Restful Objects

A hypermedia API for
domain object models

The obligatory "who's this bloke?" slide

- Freelance consultant, dev, trainer
  - Java, .NET
- Been banging on about Naked Objects for years
  - even though almost no-one seems to "get it"
  - still working at the Irish government (who do)
- Written a couple of books
  - Domain Driven Design using Naked Objects
  - A UML modelling book for TogetherJ (remember that?)
- Do open source stuff
  - Committer on Apache Isis, Restful Objects.NET
  - Author of the "Restful Objects" spec

Restful Objects … what's the deal?

- So, it's a spec
  - open source, Creative Commons
  - http://restfulobjects.org
- JSON representations, over HTTP
- Two open source server-side impls:
  - RestfulObjects.NET
  - Apache Isis (JVM)
- Some Javascript clients have been hacked together
  - a Backbone/Javascript demo
  - a JQueryMobile demo
  - an internal app for managing sales pipeline

Origins

- RO came about from me thinking about traversable graphs of domain objects
- Also have graphs of resources, on the web
- Seemed that there ought to be some sort of correspondence between these two different “graphs of stuff”?

Need a way of addressing objects

For example:

- http://localhost:8080/objects/customer/123
  - to address a customer, id=123
- http://localhost:8080/objects/order/123~3
  - to address the 3rd order placed by that customer

Resources don't have to address entities…

- … could address objects that represent application state

For example:

- http://localhost:8080/objects/basket/a4b7511 6-cff4-4440-a77c-abb3cacb5091

represents a shopping basket, keyed with a GUID

Need a way of representing objects

- Options:
  - XML with proprietary schema
  - XHTML and microformats
  - JSON
  - Atom or similar?
- Flavour of the month seems to be JSON
  - certainly lots of client-side support
- Though need to figure out how to represent links between resources
  - to walk the graph

Need a way of link representations

```
{
 "rel": "...",
 "href": "http://localhost:8080/objects/order/123",
 "type": "application/json;...",
 "method": "GET",
 "title": "Order #123",
 "arguments": { ... },
 "value": ...
}
```

More than just GET

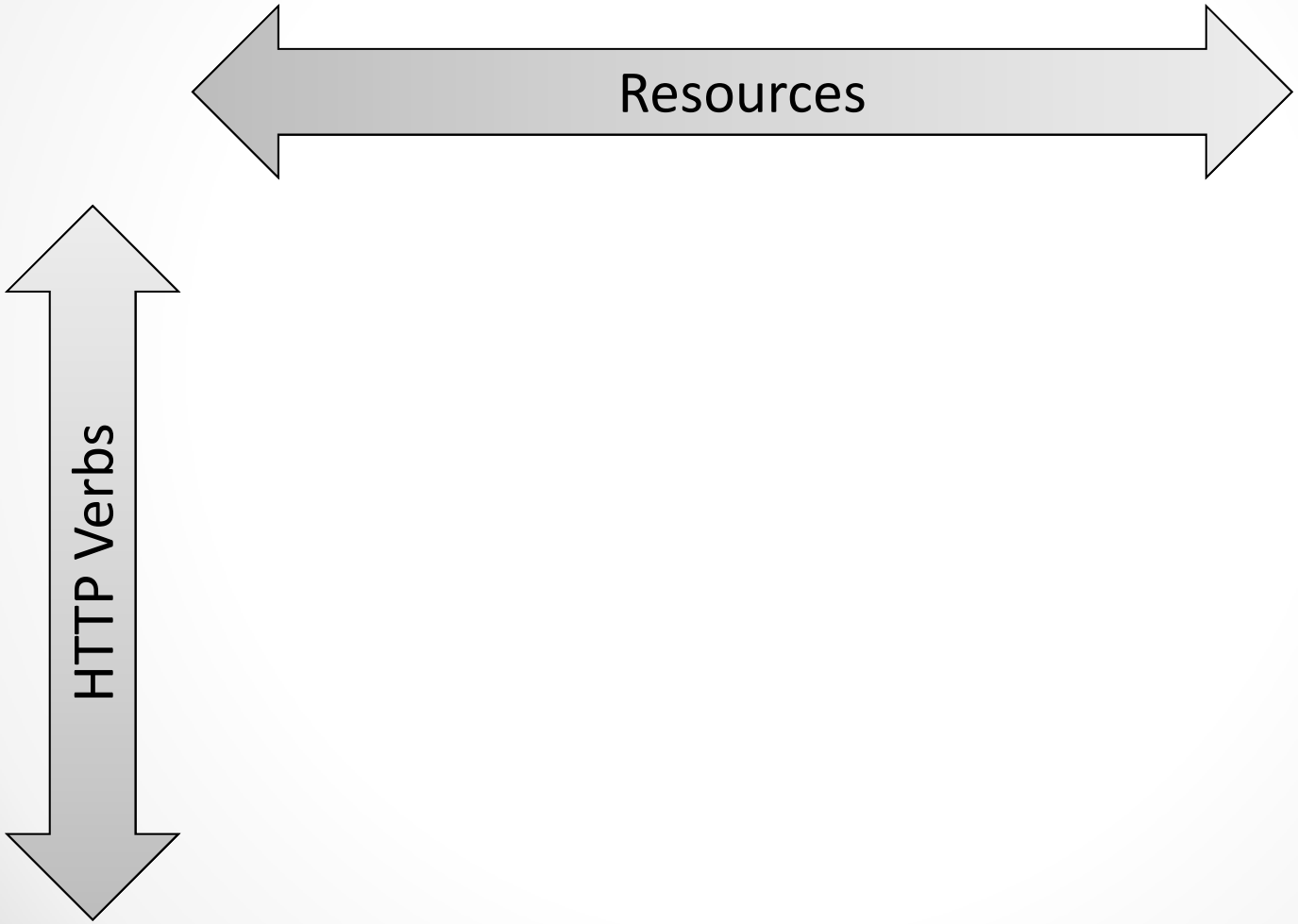
- Not just about retrieving resources
- HTTP defines a set of verbs
  - GET, PUT, DELETE, POST
- Use to manipulate the state of the underlying domain objects

