Adaptable Clients and Evolvable APIs

Mike Amundsen
API Academy / CA
@mamund

Introduction


Ctrl
PgUp
Mike Amundsen
@mamund

API Design Training | API

www.apiacademy.co

API ACADEMY

Search API Academy

API Strategy | API Design | API Management | Resources | About

Register | Sign In

Your Guide to API Design & Implementation Best Practices

API Academy delivers free online lessons and in-person consulting services covering essential API techniques and tools for business managers, interface designers and enterprise architects

What is an API?

Get an overview of what an API is and what it does, to help you realize the business value of APIs

API Design Basics

Understand the API architecture process and learn basic design and implementation best practices

Web API Architectural Styles

Get a detailed overview of the main architectural styles for Web and mobile API design

Choosing a Solution

Choose between the various solutions that offer the basic components for enterprise API Management

Creating Evolvable Hypermedia Applications

Services for a Changing World

Building

Hypermedia APIs with HTML5 & No

O'REILLY®

Mike

RESTful Web APIs

LLY®

Leonard Richardson, Mike Amundsen & Sam Ruby

O'REILLY®

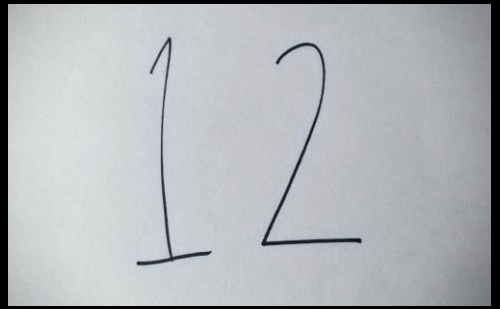
Learning Client Hypermedia

ENABLING CLIENT APPLICATIONS WITH THE POWER OF THE WEB

Mike Amundsen

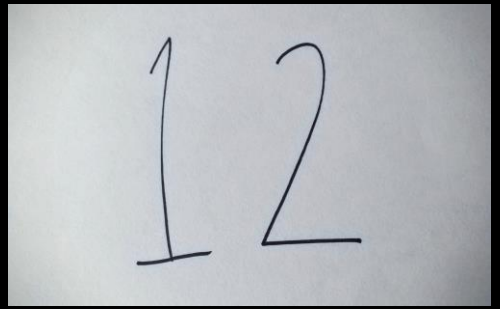
Twelve Patterns for Adaptable Apps

Four Design Patterns
Four Basic Principles
Four Shared Agreements



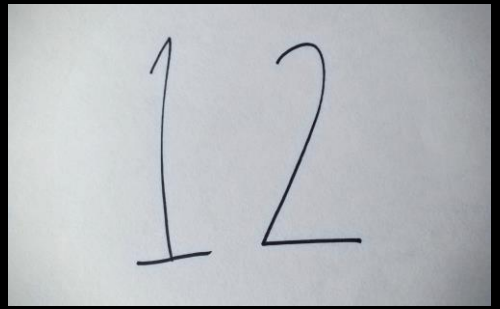
Design Patterns

1. PASS MESSAGES, NOT OBJECTS
2. SHARE VOCABULARIES, NOT MODELS
3. THE REPRESENTOR PATTERN
4. PUBLISH PROFILES



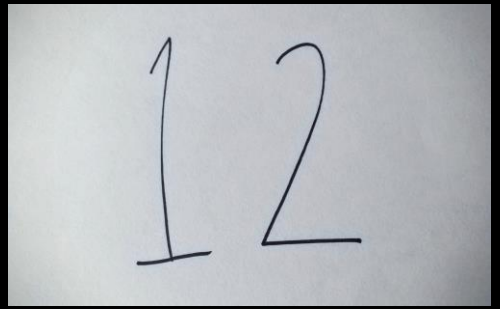
Basic Principles

5. MUST IGNORE
6. MUST FORWARD
7. PROVIDE MRU
8. USE IDEMPOTENCE



Basic Agreements

9. USE RELATED
10. USE NAVIGATION
11. PARTIAL SUBMIT
12. STATE WATCH



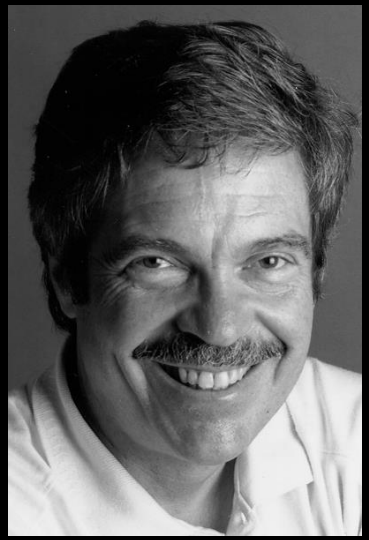
Caution!

These are preliminary drawings and may change prior to publication

Design Patterns

Pass Messages, Not Objects

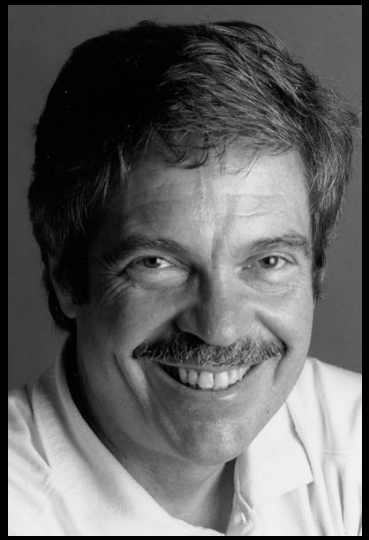
"I'm sorry that coined the term 'objects' for this topic. The big idea is 'messaging'."



