Formalising Solutions to REST API Practices as Design (Anti)patterns

By Van Tuan Tran
Under supervision of Dr. Yann-Gaël Guéhéneuc

Examining Committee:

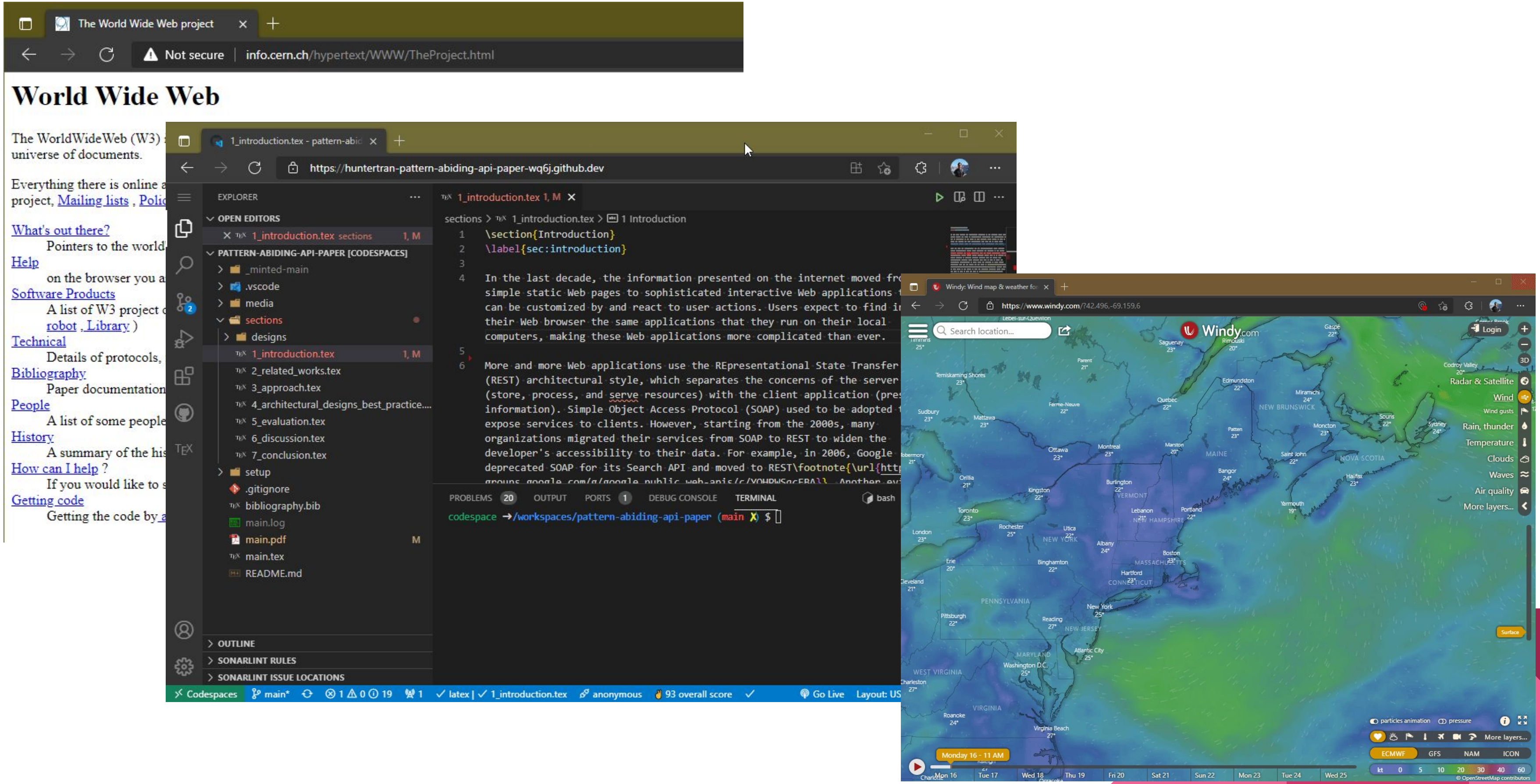
- Dr. Joey Paquet - Chair
- Dr. Brigitte Jaumard - Examiner

Table of Contents

1. Introduction
2. Related Work
3. Approach
4. (Anti)patterns
5. Evaluations
6. Discussion
7. Conclusion

1 - Introduction

1 - Introduction - Context



The World Wide Web project
Not secure
info.cern.ch/hypertext/WWW/TheProject.html

World Wide Web

The WorldWideWeb (W3)
universe of documents.

Everything there is online a
project, Mailing lists , Polic

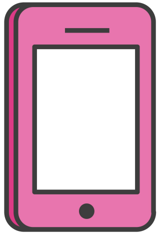
What's out there?
Pointers to the world'
Help
on the browser you a
Software Products
A list of W3 project c
robot , Library)
Technical
Details of protocols,
Bibliography
Paper documentation
People
A list of some people
History
A summary of the his
How can I help ?
If you would like to s
Getting code
Getting the code by a

1_introduction.tex - pattern-abid
https://huntertran-pattern-abiding-api-paper-wq6j.github.dev
EXPLORER
OPEN EDITORS
1_introduction.tex sections 1, M
PATTERN-ABIDING-API-PAPER [CODESPACES]
_minted-main
.vscode
media
sections
designs
1_introduction.tex 1, M
2_related_works.tex
3_approach.tex
4_architectural_designs_best_practice....
5_evaluation.tex
6_discussion.tex
7_conclusion.tex
setup
.gitignore
bibliography.bib
main.log
main.pdf M
main.tex
README.md
OUTLINE
SONARLINT RULES
SONARLINT ISSUE LOCATIONS
1_introduction.tex 1, M
sections > 1_introduction.tex > 1 Introduction

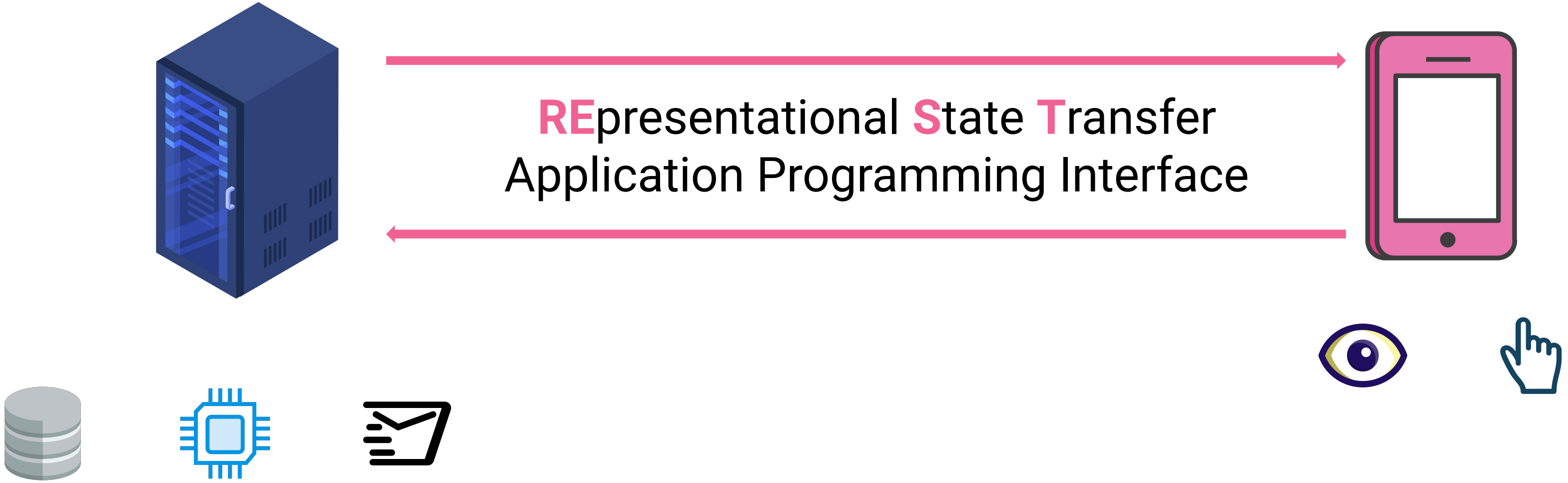
```
\section{Introduction}
\label{sec:introduction}

In the last decade, the information presented on the internet moved fro
simple static Web pages to sophisticated interactive Web applications t
can be customized by and react to user actions. Users expect to find in
their Web browser the same applications that they run on their local
computers, making these Web applications more complicated than ever.

More and more Web applications use the REpresentational State Transfer
(REST) architectural style, which separates the concerns of the server
(store, process, and serve resources) with the client application (pres
information). Simple Object Access Protocol (SOAP) used to be adopted t
expose services to clients. However, starting from the 2000s, many
organizations migrated their services from SOAP to REST to widen the
developer's accessibility to their data. For example, in 2006, Google
deprecated SOAP for its Search API and moved to REST\footnote{\url{http
groups.google.com/g/google-public-web-apis/c/YOHPWSqcFBA}}. Another evi
```

```
codespace →/workspaces/pattern-abiding-api-paper (main ✗) $ 
```

1 - Introduction - Context



1 - Introduction - Context



REpresentational State Transfer
Application Programming Interface

1 - Introduction - Context

👍

REpresentational **S**tate **T**ransfer
Application Programming Interface

👎

1 - Introduction - Terms

Good practice
good way to implement the REST API

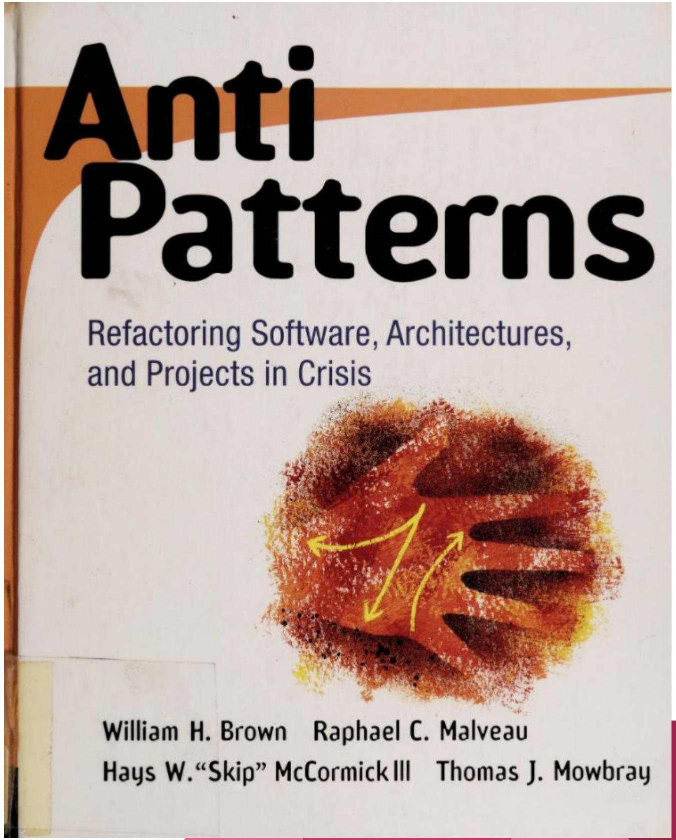
Practice *noun*
/ˈpræktɪs/
way of doing something

Bad practice
Bad way to implement the REST API

https://www.oxfordlearnersdictionaries.com/us/definition/english/practice_1

1 - Introduction - Terms

Anti-pattern		
"A commonly occurring solution to a problem that generates negative consequences"		
Problem	Bad solution(s)	Alternative good solution(s)



Anti Patterns

Refactoring Software, Architectures, and Projects in Crisis

William H. Brown Raphael C. Malveau
Hays W. "Skip" McCormick III Thomas J. Mowbray

1 - Introduction

We propose 3 contributions

- Review academic and gray literature to identify REST API practices
- Propose practical solutions to these practices
- Validate our solutions with surveys and interviews

2 - Related Work

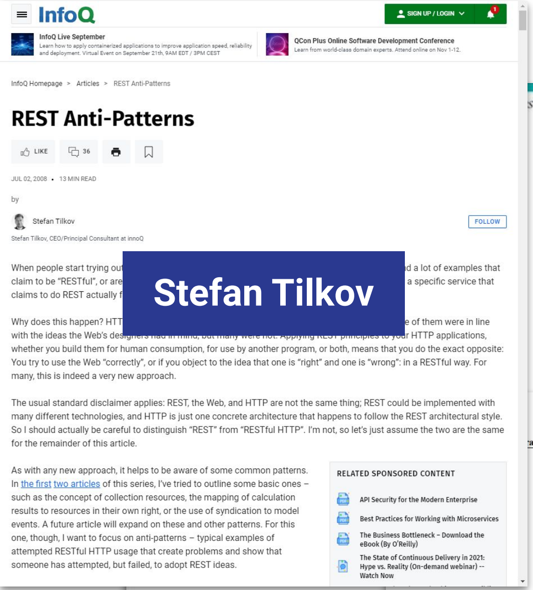
2 - Related Work - Practices

2008 - Breaking Self-descriptiveness
Forgetting Hypermedia
Ignoring MIME type
Ignoring status code
Misusing cookies
Use the wrong HTTP Verbs

2011 - CRUDYs URIs
Use the wrong HTTP Verbs
Ignoring MIME Type

2010 - Content Negotiations

ntext-less resource na
n-hierarchical nodes
gularized/Pluralized N
t Pagination
l Versioning



InfoQ

SIGN UP / LOGIN

InfoQ Live September
Learn how to apply containerized applications to improve application speed, reliability and deployment. Virtual Event on September 21th, 9AM EDT / 3PM CEST

QCon Plus Online Software Development Conference
Learn from world-class domain experts. Attend online on Nov 1-12.