Sub-resources provide finer-grained control

For example:

- http://localhost:8080/objects/customer/123/properties/surname
  - addresses the surname property of customer, id=123
- http://localhost:8080/objects/order/123~3/collections/lineItems
  - addresses the lineItems collection of an order

The spec defines which resources can be accessed by which verbs



Objects, properties, collections

URL HTTP VERB	Objects/ {Dtype}/{IID}	Objects/ {Dtype}/{IID}/ Properties/{Property}	Objects/ {Dtype}/{IID}/ Collections/{Collection}
GET	object summary, member summary, property values	property details and value	collection details and content
PUT	update or clear multiple property values	update or clear value	add object (if set semantics)
DELETE	delete object	clear value	remove object
POST	n/a – 405	n/a – 405	add object (if list semantics)

- {Dtype} is the domain type, eg. "customer"
- {IID} is the instance identifier, eg. "123"

But it's the domain object's *behaviour* that's key

- In Naked Objects, we talk about behaviourally complete objects
  - domain objects have state, sure
  - but they have behaviour also
  - it's OO like your mother taught you
- Otherwise we're just building a data browser / CRUD system
  - and what's the ~~fun~~ value in that?

Also actions and action invocation

URL / HTTP VERB	Objects/ {Dtype}/ {IID}	Objects/ {Dtype}/ {IID}/ Properties/ {Property}	Objects/ {Dtype}/ {IID}/ Collections/ {Collection}	Objects/ {Dtype}/ {IID}/ Actions/ {Action}	Objects/ {Dtype}/ {IID}/ Actions/ {Action}/ invoke
GET	object summary, member summary, property values	property details and value	collection details and content	action prompt	invoke (if query only)
PUT	update or clear multiple property values	update or clear value	add object (if set semantics)	n/a – 405	invoke (if idempotent)
DELETE	delete object	clear value	remove object	n/a – 405	n/a – 405
POST	n/a – 405	n/a – 405	add object (if list semantics)	n/a – 405	invoke (any)

The point being…

- … that this set of resources can be used to expose *any* domain model
  - a bit like how a UML class diagram can represent any domain
  - a bit like how an ORM can map any domain
  - domain-agnostic
- In other words: uniform access to the state and behaviour of domain objects
  - uniform access being a key principle of REST