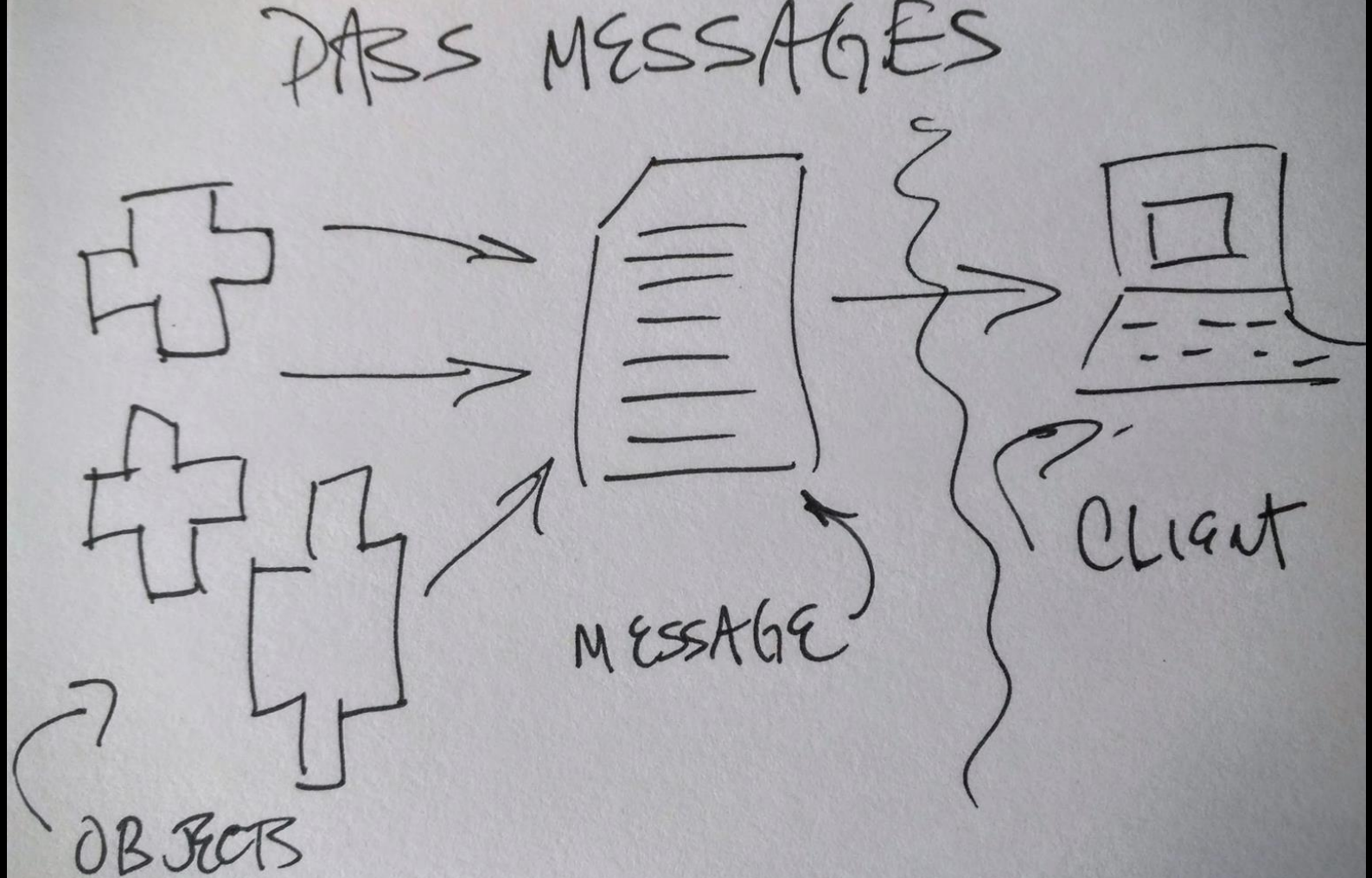
Alan Kay, 1998

Pass Messages, Not Objects

*"I'm sorry that coined the term 'objects' for this topic. **The big idea is 'messaging'.**"*



Alan Kay, 1998


PASS MESSAGES
CLIENT
MESSAGE
OBJECTS

Pass Messages, Not Objects

Use a Registered Hypermedia Type

HAL
Collection+JSON
Siren
UBER
Atom

Share Vocabularies, Not Models

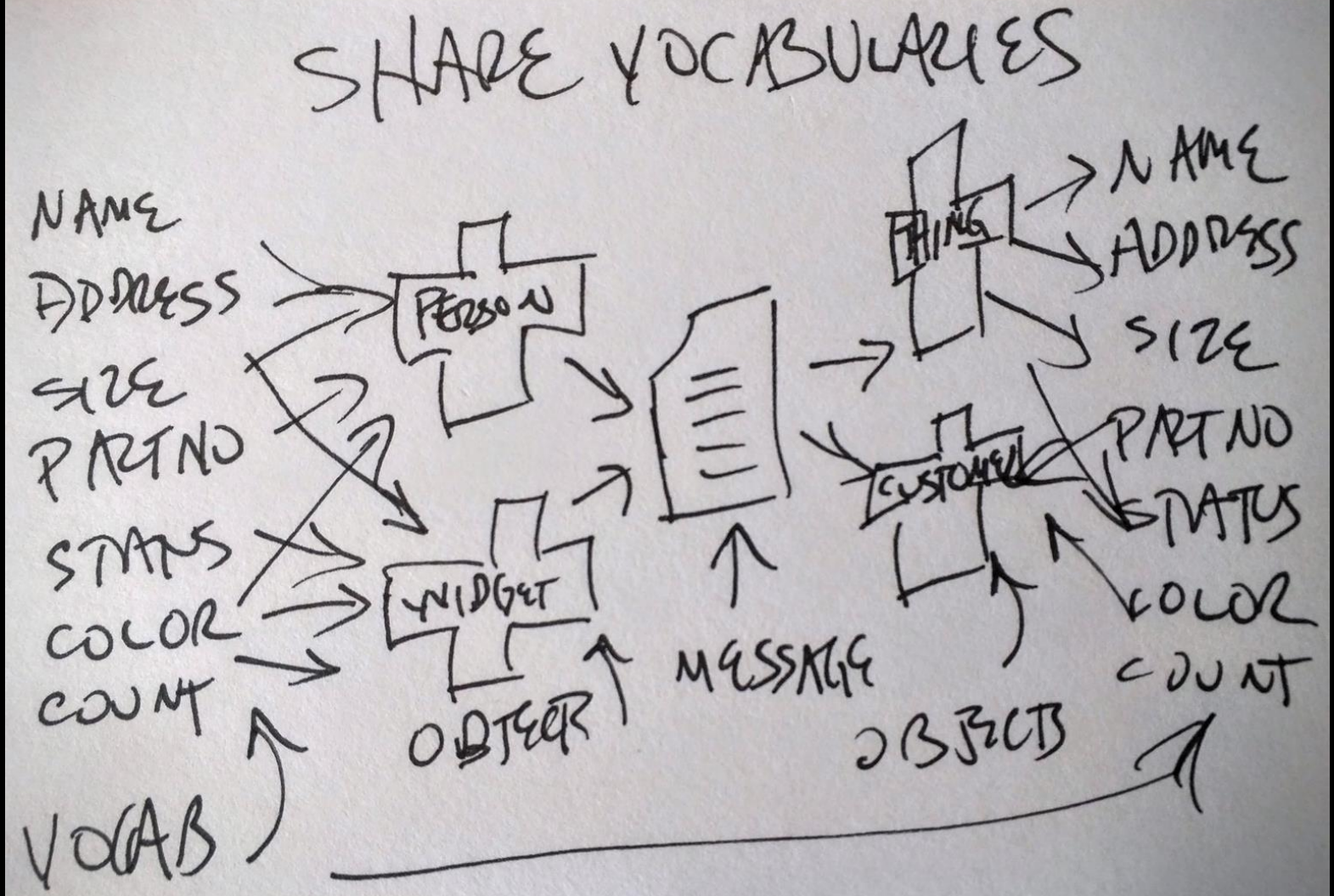
"It is easier to standardize representation and relation types than objects and object-specific interfaces."

-- Roy Fielding

Share Vocabularies, Not Models

"It is easier to standardize representation and relation types than objects and object-specific interfaces."

-- Roy Fielding


SHARE VOCABULARIES
NAME
ADDRESS
SIZE
PARTNO
STATUS
COLOR
COUNT
VOCAB
PERSON
WIDGET
OBJECT
MESSAGE
THING
CUSTOMER
NAME
ADDRESS
SIZE
PARTNO
STATUS
COLOR
COUNT
OBJECTS