InfoQ Homepage > Articles > REST Anti-Patterns

REST Anti-Patterns

LIKE 36

JUL 02, 2008 • 13 MIN READ

by

Stefan Tilkov FOLLOW

Stefan Tilkov, CEO/Principal Consultant at innoQ

When people start trying out d a lot of examples that claim to be "RESTful", or are a specific service that claims to do REST actually f

Why does this happen? HTT e of them were in line with the ideas the Web's designers had in mind, but many were not. Applying REST principles to your HTTP applications, whether you build them for human consumption, for use by another program, or both, means that you do the exact opposite: You try to use the Web "correctly", or if you object to the idea that one is "right" and one is "wrong": in a RESTful way. For many, this is indeed a very new approach.

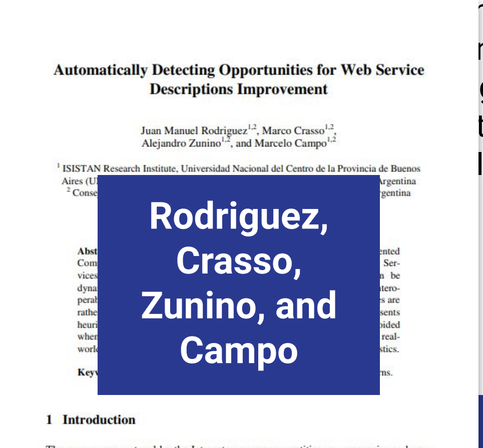
The usual standard disclaimer applies: REST, the Web, and HTTP are not the same thing; REST could be implemented with many different technologies, and HTTP is just one concrete architecture that happens to follow the REST architectural style. So I should actually be careful to distinguish "REST" from "RESTful HTTP". I'm not, so let's just assume the two are the same for the remainder of this article.

As with any new approach, it helps to be aware of some common patterns. In the first two articles of this series, I've tried to outline some basic ones – such as the concept of collection resources, the mapping of calculation results to resources in their own right, or the use of syndication to model events. A future article will expand on these and other patterns. For this one, though, I want to focus on anti-patterns – typical examples of attempted RESTful HTTP usage that create problems and show that someone has attempted, but failed, to adopt REST ideas.

RELATED SPONSORED CONTENT

- API Security for the Modern Enterprise
- Best Practices for Working with Microservices
- The Business Bottleneck – Download the eBook (By O'Reilly)
- The State of Continuous Delivery in 2021: Hype vs. Reality (On-demand webinar) -- Watch Now

Stefan Tilkov



Automatically Detecting Opportunities for Web Service Descriptions Improvement

Juan Manuel Rodriguez[1,2], Marco Crasso[1,2],
Alejandro Zunino[1,2], and Marcelo Campo[1,2]

[1] ISISTAN Research Institute, Universidad Nacional del Centro de la Provincia de Buenos Aires (U Argentina
[2] Conse rgentina

Abst ented
Com Ser-
vices n be
dyna tero-
perab s are
rathe sents
heuri oided
when real-
worl stics.

Key rns.

1 Introduction

Rodriguez, Crasso, Zunino, and Campo

O'REILLY

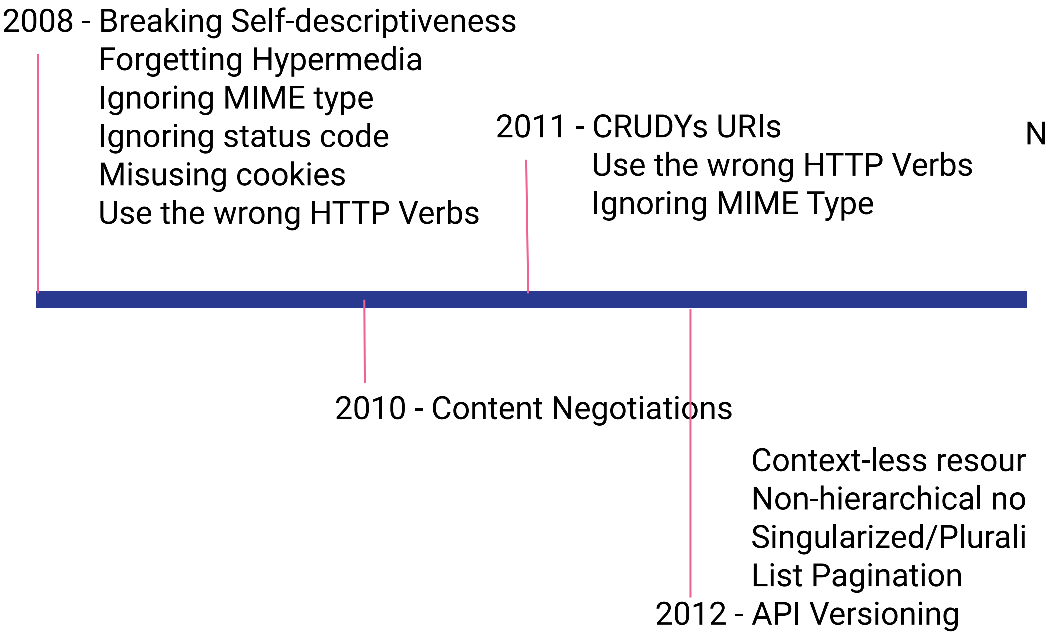
RESTful Service Best Practices
Recommendations for Creating Web Services

Todd Fredrich
Pearson eCollege
@tfredrich
www.RestApiTutorial.com

08/02/13 www.RestApiTutorial.com Page 1 of 40

Fredrich

2 - Related Work - Practices



2008 - Breaking Self-descriptiveness
Forgetting Hypermedia
Ignoring MIME type
Ignoring status code
Misusing cookies
Use the wrong HTTP Verbs

2011 - CRUDYs URIs
Use the wrong HTTP Verbs
Ignoring MIME Type

2010 - Content Negotiations

Context-less resour
Non-hierarchical no
Singularized/Plurali
List Pagination
2012 - API Versioning

Non-pertinent documentation - 2017

International Journal of Cooperative Information Systems
Vol. 26, No. 2 (2017) 1742001 (37 pages)
© World Scientific Publishing Company
DOI: 10.1142/S0218843017420011

World Scientific
www.worldscientific.com

Semantic Analysis of RESTful APIs for the Detection of Linguistic Patterns and Antipatterns

Francis Palma*
Department of Electrical and Computer Engineering
Concordia University
1515 St. Catherine West, Montréal, QC, Canada H3G 2W1
francispalmaphd@gmail.com

Javier Gonzalez-Huerta
Software Engineering Research Lab Sweden
Blekinge Institute of Technology
Campus Gräsvik, SE-371 79 Karlskrona, Sweden

Mohamed Founi, Naouel Moha and Guy Tremblay

Palma et al.

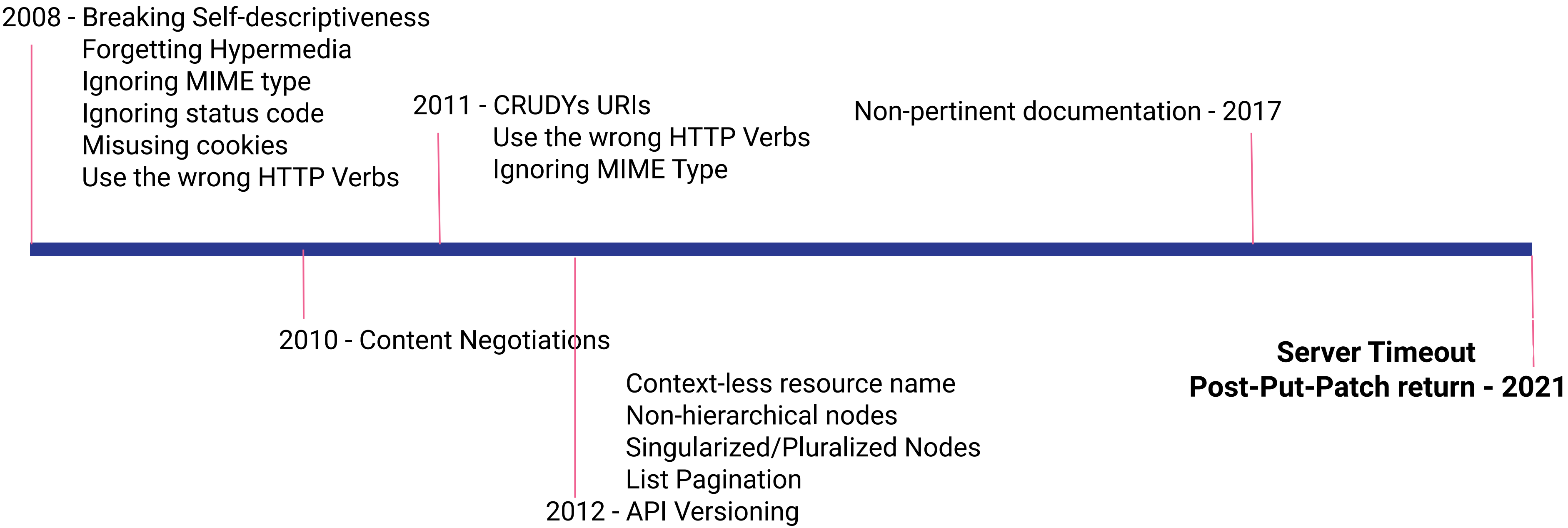
Received 15 December 2016
Accepted 21 December 2016
Published 16 May 2017

Identifier lexicon may have a direct impact on software understandability and reusability and, thus, on the quality of the final software product. Understandability and reusability are two important characteristics of software quality. REpresentational State Transfer (REST) style is becoming a *de facto* standard adopted by software organizations to build their Web applications. Understandable and reusable Uniform Resource Identifers (URIs) are important to attract client developers of RESTful APIs because *good* URIs support the client developers to understand and reuse the APIs. Consequently, the use of proper lexicon in RESTful APIs has also a direct impact on the quality of Web applications that integrate these APIs. *Linguistic antipatterns* represent poor practices in the naming, documentation, and choice of identifiers in the APIs as opposed to *linguistic patterns* that represent the corresponding best practices. In this paper, we present the Semantic Analysis of RESTful APIs (SARA) approach that employs both syntactic and semantic analyses for the detection of linguistic patterns and antipatterns in RESTful APIs. We provide detailed definitions of 12 linguistic patterns and antipatterns and define and apply their detection algorithms on 18 widely-used RESTful APIs, including `Facebook`, `Twitter`, and `Dropbox`. Our detection results show that linguistic patterns and antipatterns do occur in major RESTful APIs in particular in the form of poor

*Corresponding author.