Becoming a RESTafarian

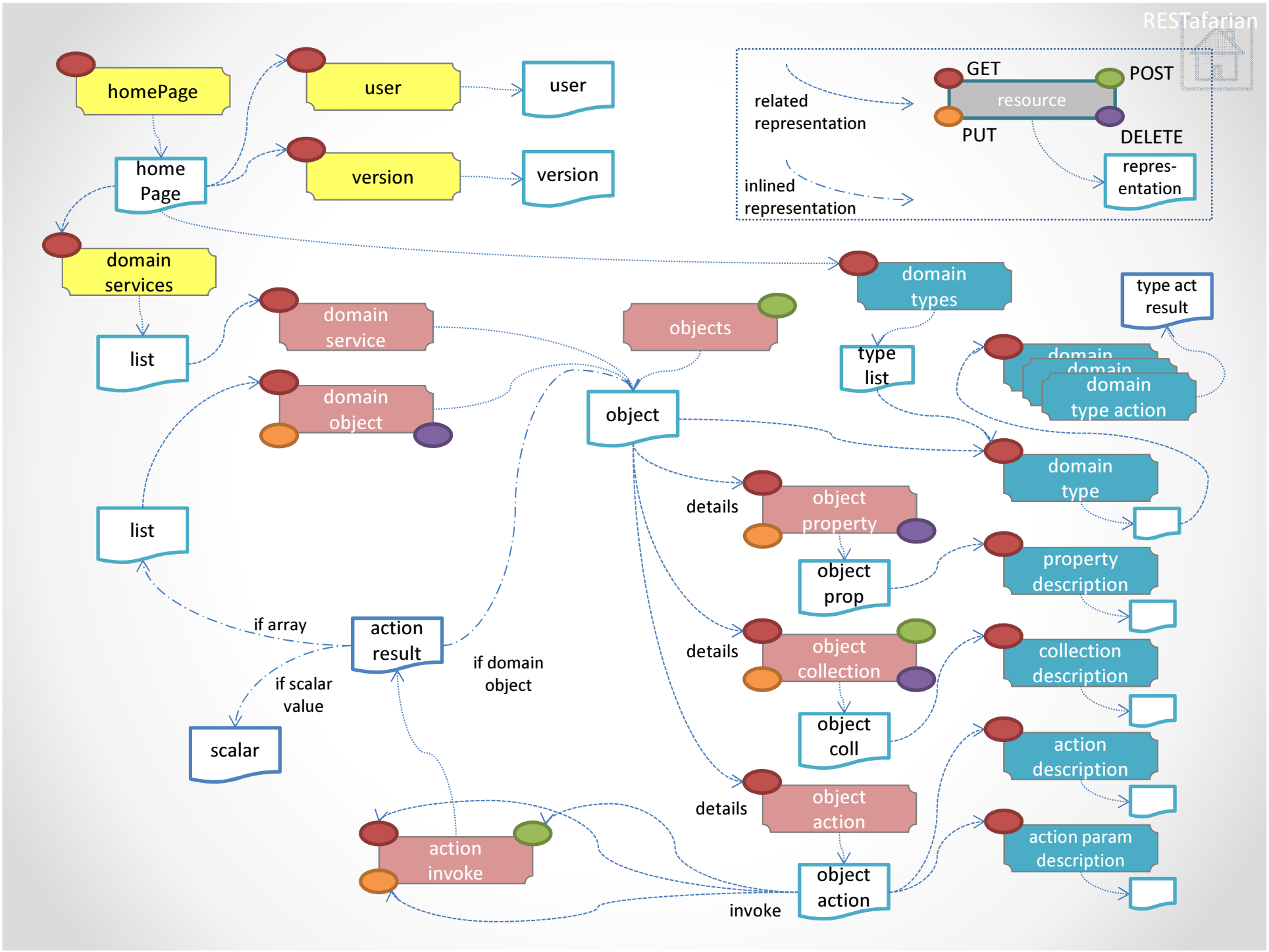
- The Restful Objects spec tries hard to live up to its name
  - define a RESTful system
- In other words, it gets into such matters as:
  - HATEOAS
  - media types and content negotation ("conneg")
  - link relationships
  - HTTP request and response headers
  - HTTP response codes (regular and arcane)
  - Caching

Hypermedia APIs

- Of all the RESTful principles, HATEOAS is the most important
  - "hypertext as the engine of application state"
- In other words, a resource's representations link in turn to other resources
  - as <FORM> and <A HREF=…> do in HTML
- All the RO representations link in this way…
  - there are no cul-de-sacs