Share Vocabularies, Not Models

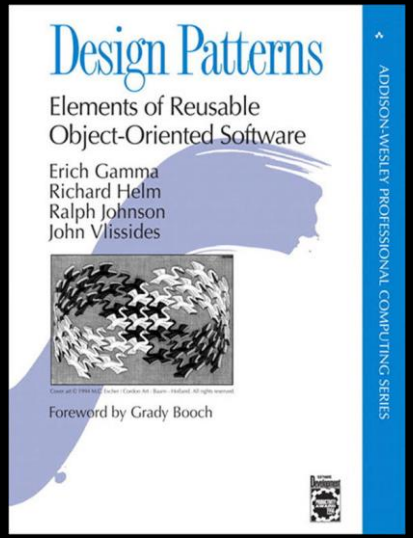
Use Existing Shared Vocabularies

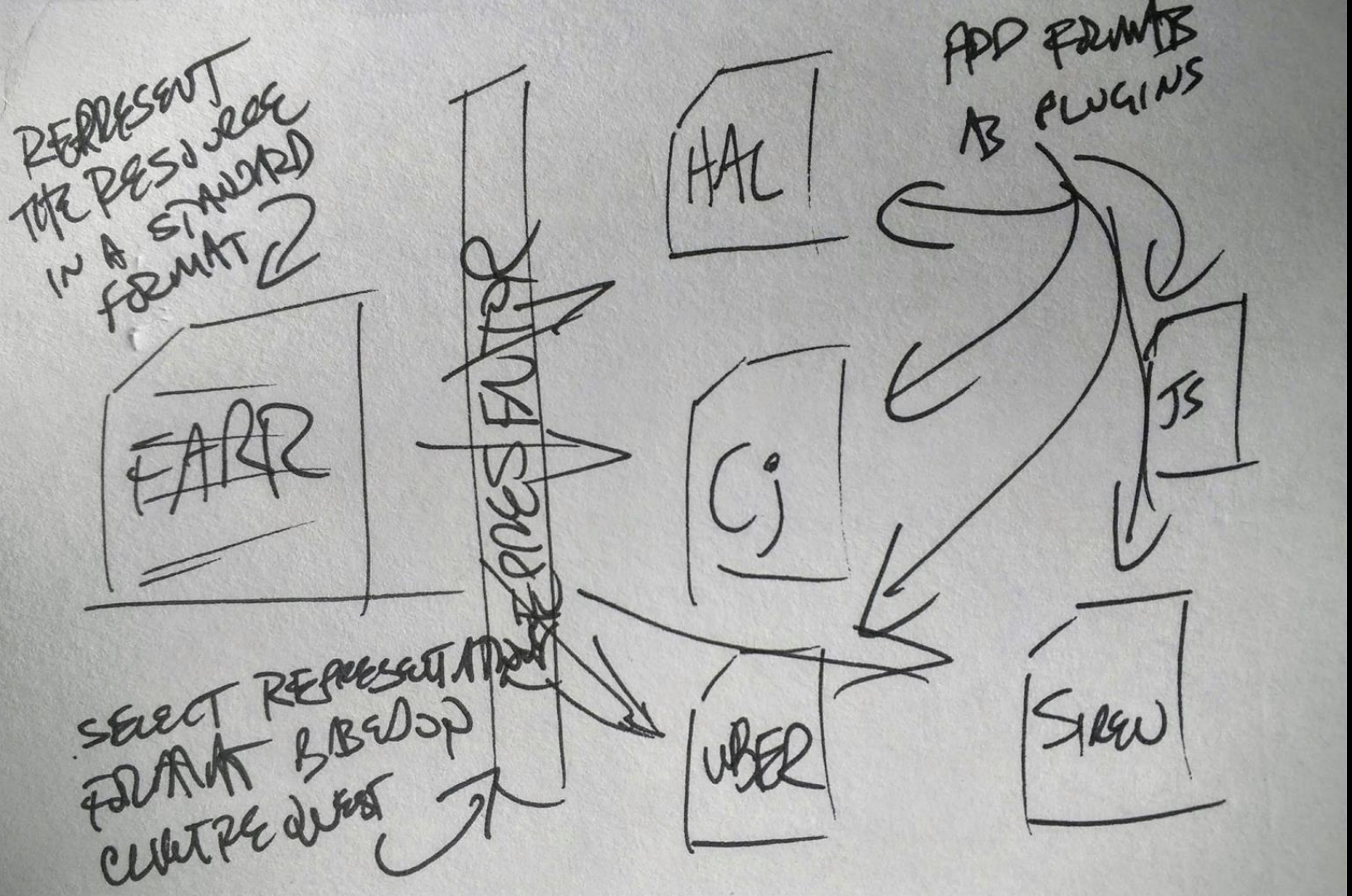
IANA Link Relation Values
Schema.org
Microformats
Dublin Core
Activity Streams

Use the Representor Pattern

"The Strategy Pattern lets the algorithm vary independently of the clients that use it."

- Gamma, et al.




REPRESENT THE RESOURCE IN A STANDARD FORMAT
EAR
NEGOTIATE
HAL
Cj
ADD FORMATS AS PLUGINS
JS
SELECT REPRESENTATION FORMAT BASED ON CLIENT REQUEST
UBER
SIREN

Use the Representor Pattern

Implement a Representor/Strategy Pattern

Standard Internal Resource Model
Strategy Messages Format Dispatch

Use the Re…rn

Implement a … Pattern

Standard Inte…
Strategy Mes…ch

```
// dispatch to requested representor
switch(mimeType.toLowerCase()) {
  case "application/json":
    doc = json(object, root);
    break;
  case "application/vnd.collection+json":
    doc = cj(object, root);
    break;
  case "application/hal+json":
    doc = haljson(object, root);
    break;
  case "application/vnd.uber+xml":
    doc = uberxml(object, root);
    break;
  case "text/html":
  case "application/html":
  default:
    doc = html(object, root);
    break;
}

return doc;
```

Publish Profiles

"Profiles provide a way to create a ubiquitous language for talking about APIs (resources) for both humans and machines."
-- Mark Foster

Publish Profiles

Use a Profile like ALPS to share vocabularies

Define all possible data and actions
Publish using Profile Standard (RFC6906)
Servers emit profile URI
Clients validate profile URI

Publish

Use a Pr ularies

Define a
Publish u 06)
Servers
Clients v

products-alps.xml

```xml
<alps version="1.0">
  <link rel="help" href="http://example.org/documentation/products.html" />
  <doc>
    This is a prototype product API.
  </doc>

  <!-- transitions -->
  <descriptor id="item" type="safe" rt="#product">
    <doc>Retrieve A Single Product</doc>
  </descriptor>

  <descriptor id="collection" type="safe" rt="#product">
    <doc>Provides access to all products</doc>
  </descriptor>

  <descriptor id="search" type="safe" rt="#product">
    <doc>Provides access to all products</doc>
    <descriptor href="#id" />
  </descriptor>

  <descriptor id="edit" type="idempotent" rt="#product">
    <doc>Updates A Product</doc>
    <descriptor href="#product" />
  </descriptor>

  <descriptor id="create" type="unsafe" rt="#product">
    <doc>Allows the creation of a new product</doc>
    <descriptor href="#product" />
  </descriptor>
```

Basic Principles

Must Ignore

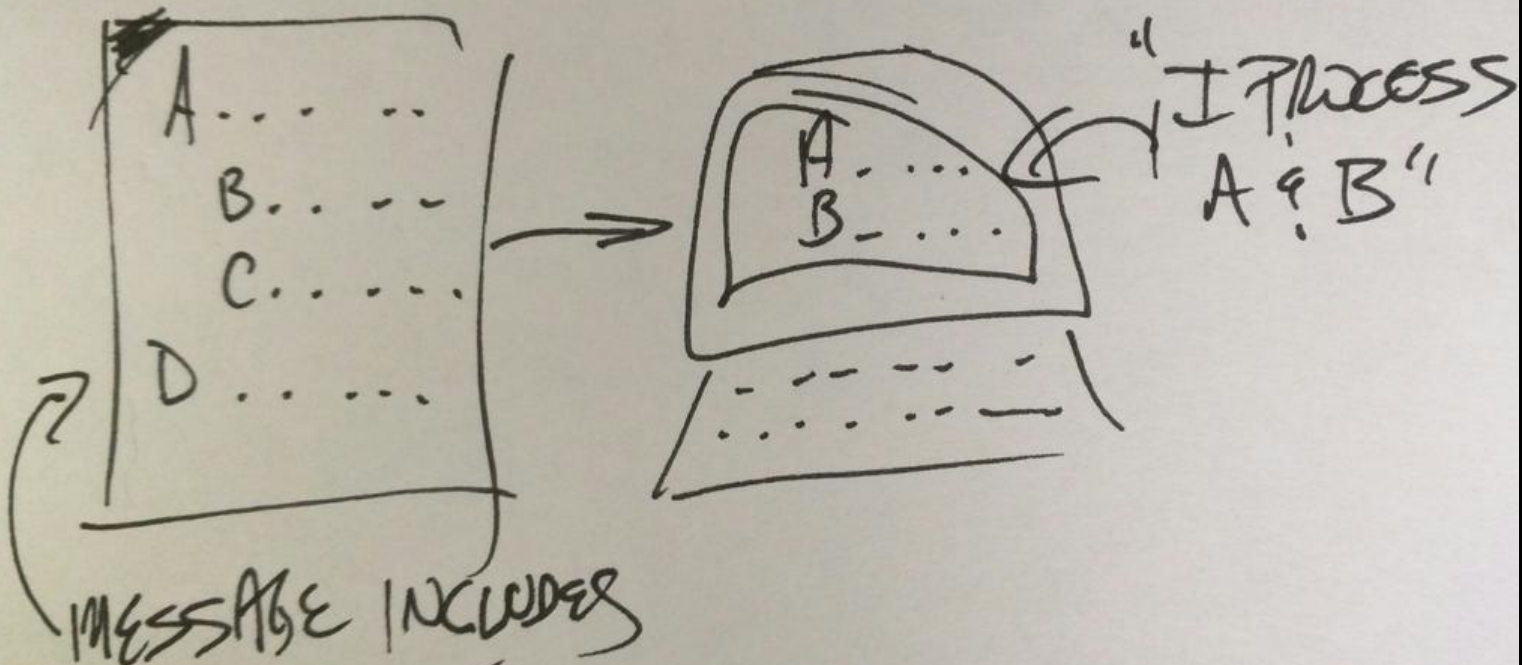
“The main goal of the MUST IGNORE pattern of extensibility is to allow backwards- and forwards-compatible changes.”