1742001-1

2 - Related Work - Practices



2008 - Breaking Self-descriptiveness
Forgetting Hypermedia
Ignoring MIME type
Ignoring status code
Misusing cookies
Use the wrong HTTP Verbs

2010 - Content Negotiations

2011 - CRUDYs URIs
Use the wrong HTTP Verbs
Ignoring MIME Type

Context-less resource name
Non-hierarchical nodes
Singularized/Pluralized Nodes
List Pagination
2012 - API Versioning

Non-pertinent documentation - 2017

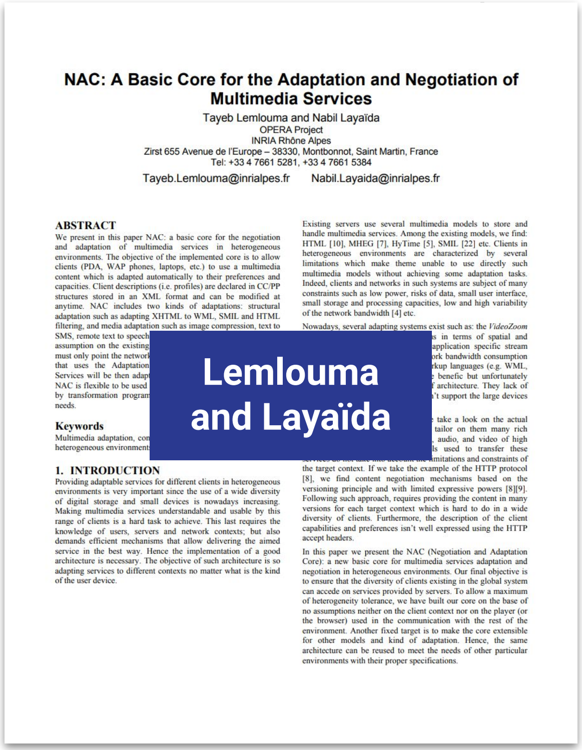
Server Timeout
Post-Put-Patch return - 2021

2 - Related Work - Existing Solutions

- Content Negotiation
- Forgetting Hypermedia
- API Versioning
- Server Timeout
- Response Caching
- List Pagination

2 - Related Work - Existing Solutions

- **Content Negotiation**



NAC: A Basic Core for the Adaptation and Negotiation of Multimedia Services

Tayeb Lemlouma and Nabil Layaïda
OPERA Project
INRIA Rhône Alpes
Zirst 655 Avenue de l'Europe – 38330, Montbonnot, Saint Martin, France
Tel: +33 4 7661 5281, +33 4 7661 5384
Tayeb.Lemlouma@inrialpes.fr Nabil.Layaida@inrialpes.fr

ABSTRACT

We present in this paper NAC: a basic core for the negotiation and adaptation of multimedia services in heterogeneous environments. The objective of the implemented core is to allow clients (PDA, WAP phones, laptops, etc.) to use a multimedia content which is adapted automatically to their preferences and capacities. Client descriptions (i.e. profiles) are declared in CC/PP structures stored in an XML format and can be modified at anytime. NAC includes two kinds of adaptations: structural adaptation such as adapting XHTML to WML, SMIL and HTML filtering, and media adaptation such as image compression, text to
SMS, remote text to speech
assumption on the existing
must only point the network
that uses the Adaptation
Services will be then adapt
NAC is flexible to be used
by transformation program
needs.

Keywords

Multimedia adaptation, con
heterogeneous environment

1. INTRODUCTION

Providing adaptable services for different clients in heterogeneous environments is very important since the use of a wide diversity of digital storage and small devices is nowadays increasing. Making multimedia services understandable and usable by this range of clients is a hard task to achieve. This last requires the knowledge of users, servers and network contexts; but also demands efficient mechanisms that allow delivering the aimed service in the best way. Hence the implementation of a good architecture is necessary. The objective of such architecture is so adapting services to different contexts no matter what is the kind of the user device.

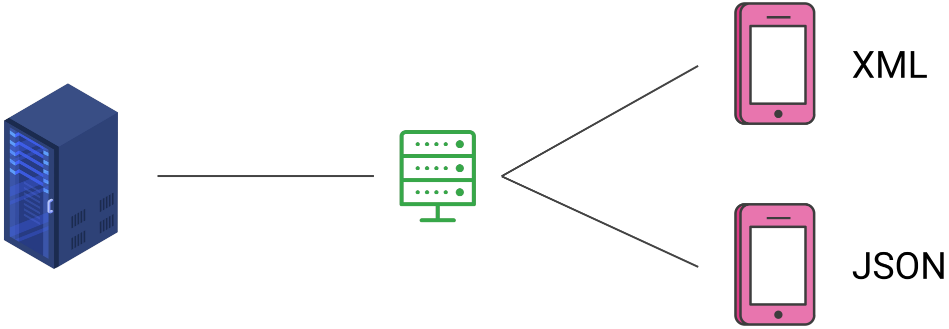
Existing servers use several multimedia models to store and handle multimedia services. Among the existing models, we find: HTML [10], MHEG [7], HyTime [5], SMIL [22] etc. Clients in heterogeneous environments are characterized by several limitations which make theme unable to use directly such multimedia models without achieving some adaptation tasks. Indeed, clients and networks in such systems are subject of many constraints such as low power, risks of data, small user interface, small storage and processing capacities, low and high variability of the network bandwidth [4] etc.

Nowadays, several adapting systems exist such as: the *VideoZoom*
s in terms of spatial and
application specific stream
ork bandwidth consumption
rkup languages (e.g. WML,
e benefic but unfortunately
f architecture. They lack of
't support the large devices

e take a look on the actual
tailor on them many rich
, audio, and video of high
ls used to transfer these
imitations and constraints of
the target context. If we take the example of the HTTP protocol [8], we find content negotiation mechanisms based on the versioning principle and with limited expressive powers [8][9]. Following such approach, requires providing the content in many versions for each target context which is hard to do in a wide diversity of clients. Furthermore, the description of the client capabilities and preferences isn't well expressed using the HTTP accept headers.

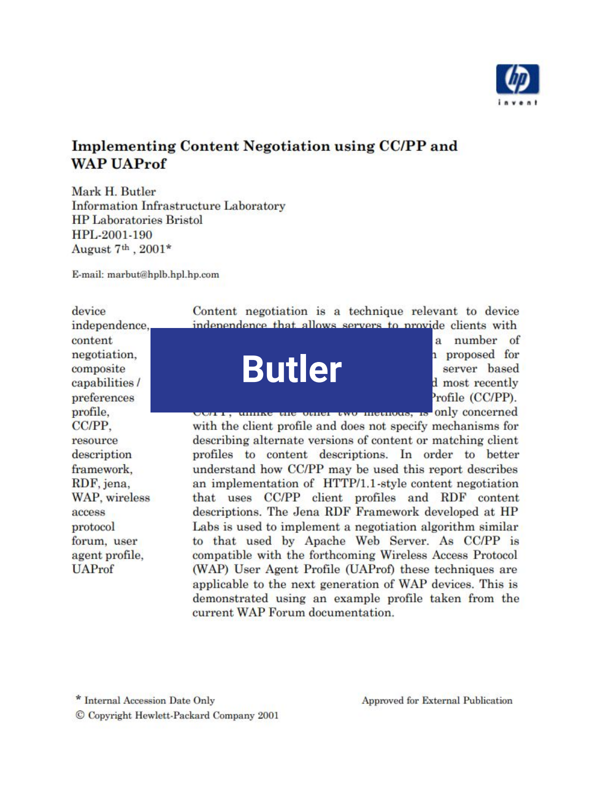
In this paper we present the NAC (Negotiation and Adaptation Core): a new basic core for multimedia services adaptation and negotiation in heterogeneous environments. Our final objective is to ensure that the diversity of clients existing in the global system can accede on services provided by servers. To allow a maximum of heterogeneity tolerance, we have built our core on the base of no assumptions neither on the client context nor on the player (or the browser) used in the communication with the rest of the environment. Another fixed target is to make the core extensible for other models and kind of adaptation. Hence, the same architecture can be reused to meet the needs of other particular environments with their proper specifications.

Lemlouma and Layaïda



2 - Related Work - Existing Solutions

- **Content Negotiation**



Implementing Content Negotiation using CC/PP and WAP UAProf

Mark H. Butler
Information Infrastructure Laboratory
HP Laboratories Bristol
HPL-2001-190
August 7th , 2001*

E-mail: marbut@hplb.hpl.hp.com

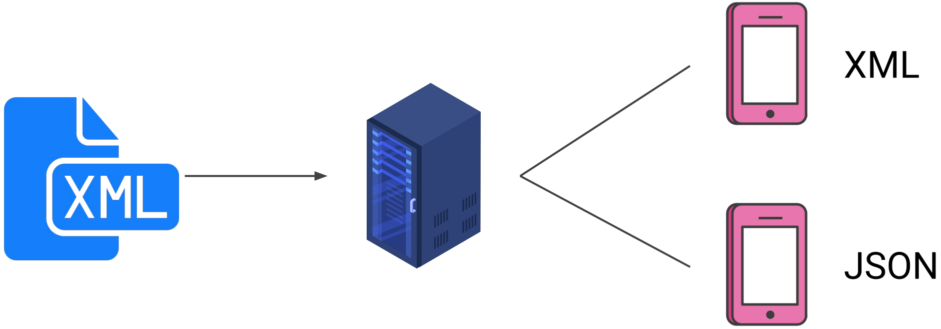
device independence, content negotiation, composite capabilities / preferences profile, CC/PP, resource description framework, RDF, jena, WAP, wireless access protocol forum, user agent profile, UAProf

Content negotiation is a technique relevant to device independence that allows servers to provide clients with a number of proposed for server based d most recently rofile (CC/PP). only concerned with the client profile and does not specify mechanisms for describing alternate versions of content or matching client profiles to content descriptions. In order to better understand how CC/PP may be used this report describes an implementation of HTTP/1.1-style content negotiation that uses CC/PP client profiles and RDF content descriptions. The Jena RDF Framework developed at HP Labs is used to implement a negotiation algorithm similar to that used by Apache Web Server. As CC/PP is compatible with the forthcoming Wireless Access Protocol (WAP) User Agent Profile (UAProf) these techniques are applicable to the next generation of WAP devices. This is demonstrated using an example profile taken from the current WAP Forum documentation.

* Internal Accession Date Only

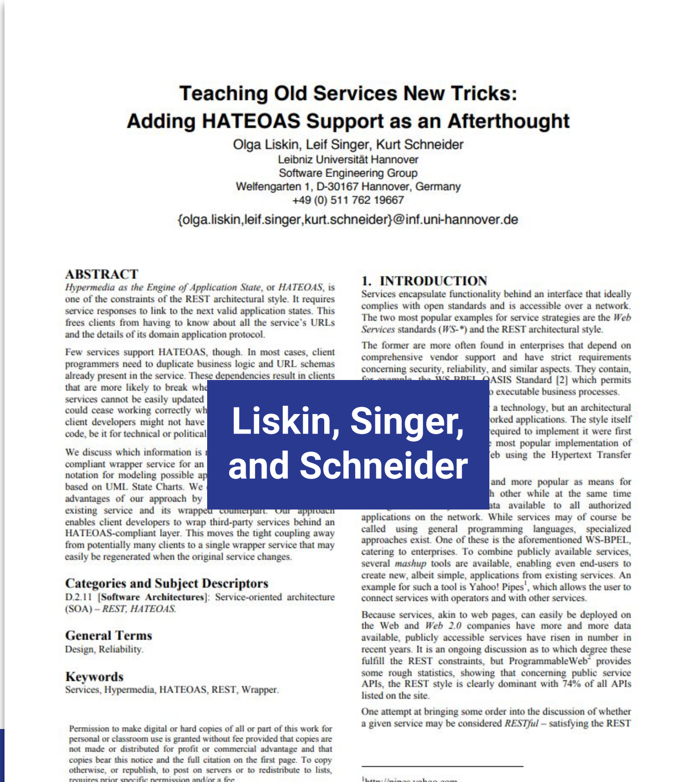
Approved for External Publication

Butler



2 - Related Work - Existing Solutions

- Content Negotiation
- **Forgetting Hypermedia**



Teaching Old Services New Tricks: Adding HATEOAS Support as an Afterthought

Olga Liskin, Leif Singer, Kurt Schneider
Leibniz Universität Hannover
Software Engineering Group
Welfengarten 1, D-30167 Hannover, Germany
+49 (0) 511 762 19667
{olga.liskin,leif.singer,kurt.schneider}@inf.uni-hannover.de

ABSTRACT

Hypermedia as the Engine of Application State, or *HATEOAS*, is one of the constraints of the REST architectural style. It requires service responses to link to the next valid application states. This frees clients from having to know about all the service's URLs and the details of its domain application protocol.