Home page

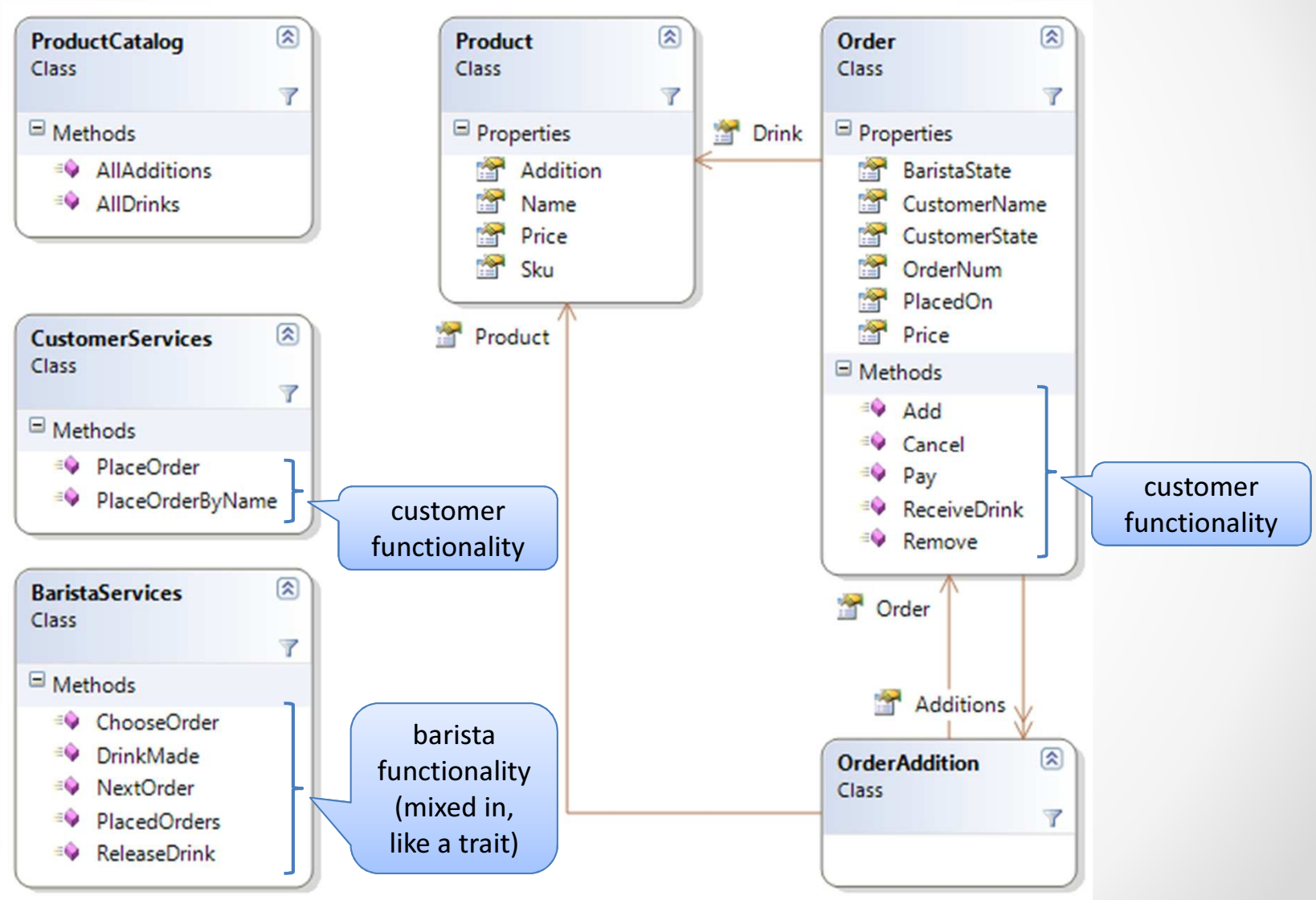
- Starting point is the home page
- http://localhost:8080
- RO defines a home page that links to:
  - services
  - user
  - version
  - domain-types (ie the metamodel)


RESTafarian
homePage
user
user
version
version
home Page
domain services
list
domain service
domain object
objects
object
domain types
type list
type act result
domain type action
domain type
list
details
object property
object prop
property description
details
object collection
object coll
collection description
details
object action
action description
action param description
if array
action result
if domain object
if scalar value
scalar
action invoke
invoke
object action
related representation
inlined representation
GET
POST
resource
PUT
DELETE
repres-entation