- David Orchard

Must Ignore

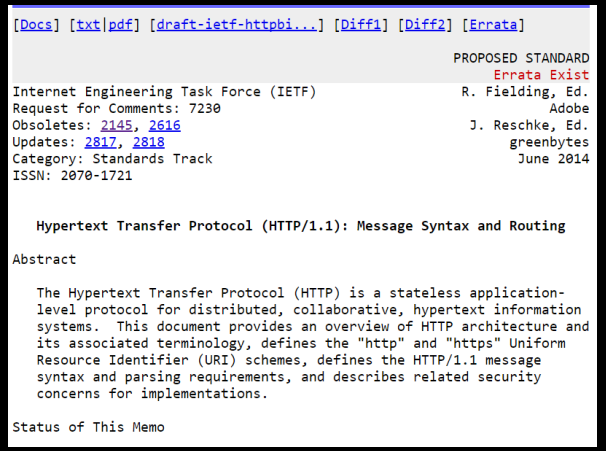
Clients MUST IGNORE any data/inputs that the client does not understand.

MUST-IGNORE


A
B
C
D
A
B
"I PROCESS A & B"
MESSAGE INCLUDES DATA I DO NOT CARE ABOUT

MUST FORWARD

“A proxy MUST forward unrecognized header fields…”
-- RFC 7230



[Docs] [txt|pdf] [draft-ietf-httpbi...] [Diff1] [Diff2] [Errata]

PROPOSED STANDARD
Errata Exist

Internet Engineering Task Force (IETF) — R. Fielding, Ed., Adobe
Request for Comments: 7230
Obsoletes: 2145, 2616 — J. Reschke, Ed., greenbytes
Updates: 2817, 2818
Category: Standards Track — June 2014
ISSN: 2070-1721

Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing

Abstract

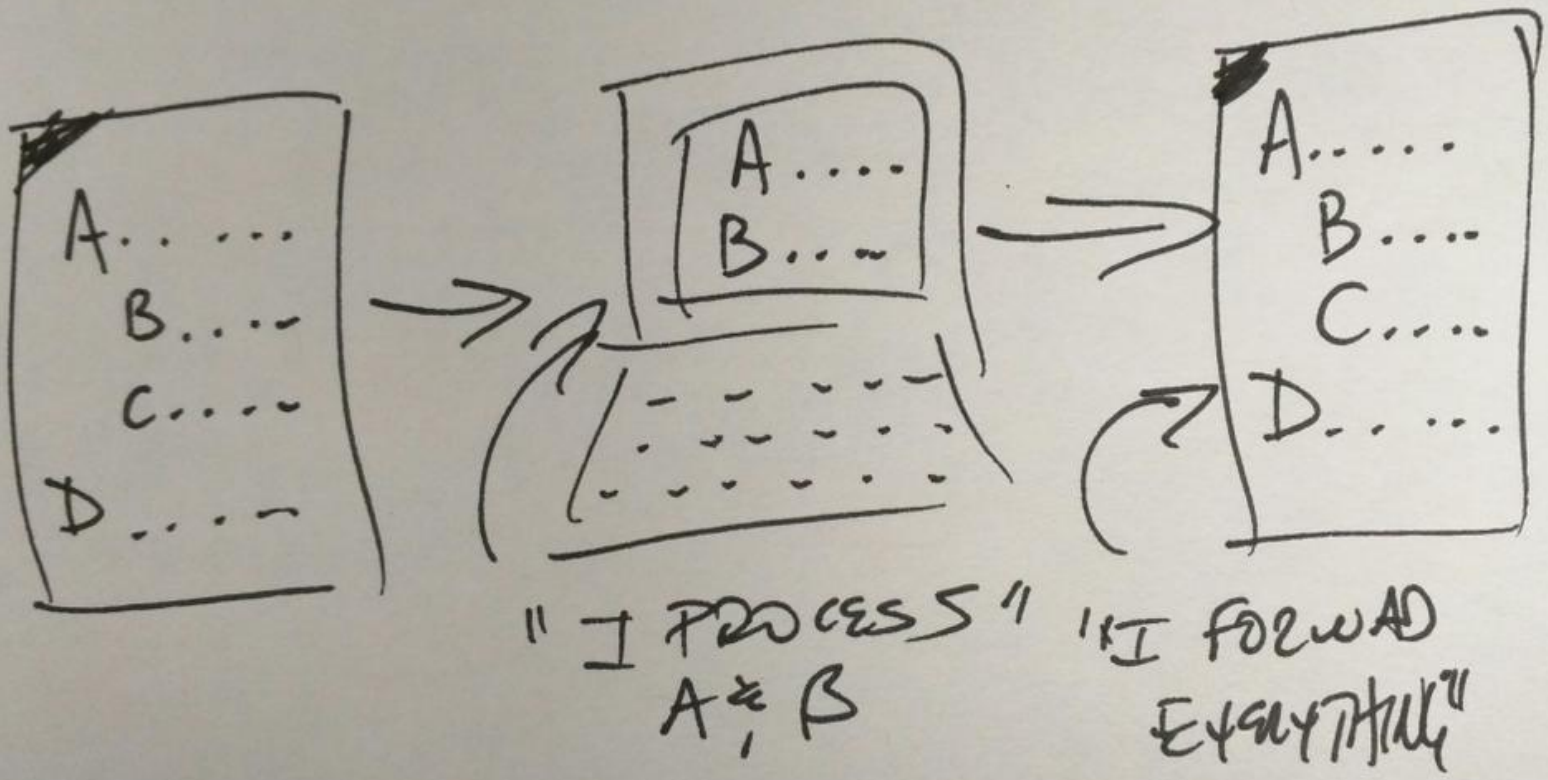
The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems. This document provides an overview of HTTP architecture and its associated terminology, defines the "http" and "https" Uniform Resource Identifier (URI) schemes, defines the HTTP/1.1 message syntax and parsing requirements, and describes related security concerns for implementations.

Status of This Memo

Must Forward

Clients MUST FORWARD (unchanged) any input fields (URL or FORM) that the client does not recognize.

MUST-FORWARD



Provide MRU

“A feature of convenience allowing users to quickly see and access the last few used files and documents.”
-- Wikipedia

Common menus in Microsoft Windows

From Wikipedia, the free encyclopedia

This is a **list of commonly used Microsoft Windows menus**.