Few services support HATEOAS, though. In most cases, client programmers need to duplicate business logic and URL schemas already present in the service. These dependencies result in clients
that are more likely to break whe
services cannot be easily updated
could cease working correctly wh
client developers might not have
code, be it for technical or political

We discuss which information is
compliant wrapper service for an
notation for modeling possible ap
based on UML State Charts. We
advantages of our approach by
existing service and its wrapped counterpart. Our approach enables client developers to wrap third-party services behind an HATEOAS-compliant layer. This moves the tight coupling away from potentially many clients to a single wrapper service that may easily be regenerated when the original service changes.

Liskin, Singer, and Schneider

Categories and Subject Descriptors
D.2.11 [**Software Architectures**]: Service-oriented architecture (SOA) – *REST, HATEOAS.*

General Terms
Design, Reliability.

Keywords
Services, Hypermedia, HATEOAS, REST, Wrapper.

1. INTRODUCTION

Services encapsulate functionality behind an interface that ideally complies with open standards and is accessible over a network. The two most popular examples for service strategies are the *Web Services* standards (*WS-**) and the REST architectural style.

The former are more often found in enterprises that depend on comprehensive vendor support and have strict requirements concerning security, reliability, and similar aspects. They contain, for example, the WS-BPEL OASIS Standard [2] which permits
o executable business processes.

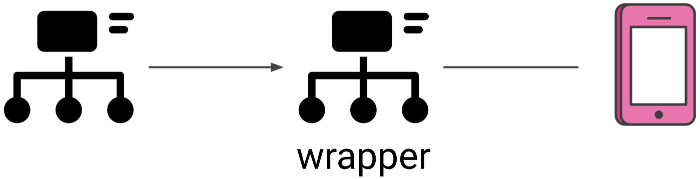
a technology, but an architectural
orked applications. The style itself
equired to implement it were first
most popular implementation of
eb using the Hypertext Transfer

and more popular as means for
h other while at the same time
ata available to all authorized
applications on the network. While services may of course be called using general programming languages, specialized approaches exist. One of these is the aforementioned WS-BPEL, catering to enterprises. To combine publicly available services, several *mashup* tools are available, enabling even end-users to create new, albeit simple, applications from existing services. An example for such a tool is Yahoo! Pipes[1], which allows the user to connect services with operators and with other services.

Because services, akin to web pages, can easily be deployed on the Web and *Web 2.0* companies have more and more data available, publicly accessible services have risen in number in recent years. It is an ongoing discussion as to which degree these fulfill the REST constraints, but ProgrammableWeb[2] provides some rough statistics, showing that concerning public service APIs, the REST style is clearly dominant with 74% of all APIs listed on the site.

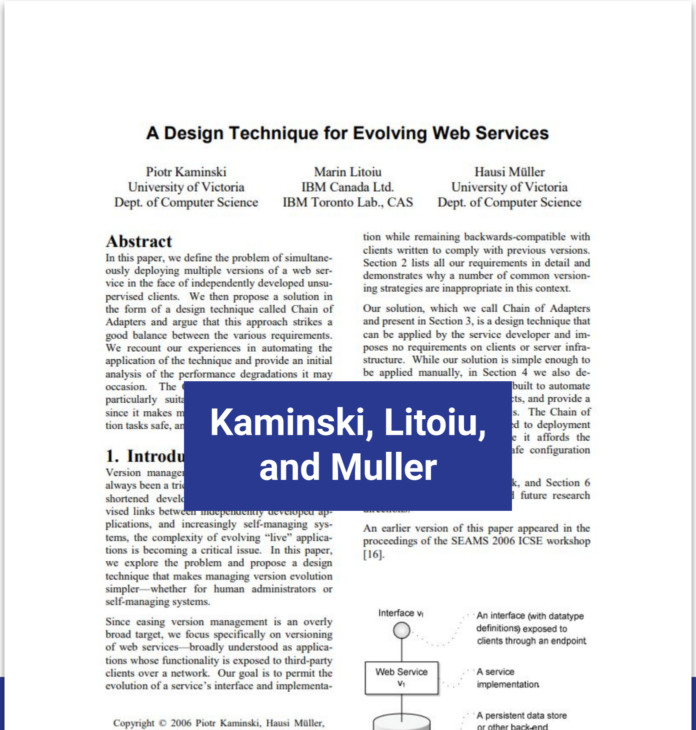
One attempt at bringing some order into the discussion of whether a given service may be considered *RESTful* – satisfying the REST

[1] http://pipes.yahoo.com



2 - Related Work - Existing Solutions

- Content Negotiation
- Forgetting Hypermedia
- **API Versioning**



A Design Technique for Evolving Web Services

Piotr Kaminski
University of Victoria
Dept. of Computer Science

Marin Litoiu
IBM Canada Ltd.
IBM Toronto Lab., CAS

Hausi Müller
University of Victoria
Dept. of Computer Science

Abstract

In this paper, we define the problem of simultaneously deploying multiple versions of a web service in the face of independently developed unsupervised clients. We then propose a solution in the form of a design technique called Chain of Adapters and argue that this approach strikes a good balance between the various requirements. We recount our experiences in automating the application of the technique and provide an initial analysis of the performance degradations it may occasion. The
particularly suita
since it makes m
tion tasks safe, an

1. Introdu

Version manage
always been a tri
shortened develo
vised links between independently developed applications, and increasingly self-managing systems, the complexity of evolving "live" applications is becoming a critical issue. In this paper, we explore the problem and propose a design technique that makes managing version evolution simpler—whether for human administrators or self-managing systems.

Since easing version management is an overly broad target, we focus specifically on versioning of web services—broadly understood as applications whose functionality is exposed to third-party clients over a network. Our goal is to permit the evolution of a service's interface and implementation while remaining backwards-compatible with clients written to comply with previous versions. Section 2 lists all our requirements in detail and demonstrates why a number of common versioning strategies are inappropriate in this context.

Our solution, which we call Chain of Adapters and present in Section 3, is a design technique that can be applied by the service developer and imposes no requirements on clients or server infrastructure. While our solution is simple enough to be applied manually, in Section 4 we also de-
built to automate
cts, and provide a
s. The Chain of
d to deployment
e it affords the
afe configuration

k, and Section 6
d future research

An earlier version of this paper appeared in the proceedings of the SEAMS 2006 ICSE workshop [16].

Interface v_1

Web Service v_1

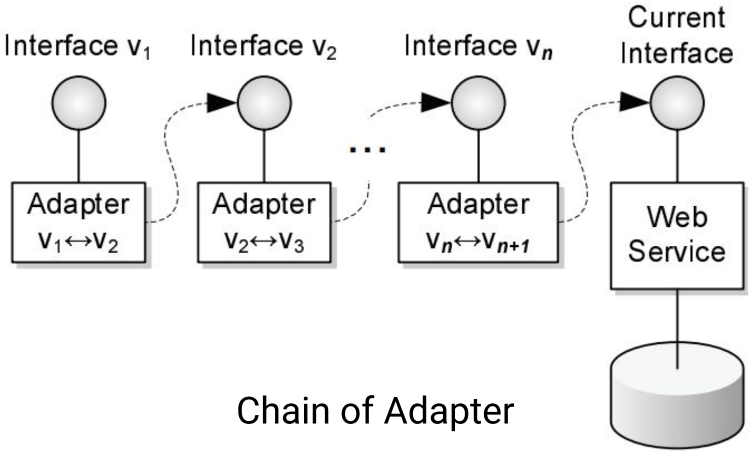
An interface (with datatype definitions) exposed to clients through an endpoint

A service implementation

A persistent data store or other back-end

Copyright © 2006 Piotr Kaminski, Hausi Müller,

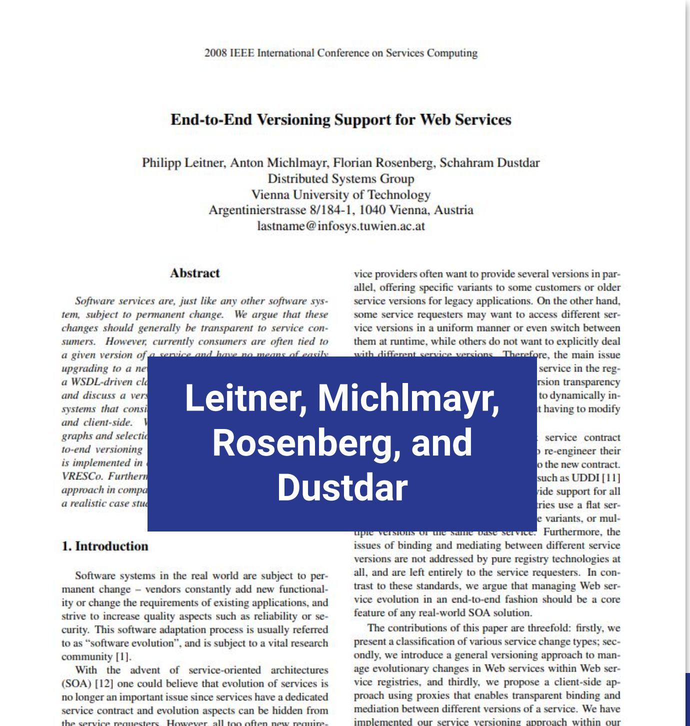
Kaminski, Litoiu, and Muller


Interface v_1
Interface v_2
Interface v_n
Current Interface
...
Adapter $v_1 \leftrightarrow v_2$
Adapter $v_2 \leftrightarrow v_3$
Adapter $v_n \leftrightarrow v_{n+1}$
Web Service

Chain of Adapter

2 - Related Work - Existing Solutions

- Content Negotiation
- Forgetting Hypermedia
- **API Versioning**



2008 IEEE International Conference on Services Computing

End-to-End Versioning Support for Web Services

Philipp Leitner, Anton Michlmayr, Florian Rosenberg, Schahram Dustdar
Distributed Systems Group
Vienna University of Technology
Argentinierstrasse 8/184-1, 1040 Vienna, Austria
lastname@infosys.tuwien.ac.at

Abstract

Software services are, just like any other software system, subject to permanent change. We argue that these changes should generally be transparent to service consumers. However, currently consumers are often tied to a given version of a service and have no means of easily
upgrading to a ne
a WSDL-driven cl
and discuss a vers
systems that consi
and client-side. W
graphs and selecti
to-end versioning
is implemented in
VRESCo. Furtherm
approach in compa
a realistic case stu

1. Introduction

Software systems in the real world are subject to permanent change – vendors constantly add new functionality or change the requirements of existing applications, and strive to increase quality aspects such as reliability or security. This software adaptation process is usually referred to as "software evolution", and is subject to a vital research community [1].

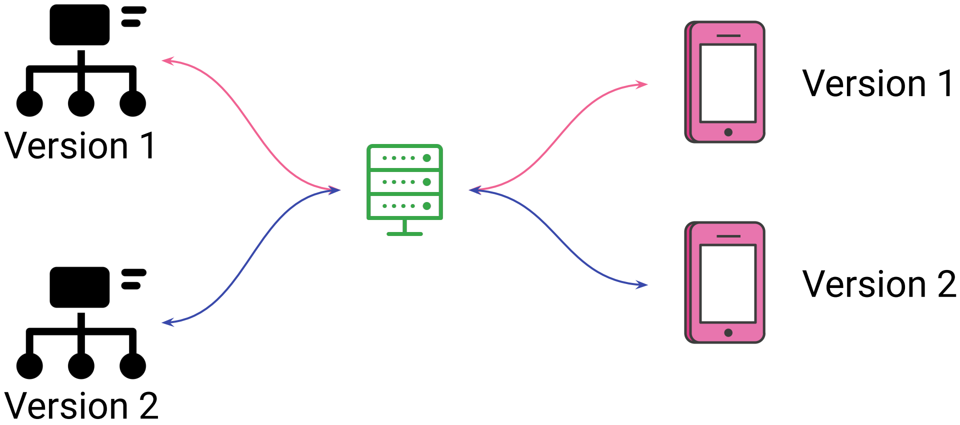
With the advent of service-oriented architectures (SOA) [12] one could believe that evolution of services is no longer an important issue since services have a dedicated service contract and evolution aspects can be hidden from the service requesters. However, all too often new require-

vice providers often want to provide several versions in parallel, offering specific variants to some customers or older service versions for legacy applications. On the other hand, some service requesters may want to access different service versions in a uniform manner or even switch between them at runtime, while others do not want to explicitly deal with different service versions. Therefore, the main issue
service in the reg-
rsion transparency
to dynamically in-
t having to modify

service contract
re-engineer their
o the new contract.
such as UDDI [11]
vide support for all
ries use a flat ser-
e variants, or mul-
tiple versions of the same base service. Furthermore, the issues of binding and mediating between different service versions are not addressed by pure registry technologies at all, and are left entirely to the service requesters. In contrast to these standards, we argue that managing Web service evolution in an end-to-end fashion should be a core feature of any real-world SOA solution.