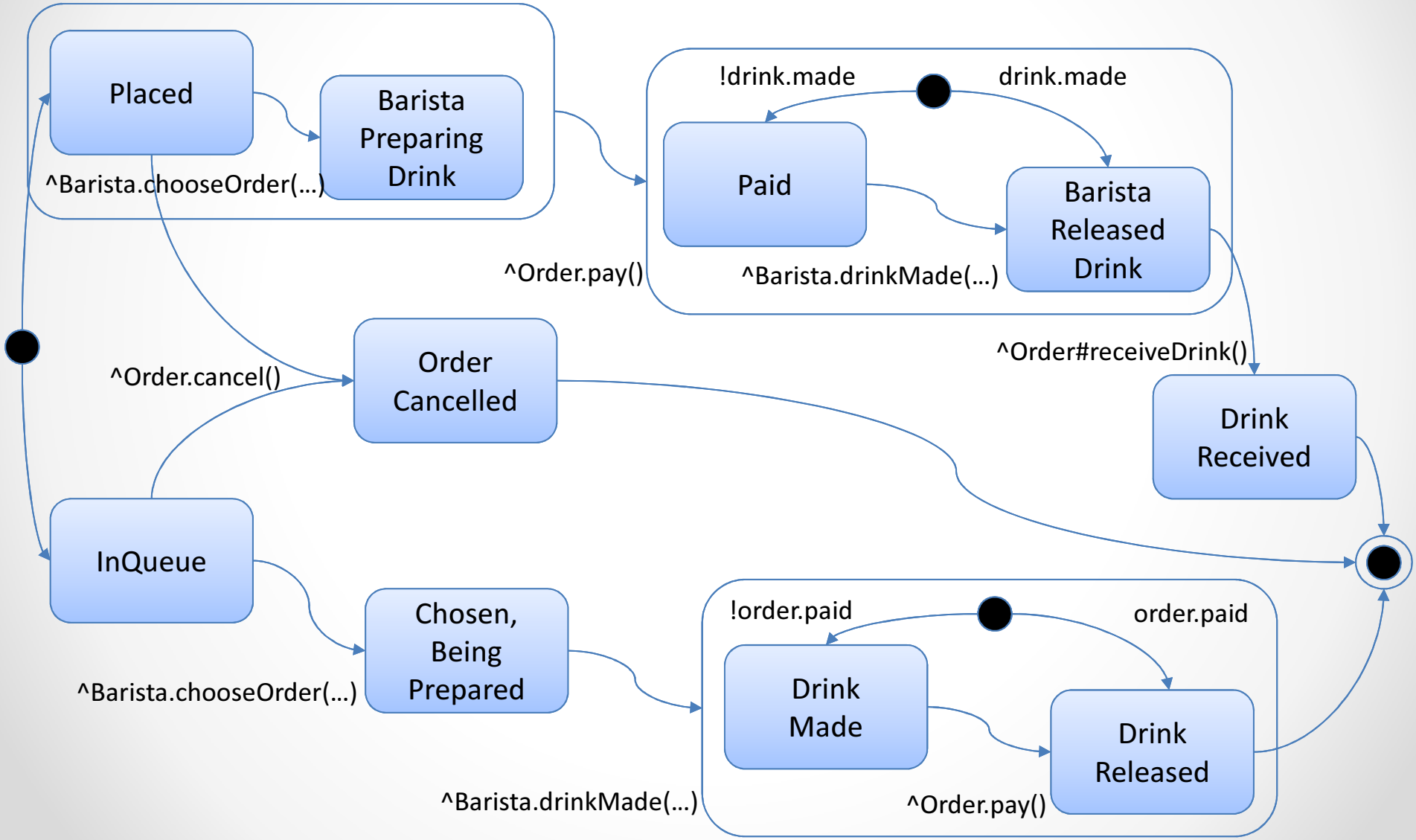
Demo

- An implementation of Jim Webber's "RestBucks" example
- Runs as an ASP.NET MVC app
  - exposes customer and barista services
- Runs as an RO app
  - exposes only the customer services
- Implementation:
  - MVC app provided by Naked Objects MVC
  - RO API is provided by Restful Objects.NET

Deeply unfashionable UML class diagram



Slightly more fashionable UML state diagram



Demo…

Media types

- Media types are important
  - allow the client to say what it can accept
  - tell the client what is being returned
  - the basis of conneg

- Of course, they are also broken

"application/json" - vs - "text/html"

RO Media types

- Being a good RESTful system, RO defines its media types
- All representations return
  - "application/json"

RO Media types

- But the spec also recognizes the different levels of abstraction
- Uses "profile" parameter to indicate nature of the representation