Contents [hide]

1 Microsoft menus
1.1 Most Recently Used menu
1.2 Properties menu
1.3 System menu
2 References

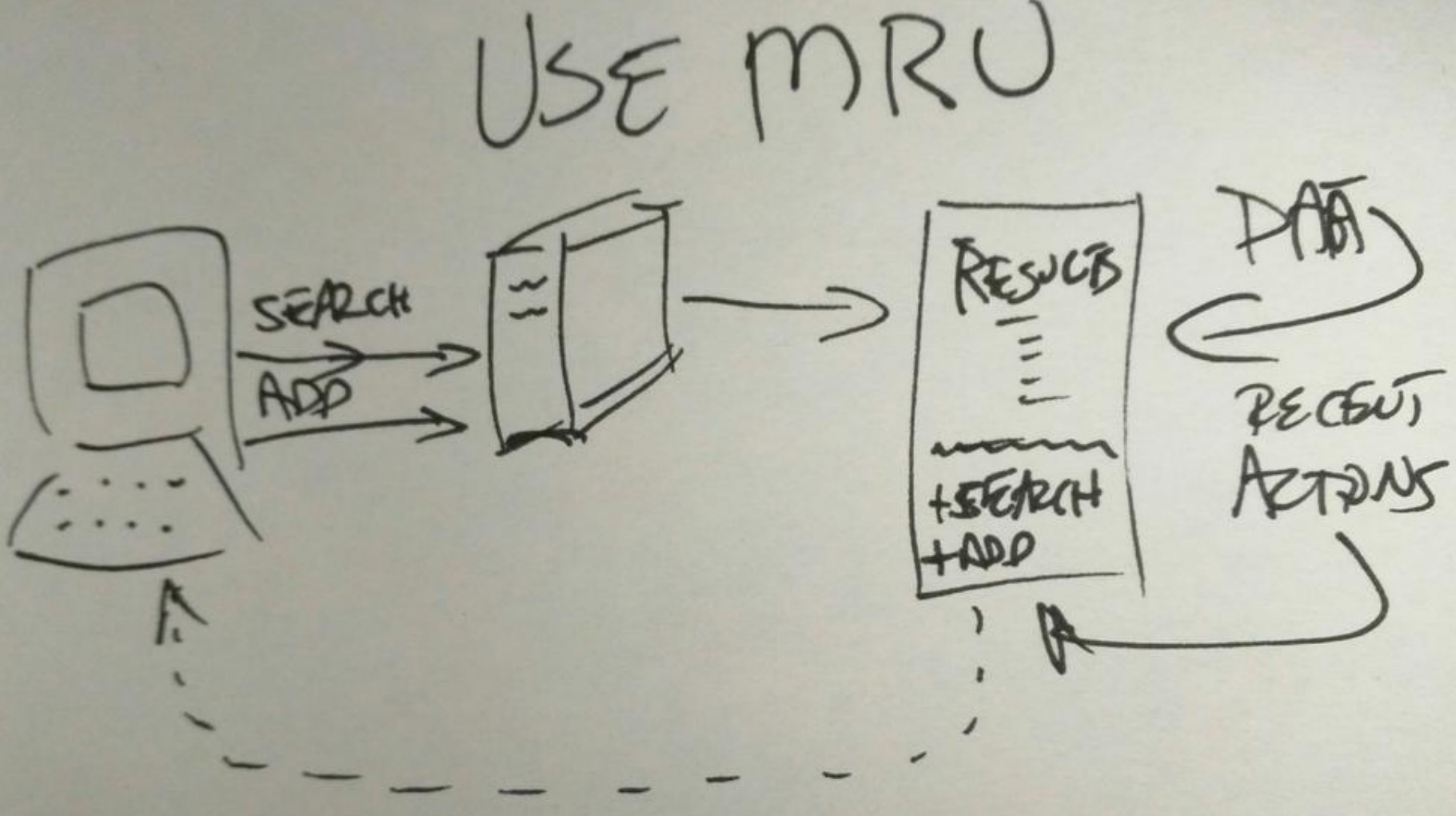
Microsoft menus [edit]

Most Recently Used menu [edit]

Most Recently Used (MRU) is a term used in computing to refer to the list of progr
quickly see and access the last few used files and documents, but could also be c

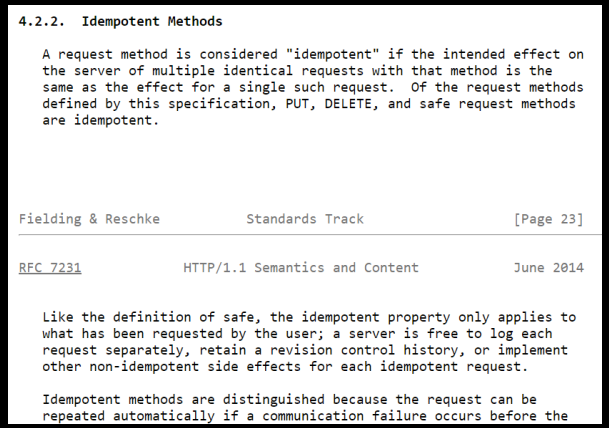
Provide MRU

Services SHOULD return the most recently-used (MRU) LINKS and FORMS in all responses.


USE MRU
SEARCH
ADD
RESWCB
+SEARCH
+ADD
DATA
RESULT
ACTIONS

Use Idempotence

“Can be applied multiple times without changing the result beyond the initial application.”
-- Wikpedia



4.2.2. Idempotent Methods

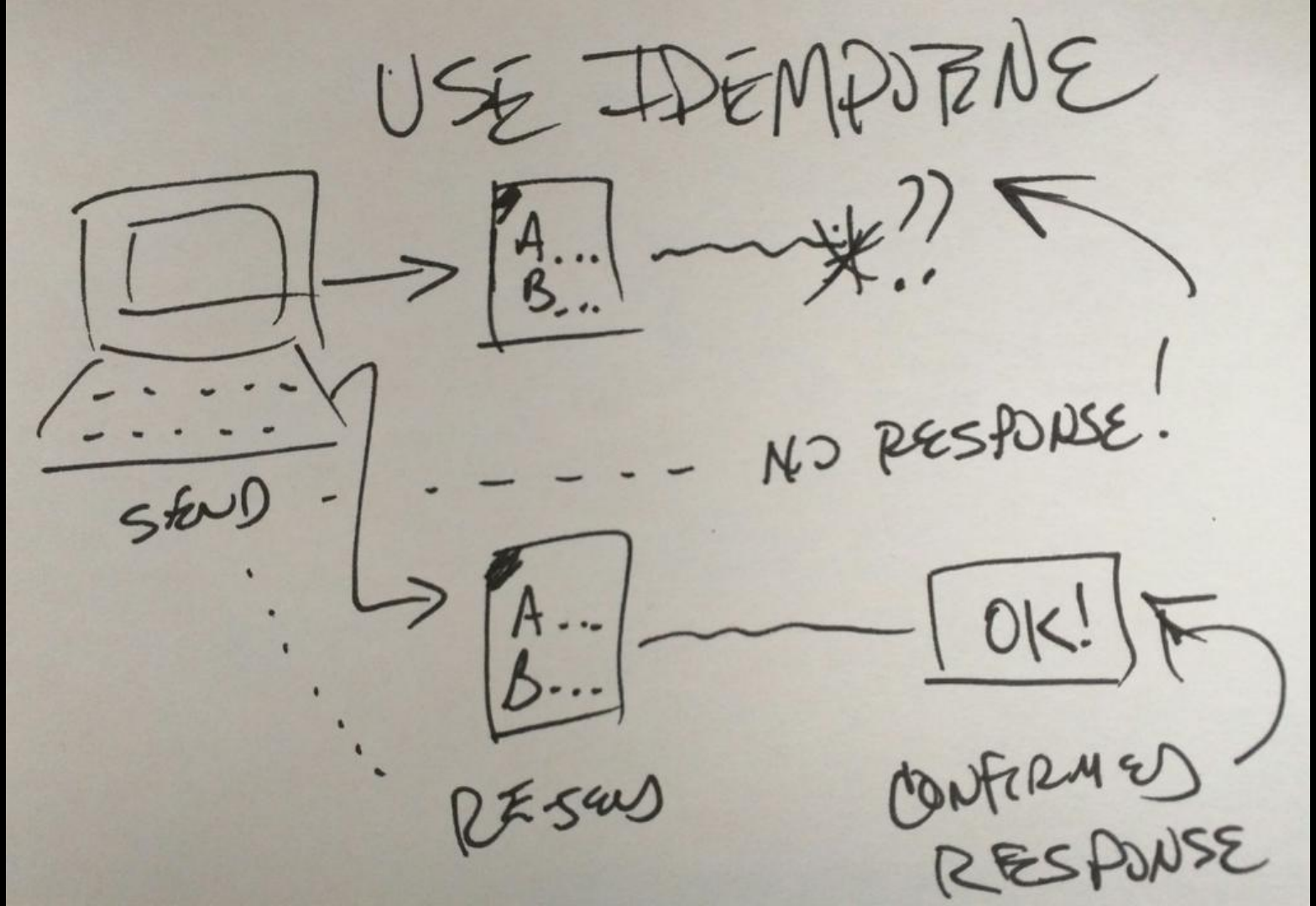
A request method is considered "idempotent" if the intended effect on the server of multiple identical requests with that method is the same as the effect for a single such request. Of the request methods defined by this specification, PUT, DELETE, and safe request methods are idempotent.