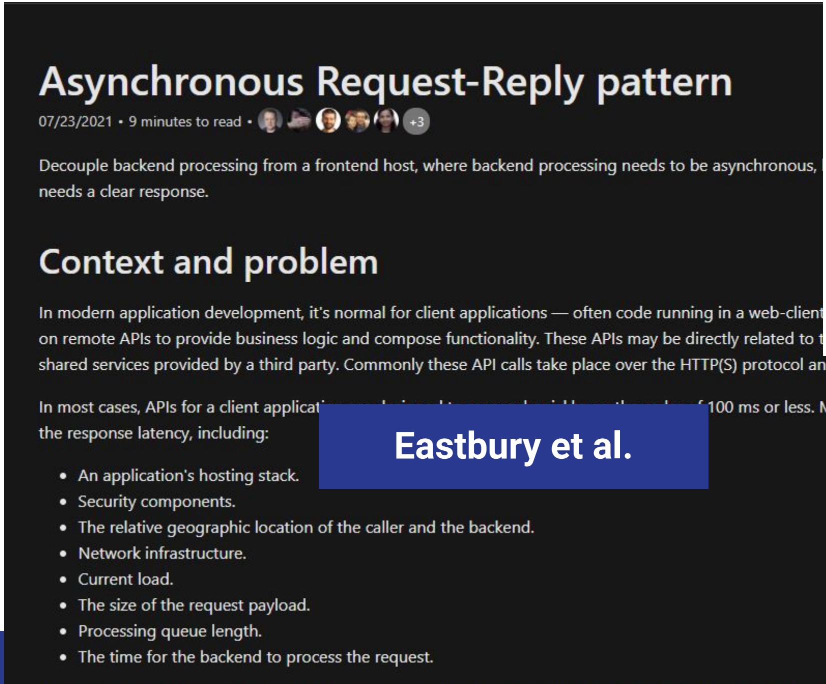
The contributions of this paper are threefold: firstly, we present a classification of various service change types; secondly, we introduce a general versioning approach to manage evolutionary changes in Web services within Web service registries, and thirdly, we propose a client-side approach using proxies that enables transparent binding and mediation between different versions of a service. We have implemented our service versioning approach within our

Leitner, Michlmayr, Rosenberg, and Dustdar



2 - Related Work - Existing Solutions

- Content Negotiation
- Forgetting Hypermedia
- API Versioning
- **Server Timeout**



Asynchronous Request-Reply pattern

07/23/2021 • 9 minutes to read • +3

Decouple backend processing from a frontend host, where backend processing needs to be asynchronous, needs a clear response.

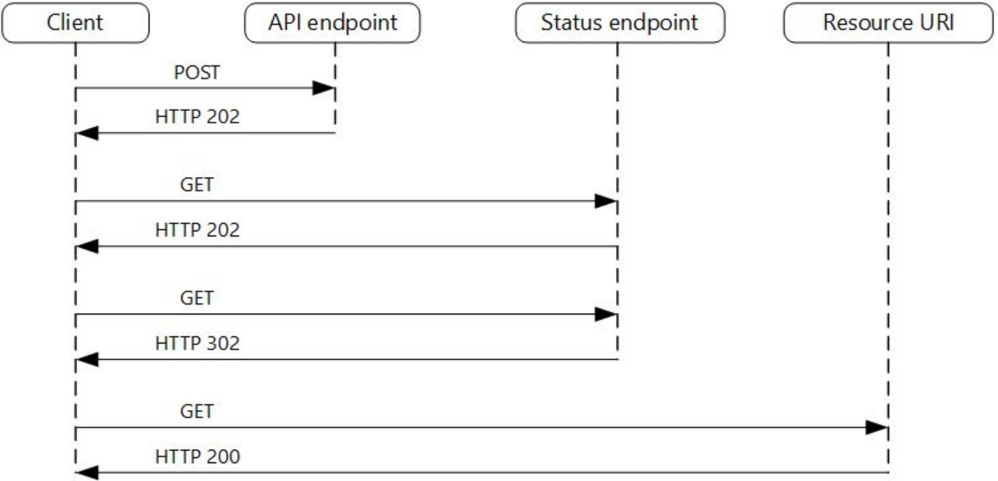
Context and problem

In modern application development, it's normal for client applications — often code running in a web-client on remote APIs to provide business logic and compose functionality. These APIs may be directly related to t shared services provided by a third party. Commonly these API calls take place over the HTTP(S) protocol and follow REST semantics.

In most cases, APIs for a client applicat 100 ms or less. Many factors can affect the response latency, including:

- An application's hosting stack.
- Security components.
- The relative geographic location of the caller and the backend.
- Network infrastructure.
- Current load.
- The size of the request payload.
- Processing queue length.
- The time for the backend to process the request.

Eastbury et al.



2 - Related Work - Existing Solutions

- Content Negotiation
- Forgetting Hypermedia
- API Versioning
- Server Timeout
- **Response Caching**
- **List Pagination**

ASP.NET Core

spring®

https://github.com/campusMVP/dotnetCoreLogoPack - https://spring.io/trademarks

3 - Approach

2 - Approach - Literature Review

InfoQ

SIGN UP / LOGIN

InfoQ Live September
Learn how to apply containerized applications to improve application speed, reliability and deployment. Virtual Event on September 21th, 9AM EDT / 3PM CEST

QCon Plus Online Software Development Conference
Learn from world-class domain experts. Attend online on Nov 1-12.

InfoQ Homepage > Articles > REST Anti-Patterns

REST A

JUL 02, 2008 • 13 MIN

by

Stefan Tilkov

Stefan Tilkov, CEO/Princ

Automatically Detecting Opportunities for Web Service Descriptions Improvement

Juan Manuel Rodriguez[1,2], Marco Crasso[1,2], Alejandro Zunino[1,2], and Marcelo Campo[1,2]

Designing Consistent RESTful Web Service Interfaces

O'REILLY

RESTful Service Best Practices

RESTful Service Best Practices

Recommendations for Creating Web Services

International Journal of Cooperative Information Systems
Vol. 26, No. 2 (2017) 1742001 (37 pages)
© World Scientific Publishing Company
DOI: 10.1142/S0218843017420011

World Scientific
www.worldscientific.com

Semantic Analysis of RESTful APIs for the Detection of Linguistic Patterns and Antipatterns

Francis Palma*
Department of Electrical and Computer Engineering
Concordia University
1515 St. Catherine West, Montréal, QC, Canada H3G 2W1
francispalmaphd@gmail.com

Javier Gonzalez-Huerta

NAC: A Basic Core for the Adaptation and Negotiation of Multimedia Services

ABSTRACT

Keywords

1. INTRODUC

Impleme
WAP UA

Mark H. Bu
Information
HP Laborat
HPL-2001-
August 7th

E-mail: marbu

Teaching Old Services New Tricks: Adding HATEOAS Support as an Afterthought

ABSTRACT

Categories an

General Term

Keywords

A Design Technique for Evolving Web Services

Abstra

1. Intr

2008 IEEE International Conference on

End-to-End Versioning Suppor

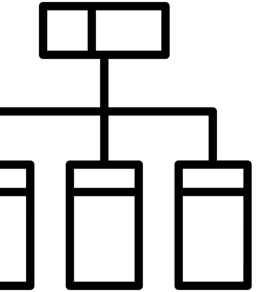
Asynchrono

07/23/2021 • 9 minutes to read •

Decouple backend processing
needs a clear response.

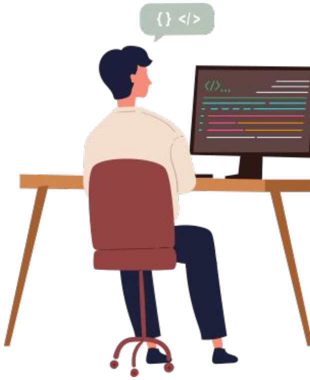
Context and p

2 - Approach - Categorization



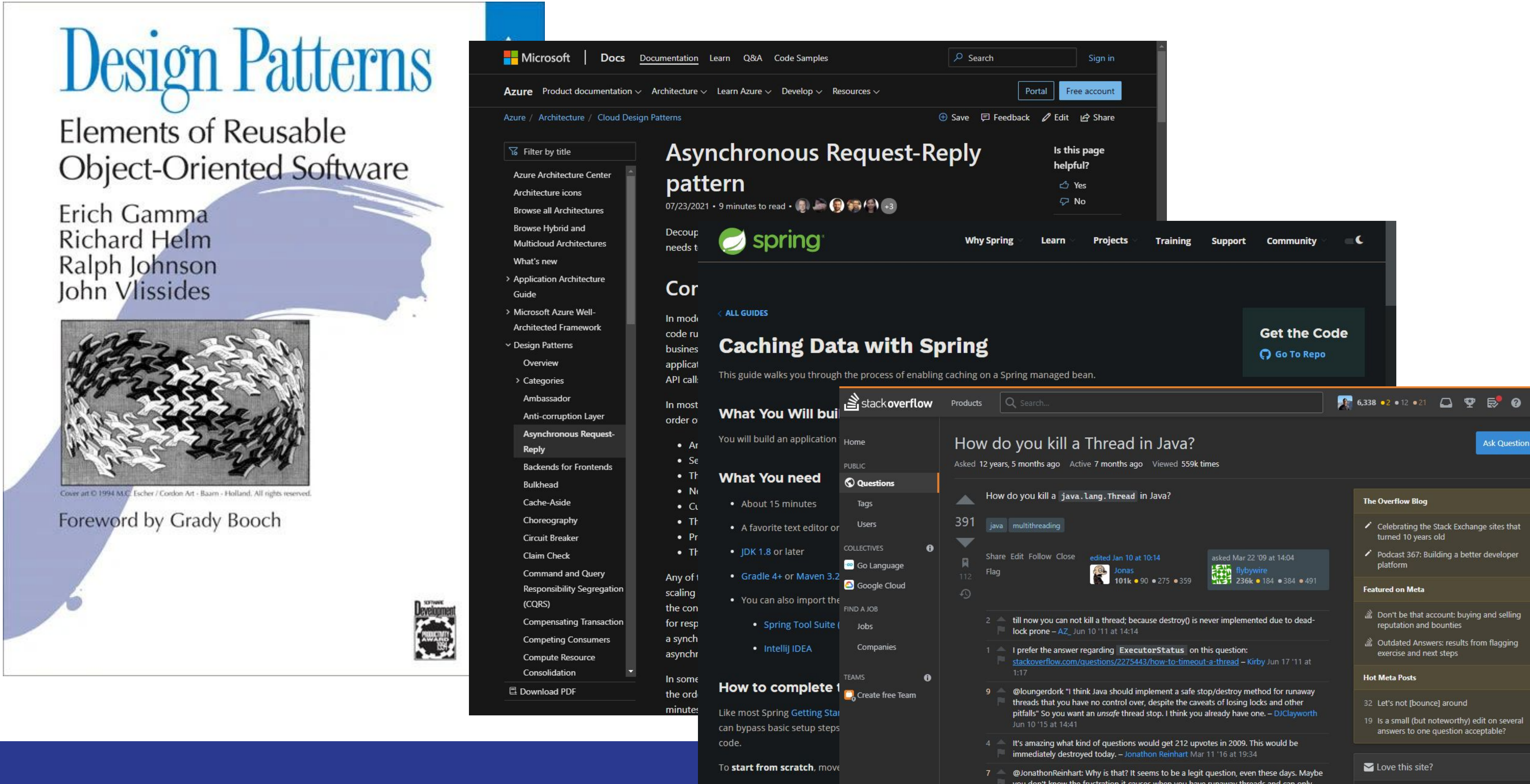
Architectural

Technical	Non-technical
Content negotiation	Entity Endpoint
Endpoint redirection	Contextless Resource name
Entity Linking	Non-hierarchical Nodes
Response caching	Amorphous URIs
API Versioning	CRUDy URIs
Server Timeout	Singularized & Pluralized Nodes
POST-PUT-PATCH Return	Non-pertinent Documentation
List Pagination	Breaking Self-descriptiveness
	Ignoring status code
	Using the wrong HTTP Verbs
	Misusing Cookies



Devs pay attention

2 - Approach - Looking for Solutions



Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

Asynchronous Request-Reply pattern

07/23/2021 • 9 minutes to read • +3

Is this page helpful?