"application/json";profile="org.restfulobjects/repr-types/object"

-or-

"application/json";profile="org.restfulobjects/repr-types/action"

RO Media Types

- And goes one level further, to support generic vs bespoke clients

"application/json";profile="org.restfulobjects/repr-types/object";x-ro-domain-type="com.mycompany.myapp.v2.ShoppingBasket"

- NB:
  - **x-ro-** used as a prefix in parameters to avoid conflicts
  - RO spec does not define any custom HTTP headers

RO Media Types as Layers

x-ro-domain-type=
"com.mycompany.myapp.v2.ShoppingBasket"

profile=
"org.restfulobjects/repr-types/object"

application/json

Media type also part of the RO Link

```
{
 "rel": ".../...",
 "href": "http://localhost:8080/objects/order/123/properties/status",
 "type": "application/json;
         profile=\"org.restfulobjects/repr-types/property\"",
 "method": "GET"
}
```

HATEOAS

- Hypermedia APIs use links in representations
  - to walk from one resource to the next
- The link values are (can be) opaque
  - REST is not about "pretty URLs"
- Instead, it's the link's "rel" that defines the semantics
  - eg: "next", "prev", "describedby"

RO Link Rels

- The spec uses IANA-defined rel values where they exist
- Otherwise, the spec defines rel values that are similar to the RO media types

urn:org.restfulobjects:rels/details;property="firstName"

-or-

urn:org.restfulobjects:rels/choices;action="placeOrder";param="product"

RO Link Rels

```
{
 "rel": "urn:org.restfulobjects:rels/details;property=\"status\"",
 "href": "http://localhost:8080/objects/order/123/properties/status",
 "type": "application/json;
         profile=\"org.restfulobjects/repr-types/property\"",
 "method": "GET"
}
```

Other (Boring) Stuff

- Data types
- Resource argument representation
  - simple arguments
  - formal arguments
- Concurrency control
- Extensible representations
  - "links" list
  - "extensions" map

Optional capabilities

- Domain metadata (x-ro-domain-model)
- Validation (x-ro-validate-only)
- Blobs/clob data type and attachments
- Direct persistence

- The "version" resource lists support for optional capabilities

And direct persistence

URL / HTTP VERB	Objects/ {Dtype}	Objects/ {Dtype}/ {IID}	Objects/ {Dtype}/ {IID}/ Properties/ {Property}	Objects/ {Dtype}/ {IID}/ Collections/ {Collection}	Objects/ {Dtype}/ {IID}/ Actions/ {Action}	Objects/ {Dtype}/ {IID}/ Actions/ {Action}/ invoke
GET	n/a – 405	object summary, member summary, property values	property details and value	collection details and content	action prompt	invoke (if query only)
PUT	n/a – 405	update or clear multiple property values	update or clear value	add object (if set semantics)	n/a – 405	invoke (if idempotent)
DELETE	n/a – 405	delete object	clear value	remove object	n/a – 405	n/a – 405
POST	persist instance	n/a – 405	n/a – 405	add object (if list semantics)	n/a – 405	invoke (any)

Should entities be exposed as resources?

- Oberg says no
  - "The domain model as REST anti-pattern" blog post
- Webber says no
  - cf #DDDX 2011, and a bunch of disparaging remarks about Rails
- I say: it's a little more nuanced than that
  - who owns the pipe?
  - should the client depend on out-of-band info?

Who owns the pipe?

- An enterprise app, deployed on intranet
  - both client and server built by same team

- vs -

- A REST API, deployed on internet
  - multiple clients, developed by 3rd parties
  - API can't be broken willy-nilly

Out-of-band info?

- RESTafarians like to talk about the evil of out-of-band information
- Generic client
  - understands semantics of media type
  - makes no hard-coded assumptions about representation content
  - RO specifies comprehensive metadata to support such clients
    - eg "isSuperTypeOf" type action
- Bespoke client
  - does assume presence of specific content
  - optimized to support specific use cases

Client/server independence

- REST says: "Client and server should evolve independently"
  - not always required, though

Deployment Client type	Intranet	Internet
Generic	no need for independence	no need for independence
Bespoke	no need for independence	**must** be independent

Resources as use cases/commands

- The current "conventional wisdom"
- Does enable client/server independence
  - relevant for bespoke/internet deployments
    - and is a good design, one that is supported by RO spec
  - however, it isn't required for other deployment scenarios
- So why strongly advocated for all scenarios?
  - probably because creating a general-purpose graph of hyperlinks is difficult without a metamodel
  - use case resources artificially constrain the links to those defined by a small state machine