Like the definition of safe, the idempotent property only applies to what has been requested by the user; a server is free to log each request separately, retain a revision control history, or implement other non-idempotent side effects for each idempotent request.

Idempotent methods are distinguished because the request can be repeated automatically if a communication failure occurs before the

Use Idempotence

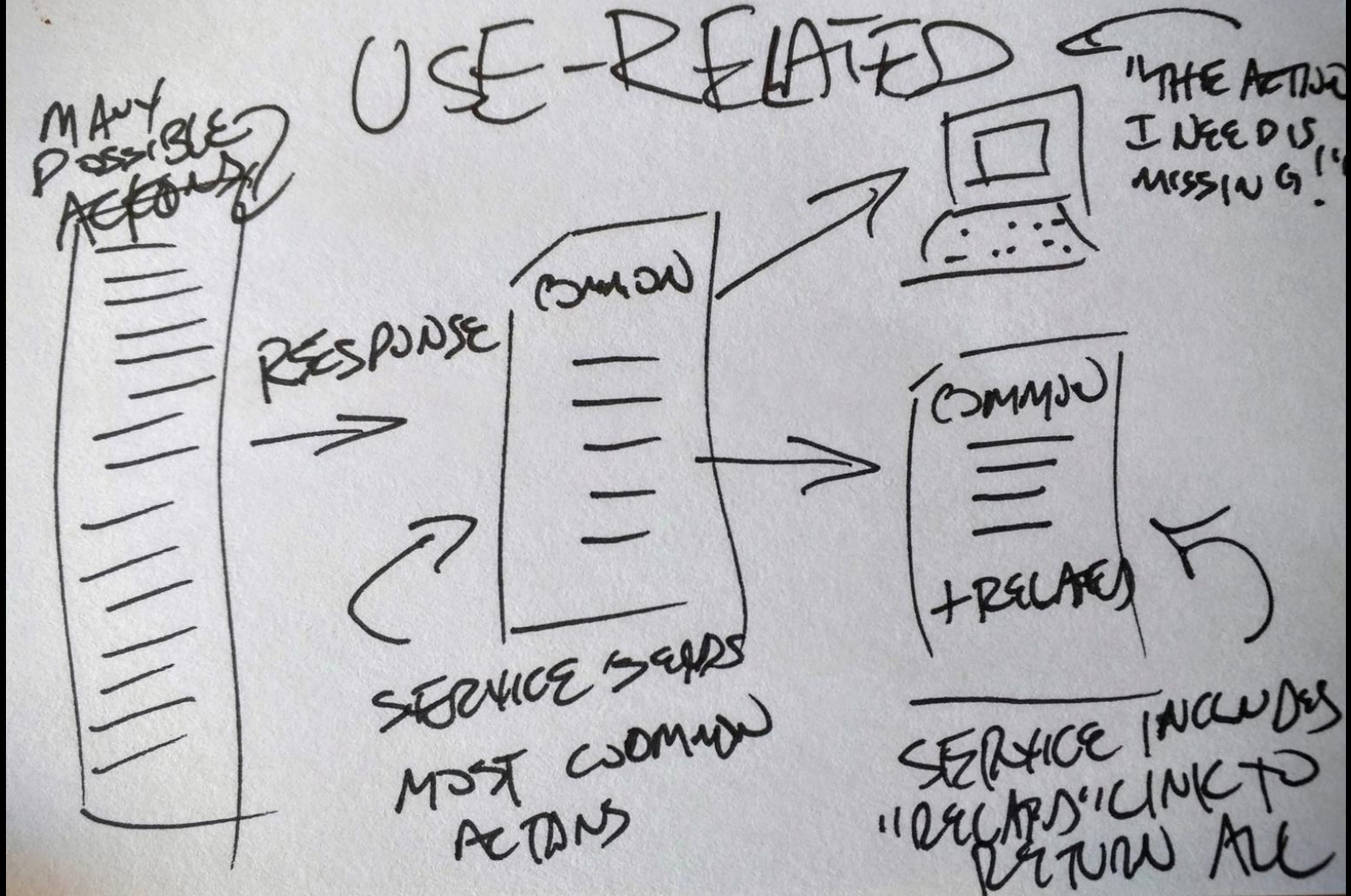
All network requests SHOULD be idempotent in order to allow clients to safely repeat them when response is unclear.


USE IDEMPOTENT
A...
B...
??
NO RESPONSE!
SEND
A...
B...
OK!
RESEND
CONFIRMED
RESPONSE

Shared Agreements

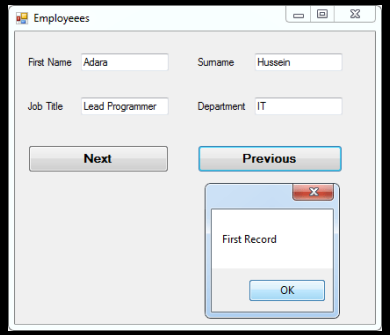
Use Related

Services SHOULD return a RELATED LINK that responds with ALL the possible actions for this context.


USE-RELATED
"THE ACTION I NEED IS MISSING!"
MANY POSSIBLE ACTIONS
RESPONSE
COMMON
SERVICE SENDS MOST COMMON ACTIONS
COMMON
+RELATED
SERVICE INCLUDES "RELATED" LINK TO RETURN ALL