Yes

No

Decoup needs t

Cor

In mod code ru busine applica API call

In most order o

- Ar
- Se
- Th
- N
- Cu
- Th
- Pr
- Th

Any of scaling the con for resp a synch asynchr

In some the ord minute

Caching Data with Spring

This guide walks you through the process of enabling caching on a Spring managed bean.

Get the Code

Go To Repo

What You Will bui

You will build an application

What You need

- About 15 minutes
- A favorite text editor or
- JDK 1.8 or later
- Gradle 4+ or Maven 3.2
- You can also import the
  - Spring Tool Suite (
  - IntelliJ IDEA

How to complete t

Like most Spring Getting Star can bypass basic setup steps code.

To **start from scratch**, move

How do you kill a Thread in Java?

Asked 12 years, 5 months ago Active 7 months ago Viewed 559k times

391

How do you kill a `java.lang.Thread` in Java?

java multithreading

112

Share Edit Follow Close Flag

edited Jan 10 at 10:14
Jonas
101k ● 90 ● 275 ● 359

asked Mar 22 '09 at 14:04
flybywire
236k ● 184 ● 384 ● 491

2 till now you can not kill a thread; because destroy() is never implemented due to dead-lock prone – AZ_ Jun 10 '11 at 14:14

1 I prefer the answer regarding `ExecutorStatus` on this question: stackoverflow.com/questions/2275443/how-to-timeout-a-thread – Kirby Jun 17 '11 at 1:17

9 @loungerdork "I think Java should implement a safe stop/destroy method for runaway threads that you have no control over, despite the caveats of losing locks and other pitfalls" So you want an *unsafe* thread stop. I think you already have one. – DJClayworth Jun 10 '15 at 14:41

4 It's amazing what kind of questions would get 212 upvotes in 2009. This would be immediately destroyed today. – Jonathon Reinhart Mar 11 '16 at 19:34

7 @JonathonReinhart: Why is that? It seems to be a legit question, even these days. Maybe

The Overflow Blog

- Celebrating the Stack Exchange sites that turned 10 years old
- Podcast 367: Building a better developer platform

Featured on Meta

- Don't be that account: buying and selling reputation and bounties
- Outdated Answers: results from flagging exercise and next steps

Hot Meta Posts

- 32 Let's not [bounce] around
- 19 Is a small (but noteworthy) edit on several answers to one question acceptable?

Love this site?

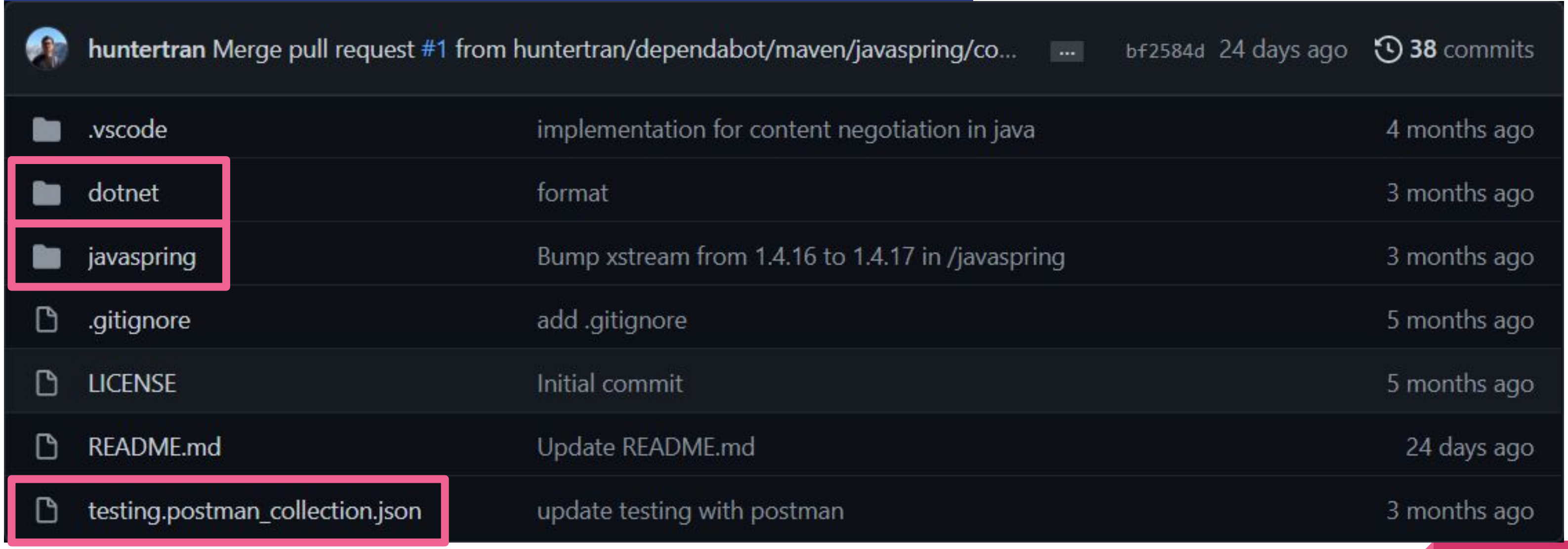
4 - REST API Anti-patterns

4 - REST API Anti-patterns - Overview

- Practice name
- Problem
- Expected result
- Solution

4 - REST API Anti-patterns - Sample Implementation

https://github.com/huntertran/restapi-practices-impl

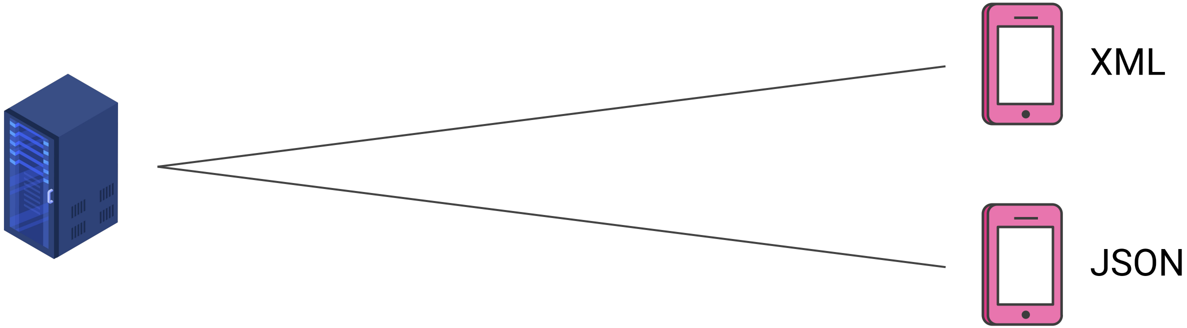


huntertran Merge pull request #1 from huntertran/dependabot/maven/javaspring/co... ... bf2584d 24 days ago 38 commits

.vscode	implementation for content negotiation in java	4 months ago
dotnet	format	3 months ago
javaspring	Bump xstream from 1.4.16 to 1.4.17 in /javaspring	3 months ago
.gitignore	add .gitignore	5 months ago
LICENSE	Initial commit	5 months ago
README.md	Update README.md	24 days ago
testing.postman_collection.json	update testing with postman	3 months ago

4 - REST API Anti-patterns - Content Negotiation

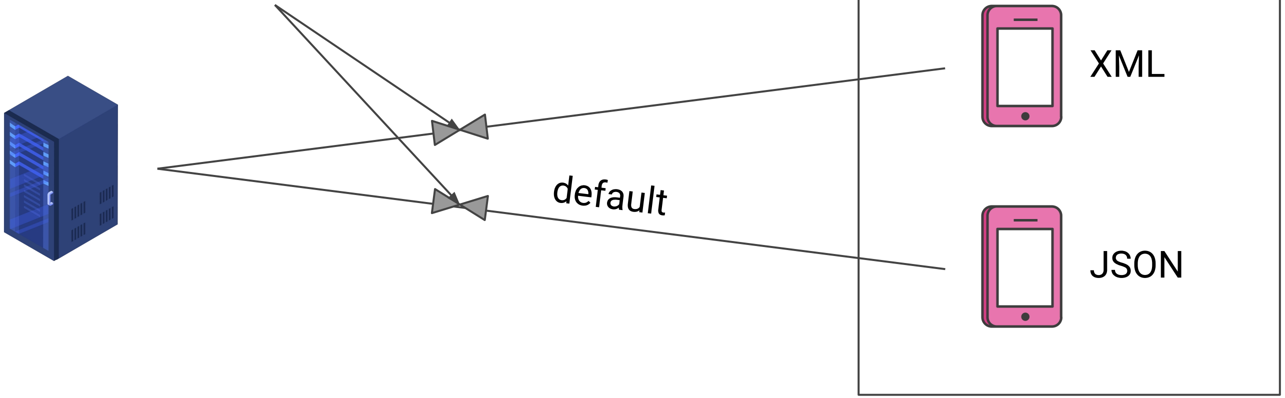
- Problem



4 - REST API Anti-patterns - Content Negotiation

- Expected result

Easily modifiable



default

XML

JSON

- JSON-XML
- JPG, PNG, Base64
- CSV, XLSX, ODS

1

- **JSON**-XML
- **JPG**, PNG, Base64
- **CSV**, XLSX, ODS

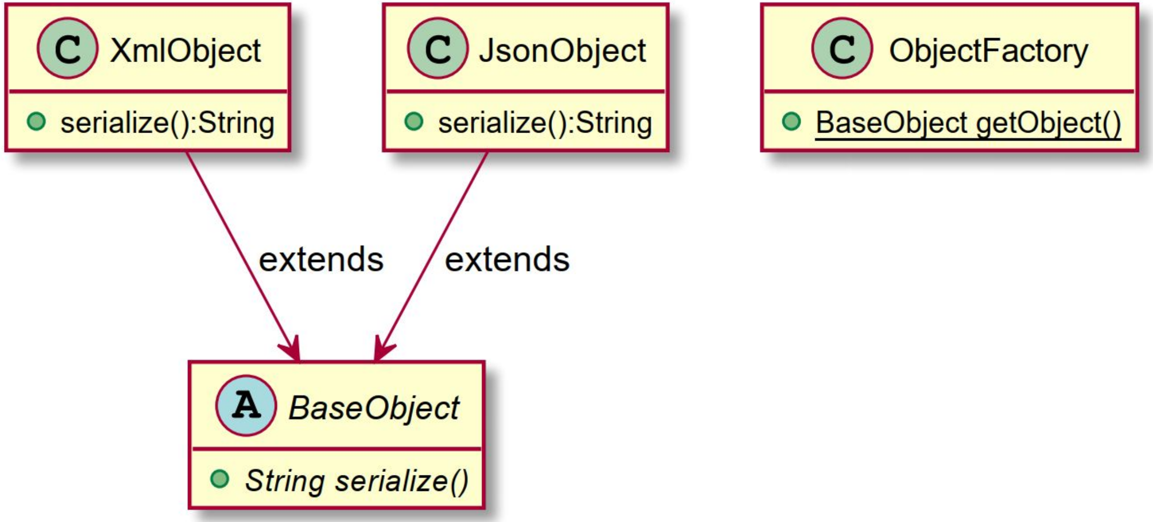
2

- **GSON/Jackson/Javax.Json**
- **Javax.ImageIO/ImageJ/imgscalr**
- **FileReader/XStream/JacksonXML**