RO Domain Object ontology

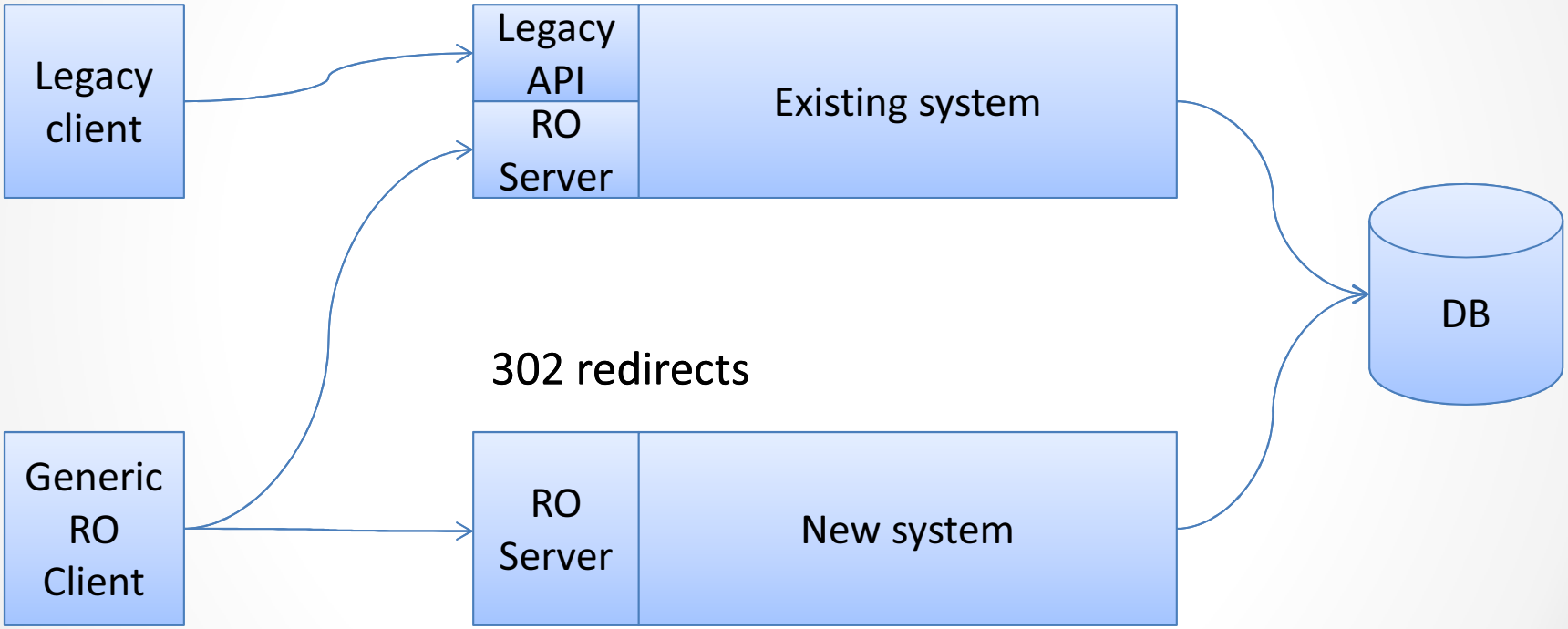
- Persistent domain entity
- Proto-persistent domain entity