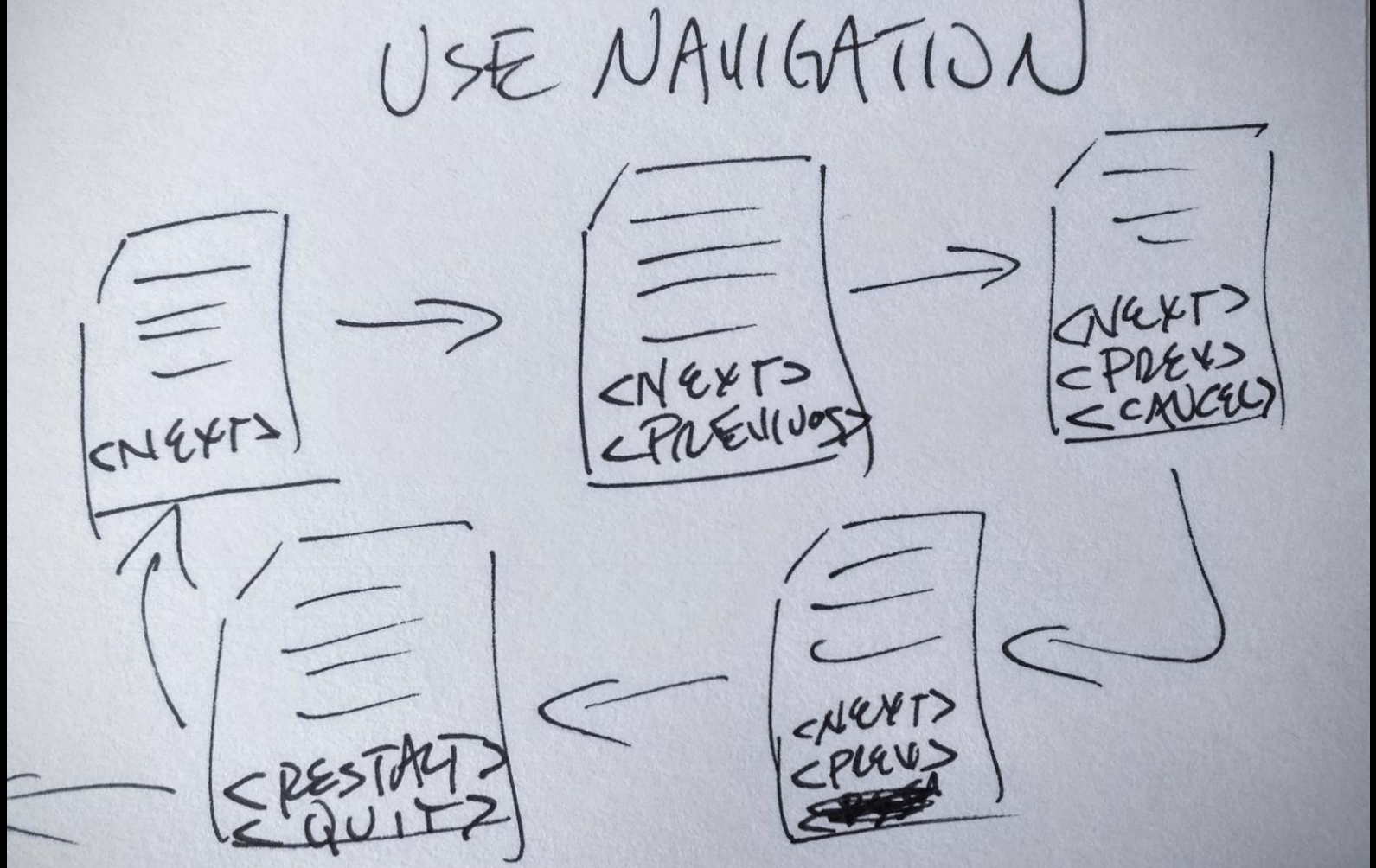
Use Navigation

“To achieve a single goal which can be broken down into dependable sub-tasks.”
-- Design Patterns (@uipatterns)



Use Navigation

Services SHOULD provide "next/previous" LINK to handle multi-step workflow with "cancel", "restart", & "done."


USE NAVIGATION
<NEXT>
<NEXT>
<PREVIOUS>
<NEXT>
<PREV>
<CANCEL>
<NEXT>
<PREV>
<RESTART>
<QUIT>

Partial Submit

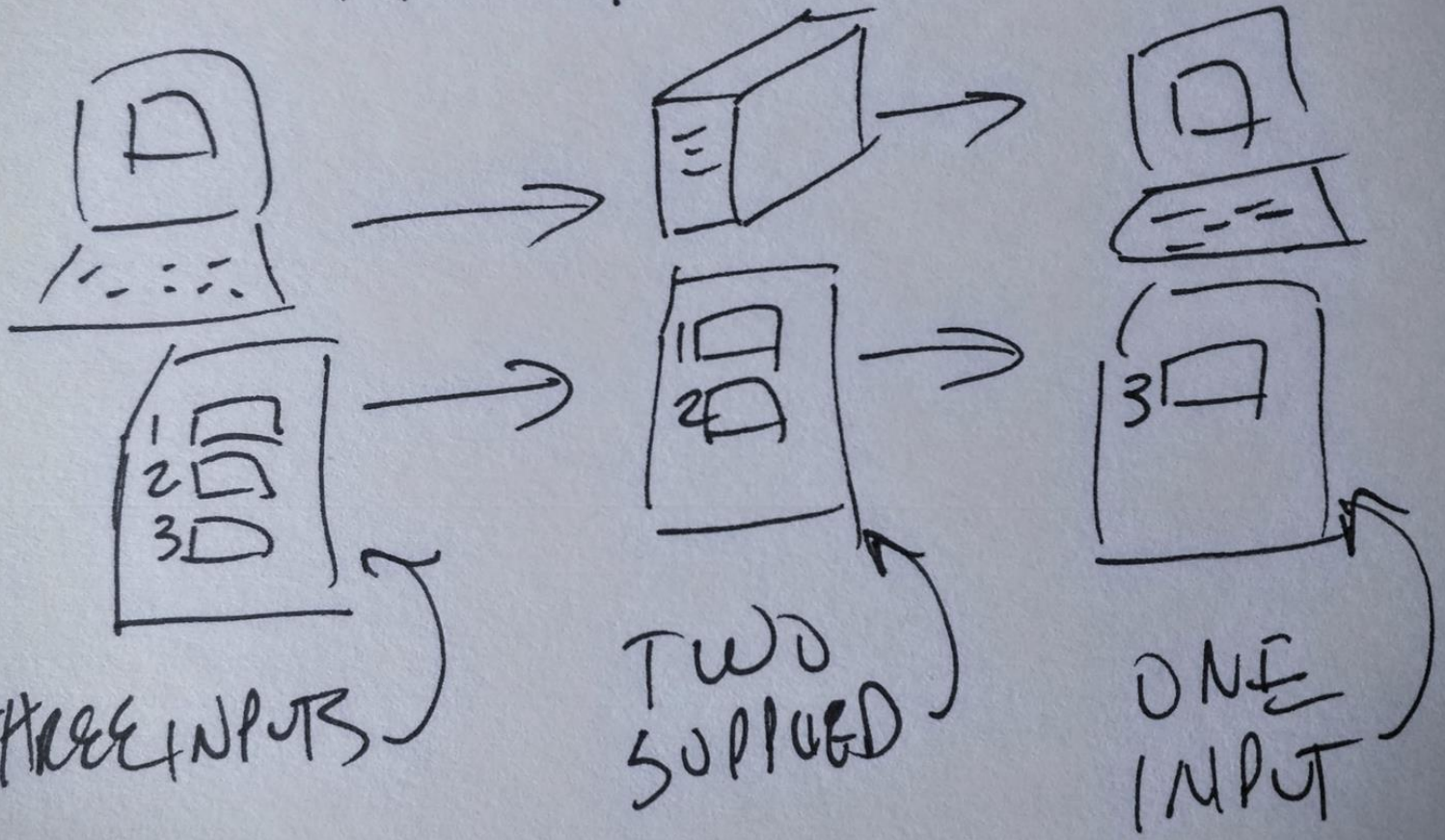
“Think of the actions as approximations of what is desired.”

-- Donald Norman

Partial Submit

Services SHOULD accept partially filled-in FORM and return a new FORM with the remaining fields.

PARTIAL SUBMIT



State Watch

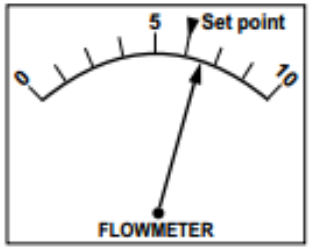
“Data representing variables in a dynamical system…”

-- Jens Rassmussen

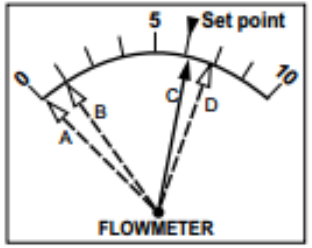
State Wato

“Data rep *a dynamical system…”*

assmussen

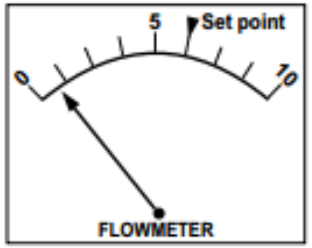


SIGNAL
- Keep at set point
- Use deviation as error signal
- Track continuously



SIGN
Stereotype acts

If Valve Open	If C, ok If D, adjust flow
If Valve Closed	If A, ok If B, recalibrate meter

