
GET /users HTTP/1.1
Accept: application/vnd.hal+json, application/vnd.uber+json
...
```

And, for a service that supports HAL but does not support UBER, the response would be:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.hal+json
...
```



Strategy Pattern(GoF)



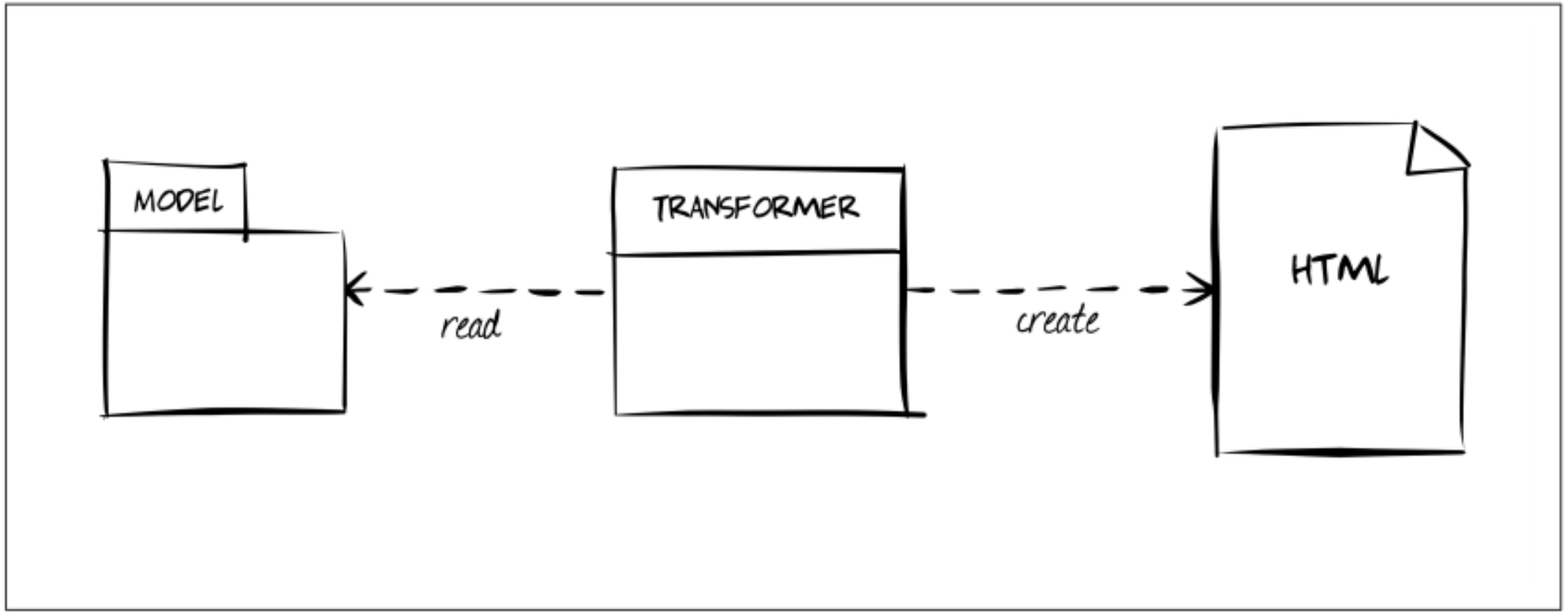
Dispatch (IStrategy)

```
18 function main(object, mimeType, root) {
19   var doc;
20
21   // clueless? assume JSON
22   if (!mimeType) {
23     mimeType = "application/vnd.collection+json";
24   }
25
26   // dispatch to requested representor
27   switch (mimeType.toLowerCase()) {
28     case "application/json":
29       doc = json(object, root);
30       break;
31     case "application/vnd.collection+json":
32       doc = cj(object, root);
33       break;
34     case "application/representor+json":
35       doc = repjson(object, root);
36       break;
37     default:
38       doc = cj(object, root);
39       break;
40   }
41
42   return doc;
43 }
```

Cj Representor (ConcreteStrategy)

```
14 function cj(object, root) {
15
16   var rtn = {};
17   rtn.collection = {};
18
19   rtn.collection.version = "1.0";
20   rtn.collection.href = root.replace(/^\/\//,"http://")||"";
21
22   for(var o in object) {
23     rtn.collection.title = getTitle(object[o]);
24     rtn.collection.links = getLinks(object[o].actions);
25     rtn.collection.items = getItems(object[o],root);
26     rtn.collection.queries = getQueries(object[o].actions);
27     rtn.collection.template = getTemplate(object[o].actions);
28
29     if(object.error) {
30       rtn.collection.error = getError(object.error);
31     }
32   }
33
34   return JSON.stringify(rtn, null, 2);
35 }
36
```

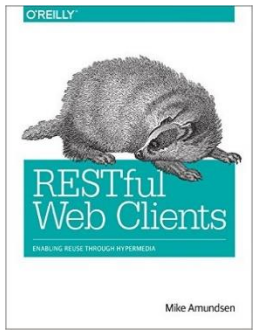
Transform View Pattern (Fowler)



Web Service Transition Language (WeSTL)