3

4 - REST API Anti-patterns - Content Negotiation

- Solution



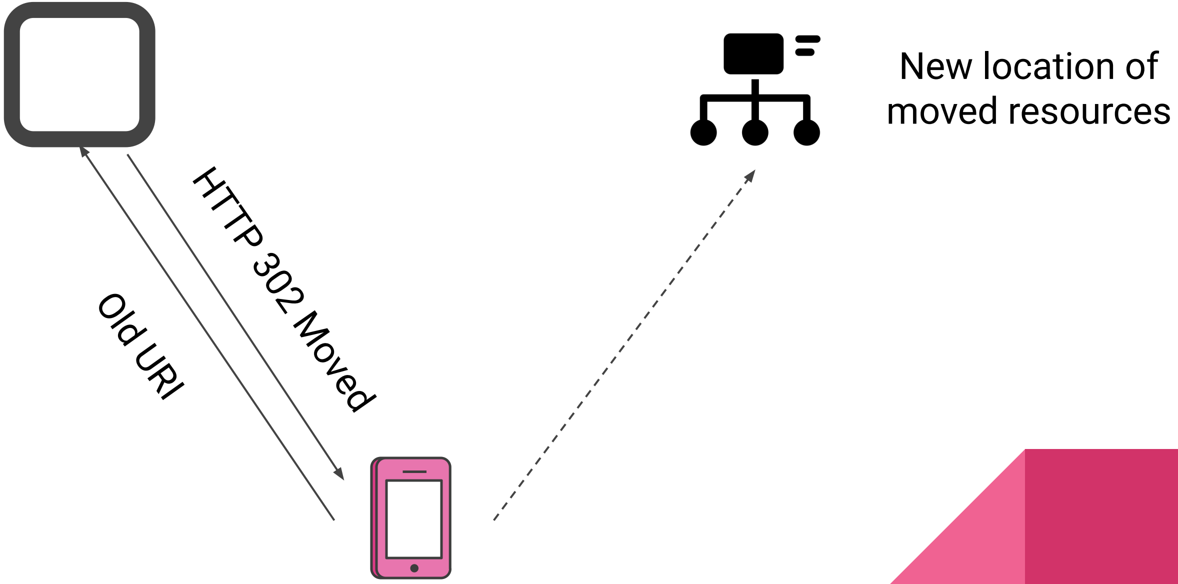
4 - REST API Anti-patterns - Content Negotiation

- Comparison

	Java Spring	**ASP.NET Core**	**Our solution**
Common media types	Yes	Yes	Yes
Customizable serializer	No	Yes	Yes
No data annotation on model	No	Yes	Yes
Built-in support ignorable	Yes	Yes	N/A

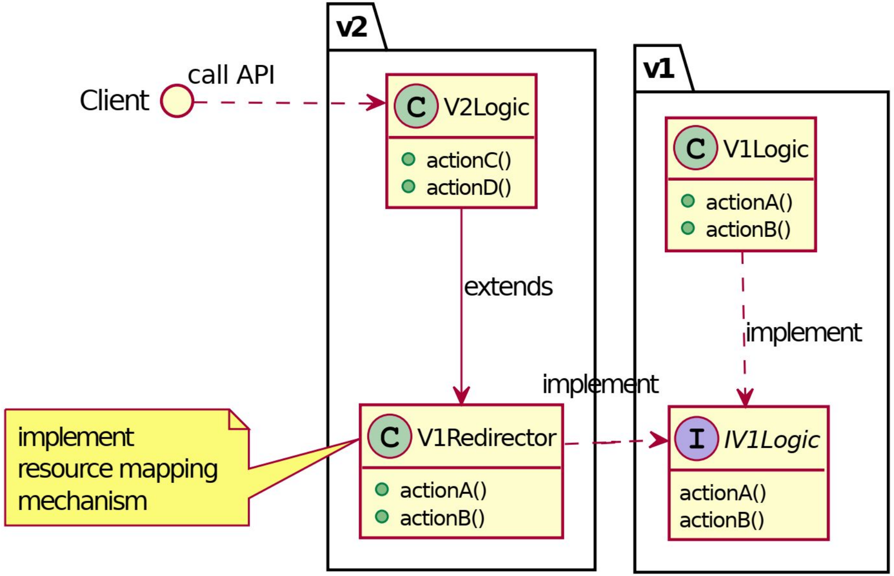
4 - REST API Anti-patterns - Endpoint Redirection

- Problem



4 - REST API Anti-patterns - Endpoint Redirection

- Solution



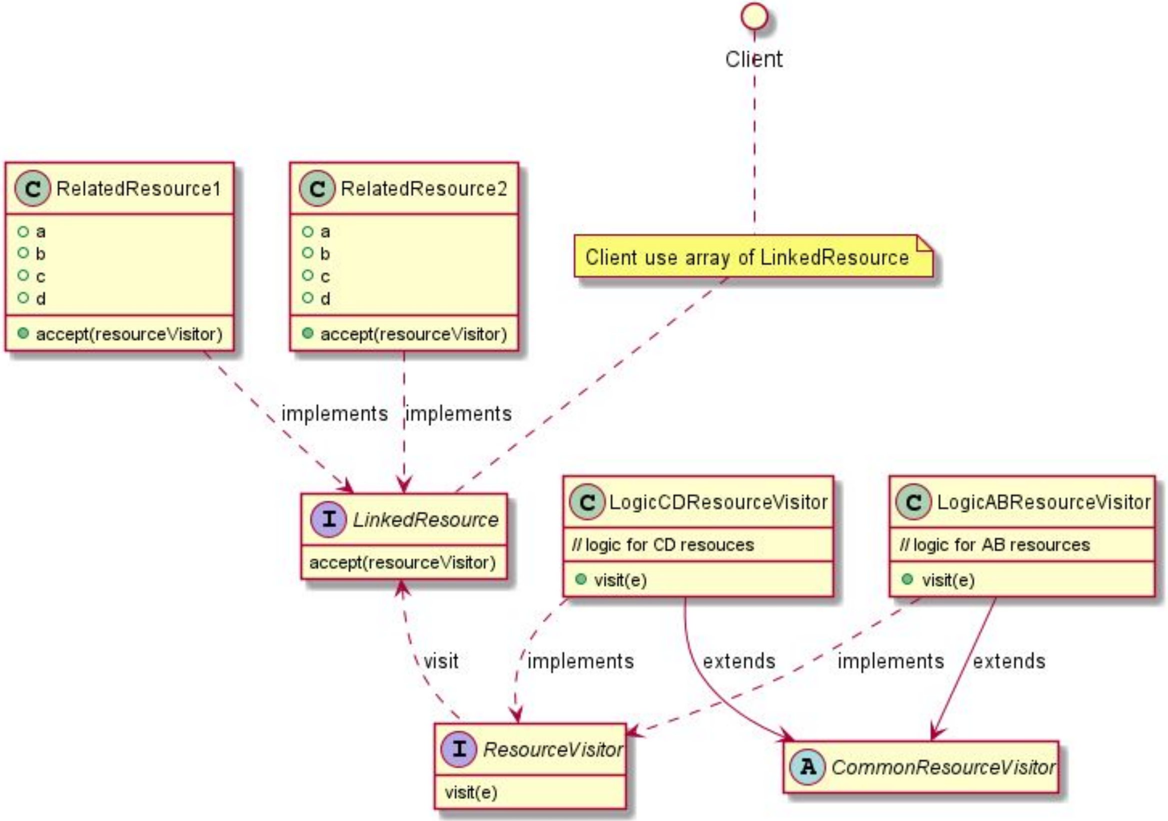
4 - REST API Anti-patterns - Entity Linking

- Problem

```
{
  "post": {
    "title": "Lorem ipsum",
    "content": "Lorem ipsum",
    "links": [
      {
        "rel": "comment",
        "method": "post",
        "uri": "/post/123/comment"
      },
      {
        "rel": "like",
        "method": "get",
        "uri": "/post/123/like"
      }
    ]
  }
}
```

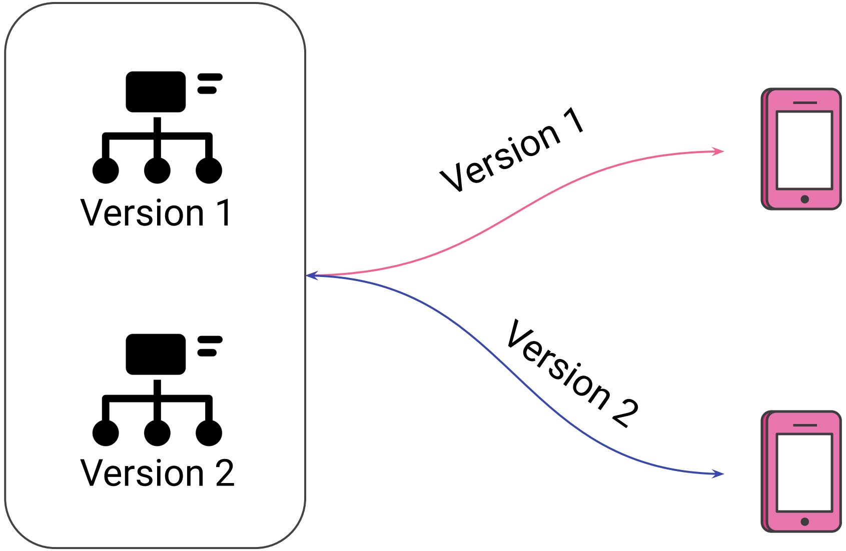
4 - REST API Anti-patterns - Entity Linking

- Solution



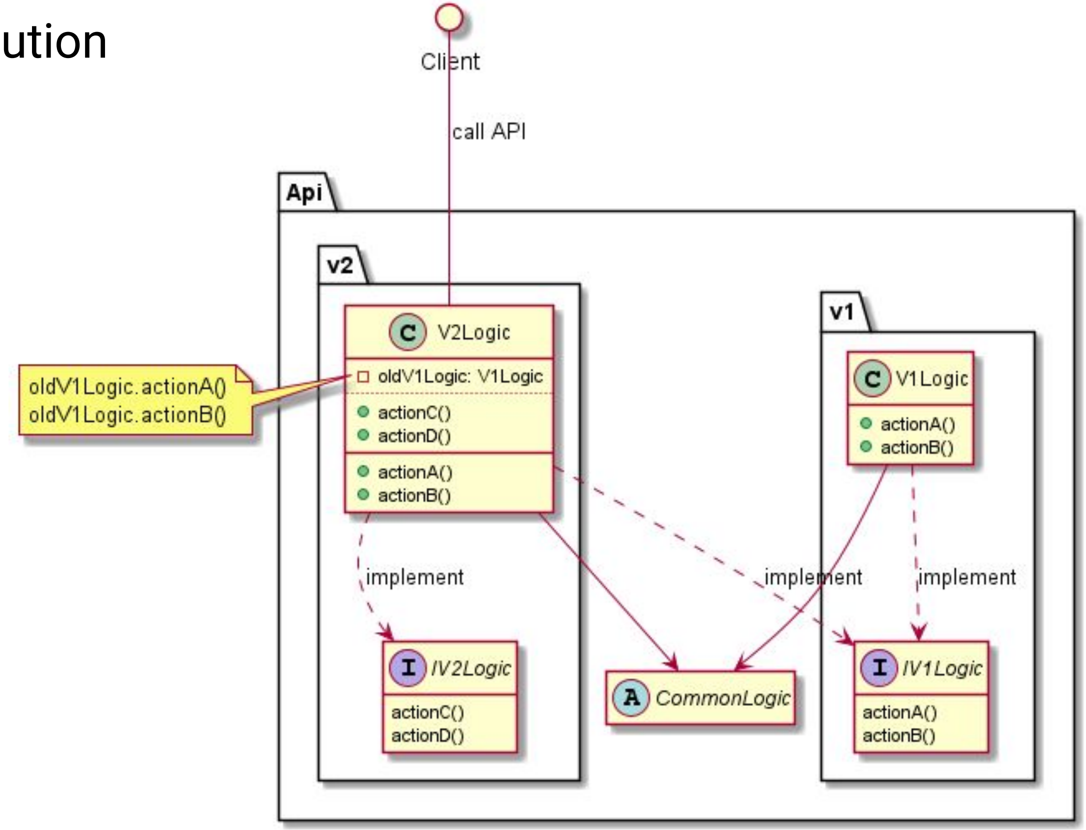
4 - REST API Anti-patterns - API Versioning

- Problem



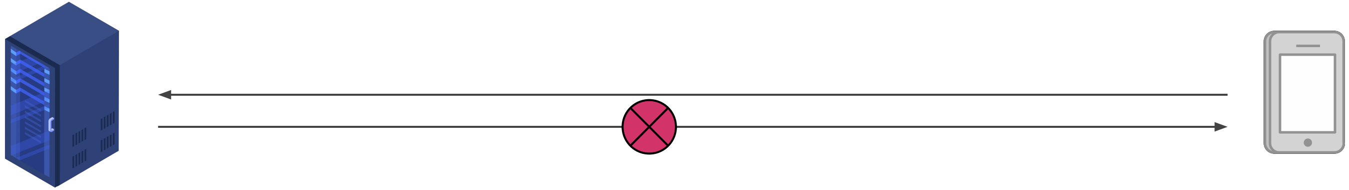
4 - REST API Anti-patterns - API Versioning