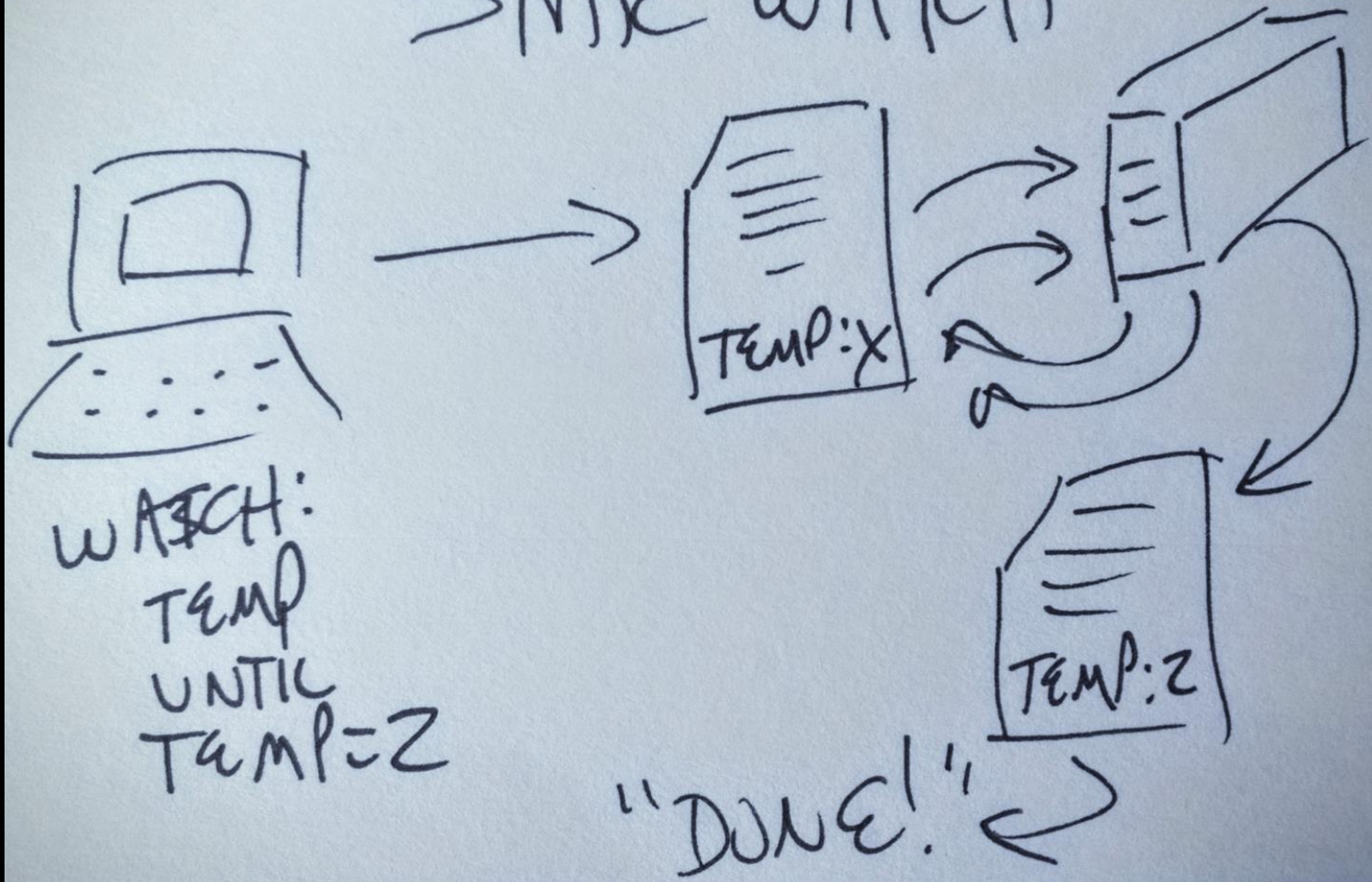


SYMBOL
If, after calibration, is still B, begin to read meter and speculate functionally (could be a leak)

State Watch

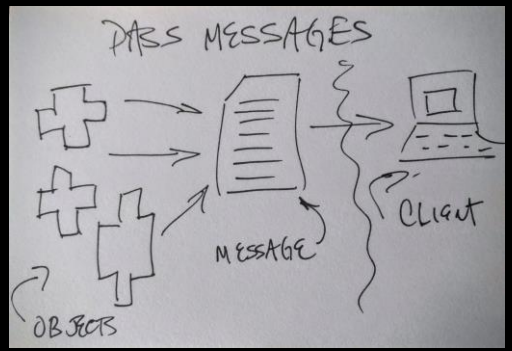
Services SHOULD allow clients to subscribe to WATCH VALUES so that clients can deterimine "done."

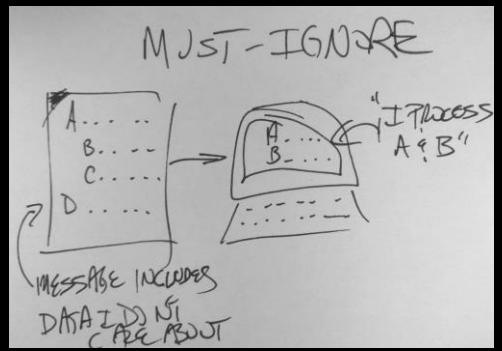
STATE WATCH

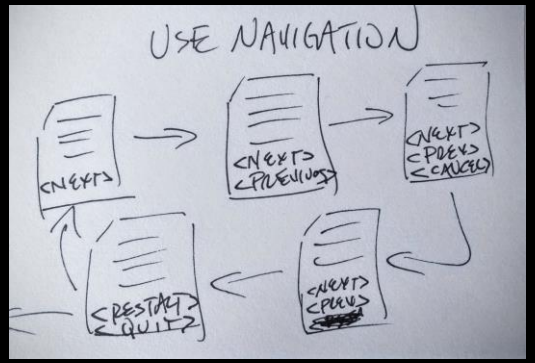


Twelve Patterns for Adaptable Apps

Four Design Patterns
Four Basic Principles
Four Shared Agreements

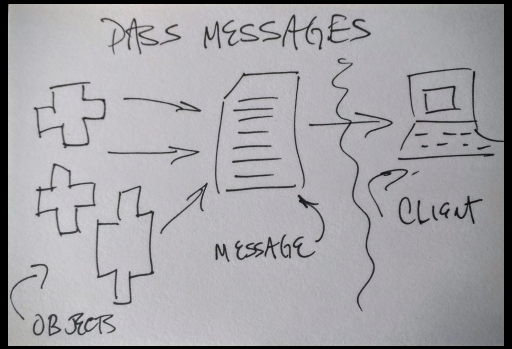






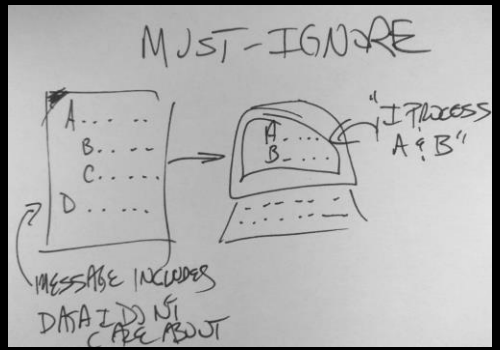
Design Patterns

1. PASS MESSAGES, NOT OBJECTS
2. SHARE VOCABULARIES, NOT MODELS
3. THE REPRESENTOR PATTERN
4. PUBLISH PROFILES



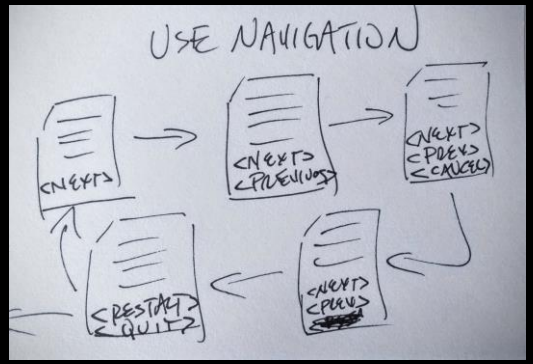
Basic Principles

5. MUST IGNORE
6. MUST FORWARD
7. PROVIDE MRU
8. USE IDEMPOTENCE



Basic Agreements

9. USE RELATED
10. USE NAVIGATION
11. PARTIAL SUBMIT
12. STATE WATCH



The Best Software Architecture

"The best software architecture 'knows' what changes often and makes that easy."

- Paul Clements

Adaptable Clients and Evolvable APIs

Mike Amundsen
API Academy / CA
@mamund