WeSTL - Web Service Transition

Table of Contents



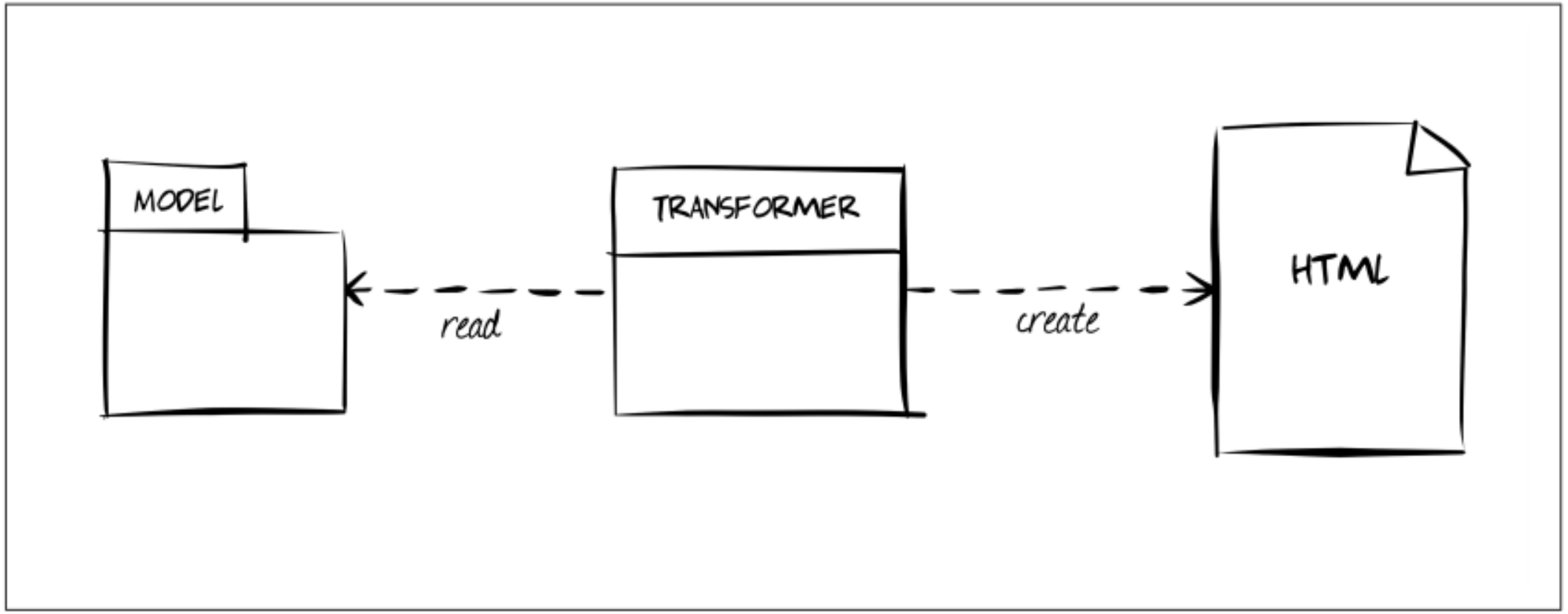
WeSTL

```
{
  "name": "searchLink",
  "description" : "Search page",
  "type": "safe",
  "action": "read",
  "kind": "search",
  "target": "list menu",
  "prompt": "Search",
},
{
  "name": "searchForm",
  "description" : "Search for content",
  "type": "safe",
  "action": "read",
  "kind": "search",
  "target": "list form",
  "prompt": "Search",
  "inputs": [
    {
      "name": "text",
      "prompt": "Search Text",
      "value": "",
      "required" : true
    },
    {
      "name": "external",
      "prompt": "External Search?",
      "value": "",
      "required: true,
      "suggest": [{"value":"true"},{"value":"false"}]
    }
  }
}
```

Transitions.js

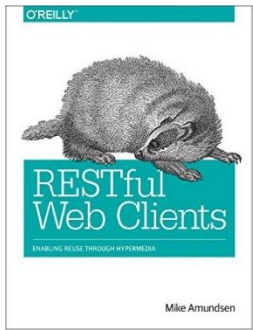
```
109  trans.push({
110    name : "addLink",
111    type : "safe",
112    action : "read",
113    kind : "task",
114    target : "list",
115    prompt : "Add task"
116  });
117  trans.push({
118    name : "addForm",
119    type : "unsafe",
120    action : "append",
121    kind : "task",
122    target : "list",
123    prompt : "Add task",
124    inputs : [
125      {name : "title", prompt : "Title"},
126      {name : "completed", prompt : "Complete", value : "false"}
127    ]
128  });
```

Transform View Pattern (Fowler)



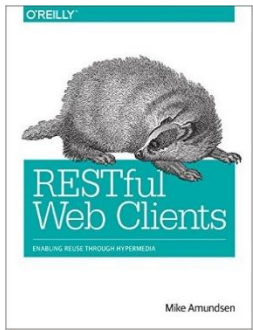
XML or JSON?

- Many options available today
  - Atom (2006)
  - HAL (2011)
  - Collection+JSON (2011)
  - Siren (2012)
  - UBER (2014)
  - Mason, JSON API, CPHL, etc.
- Expect more to come, esp. due to IoT



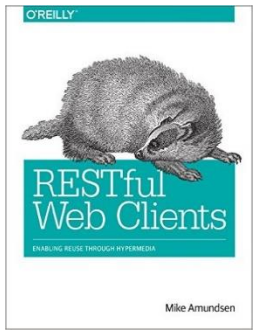
Negotiating is the Key

- The Fallacy of “The Right Format”
- Make supporting multiple formats:
  - Cheaper
  - Easier
  - Controlled



Implementing Representor Pattern

- Separate Form(at) from Function
- Use a Selecting Algorithm (accept & content-type)
- Implement a Dispatch Routine (representor.js)
- Create Transformers for Each Format (cj.js)
- Rely on Shared Model for All Transformers (wstl.js)



The Representor Pattern

Mike Amundsen
API Academy
@mamund