domain layer

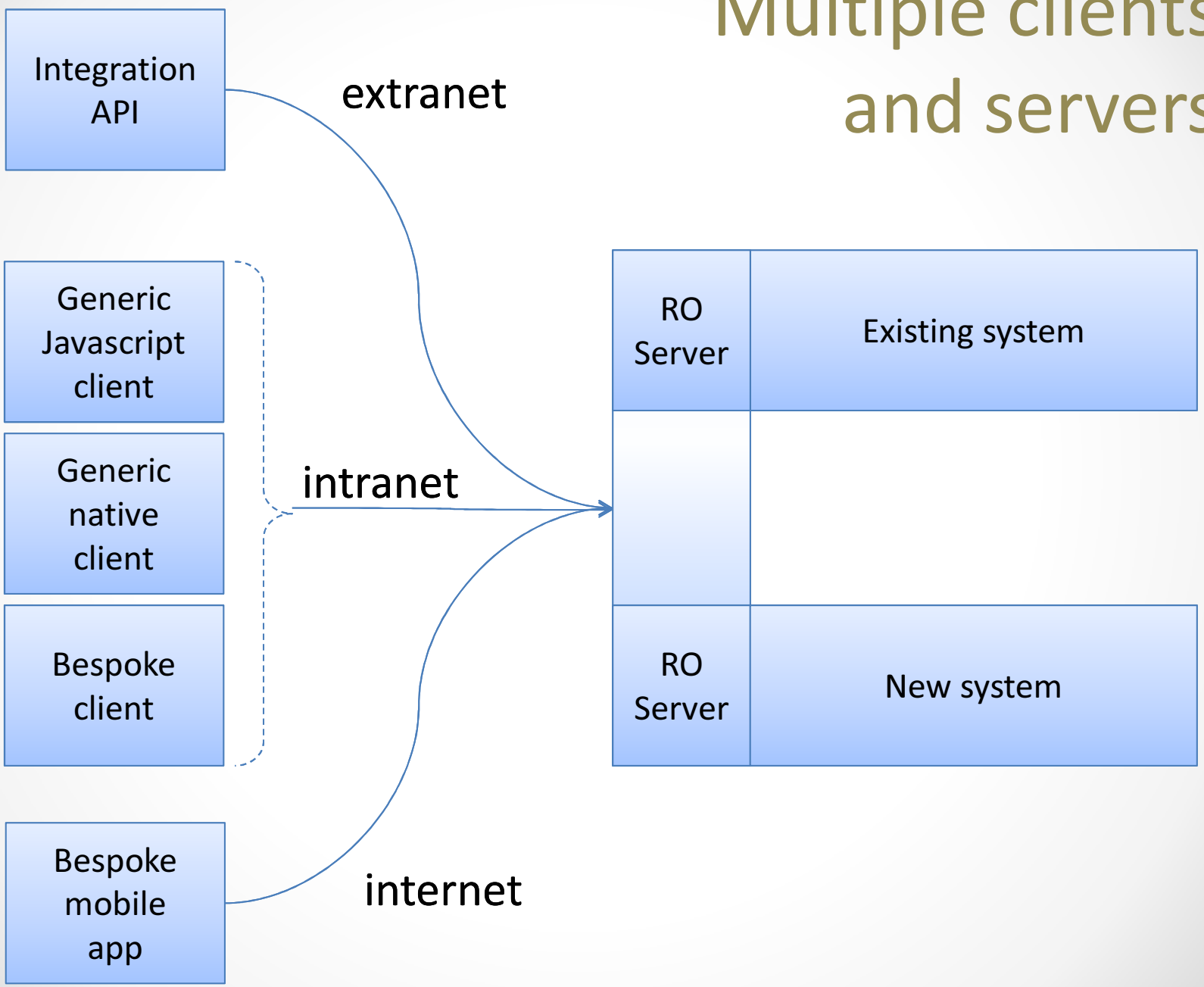
- View model
- Addressable view model

application layer

Migration


Legacy client
Legacy API
RO Server
Existing system
DB
302 redirects
Generic RO Client
RO Server
New system

Multiple clients and servers



Concluding…

- The Restful Objects spec
  - defines a hypermedia API
  - for behaviourally complete domain objects
  - JSON over HTTP
  - supports both generic and bespoke clients
- There are two open source implementations
- There are some nascent Javascript clients
- Could use spec independently of a framework
  - that said, HATEOAS is difficult to support without a metamodel
  - unless artificially restrict links to narrow state machine

(Not so) hidden agenda

- Most people don't get Naked Objects
  - maybe the generic UI puts them (you?) off
- But the benefit of NO is that it actively promotes building the ubiquitous language
  - NO ain't about UI, it's about building a richer domain model
- RO retains the essence of NO
  - but lets you skin the model as you see fit
    - using cool technologies, if that's your bag
  - also opens up integration/migration scenarios

References

Restful Objects spec	http://restfulobjects.org http://github.com/danhaywood/restfulobjects-spec
Apache Isis (JVM)	http://incubator.apache.org/isis
Restful Objects.NET	http://restfulobjects.codeplex.com
Naked Objects MVC	http://nakedobjects.codeplex.com
My Blog	http://danhaywood.com
Twitter	@dkhaywood
Coffee Shop demo	http://github.com/danhaywood/dotnet-coffeeshop