- Solution



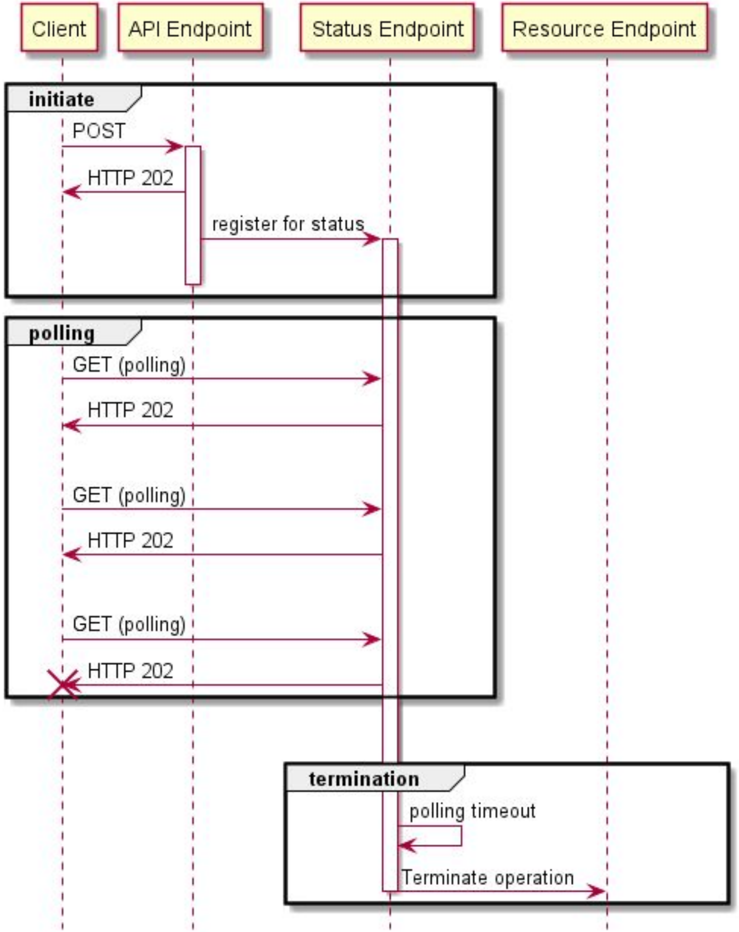
4 - REST API Anti-patterns - Server Timeout

- Problem



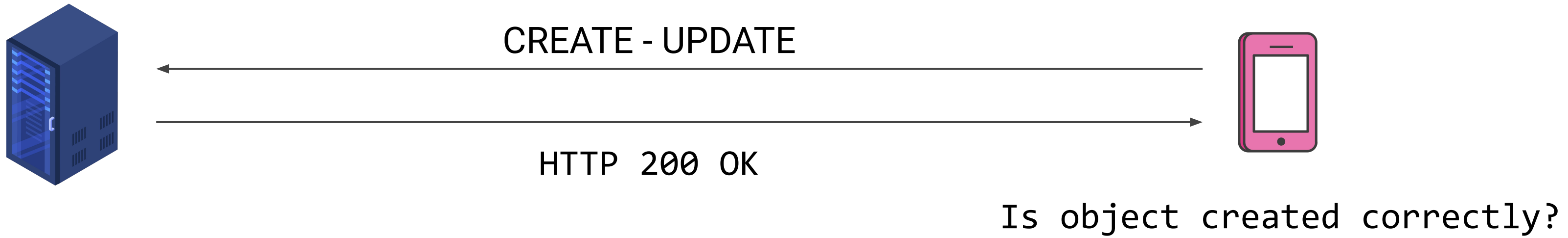
4 - REST API Anti-patterns - Server Timeout

- Solution



4 - REST API Anti-patterns - POST-PUT-PATCH return

- Problem



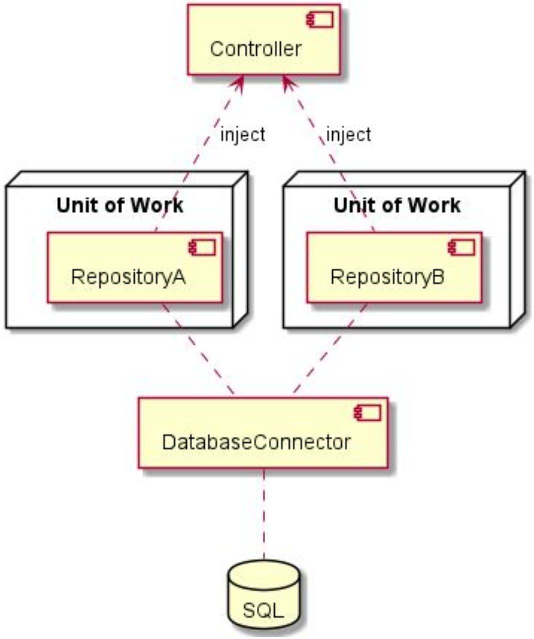
CREATE - UPDATE

HTTP 200 OK

Is object created correctly?

4 - REST API Anti-patterns - POST-PUT-PATCH return

- Solution



5 - Evaluations

5 - Evaluations - Survey Design

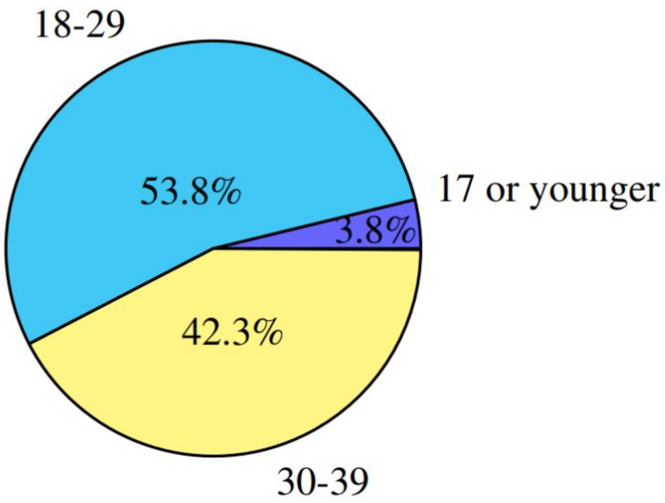
55 participants

- Have you faced this problem(s) in some of your project
- Is the proposed solution a good solution?

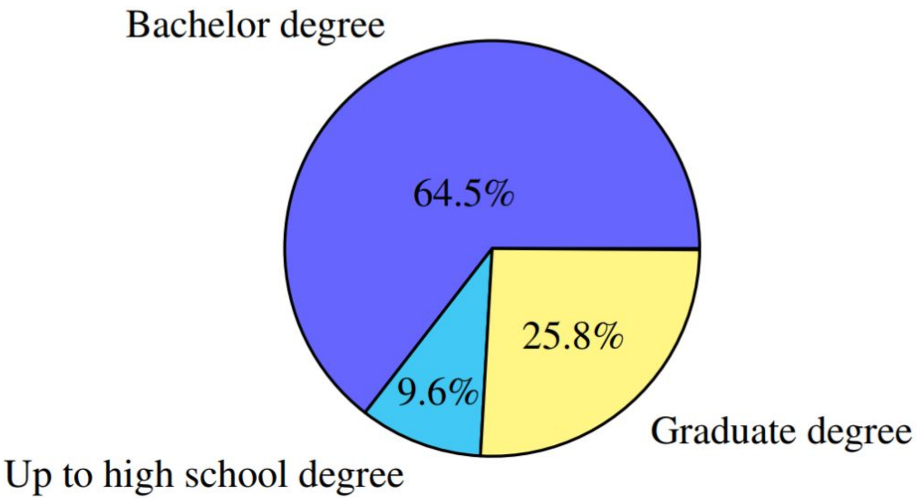
5 participants

ACM SIGSOFT
Empirical Standards
Version 0.1.0

5 - Evaluations - Survey Result - Age Groups & Education

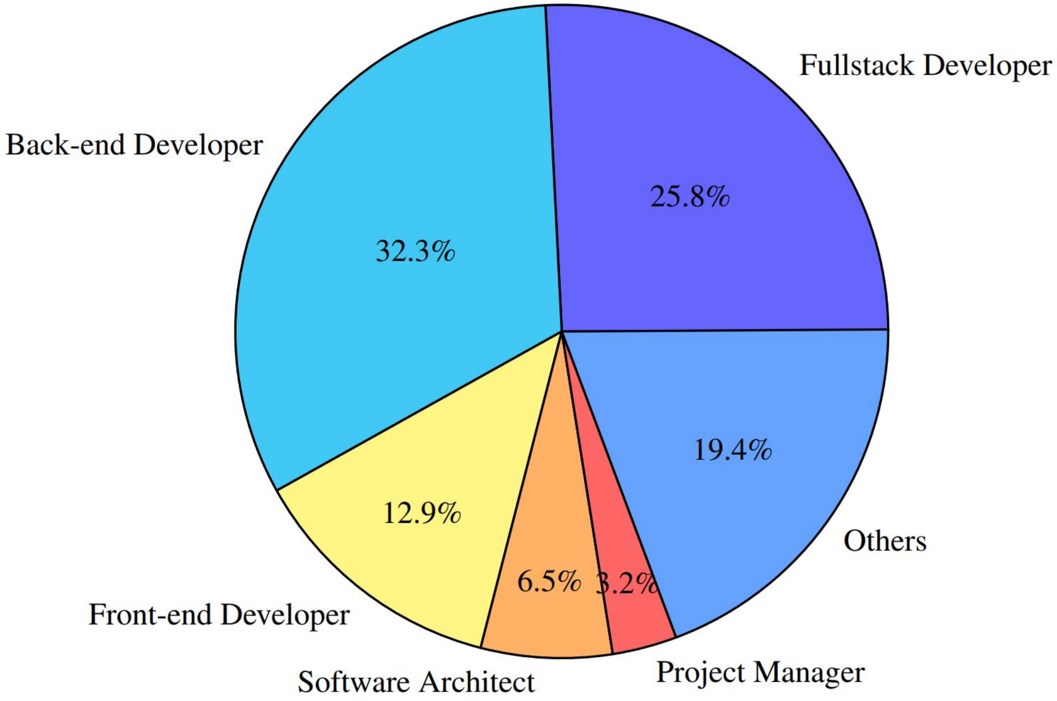


Age Groups

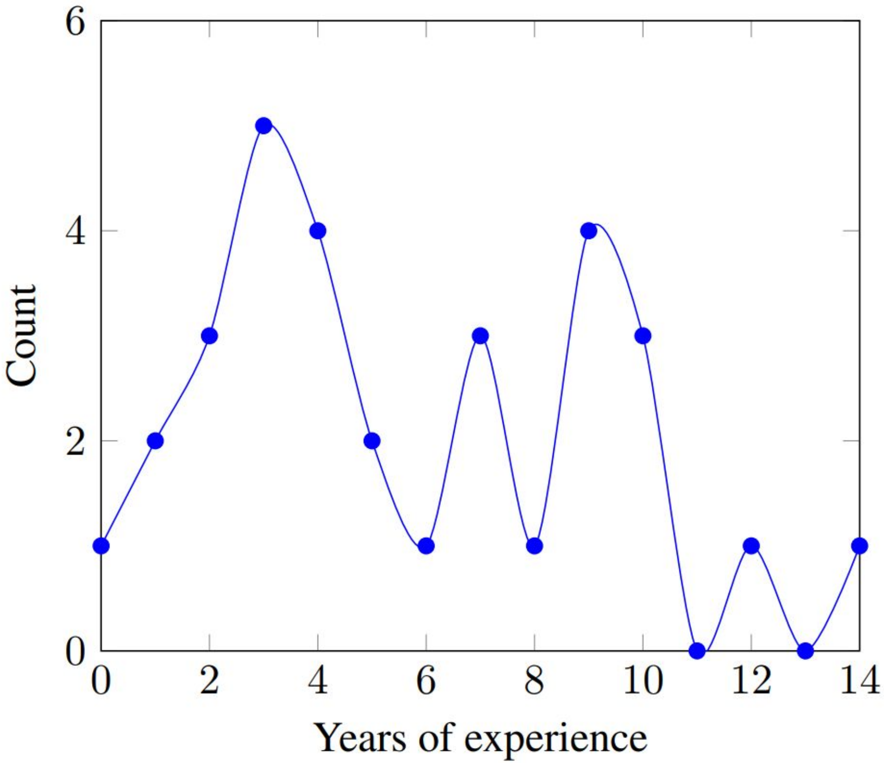


Education Levels

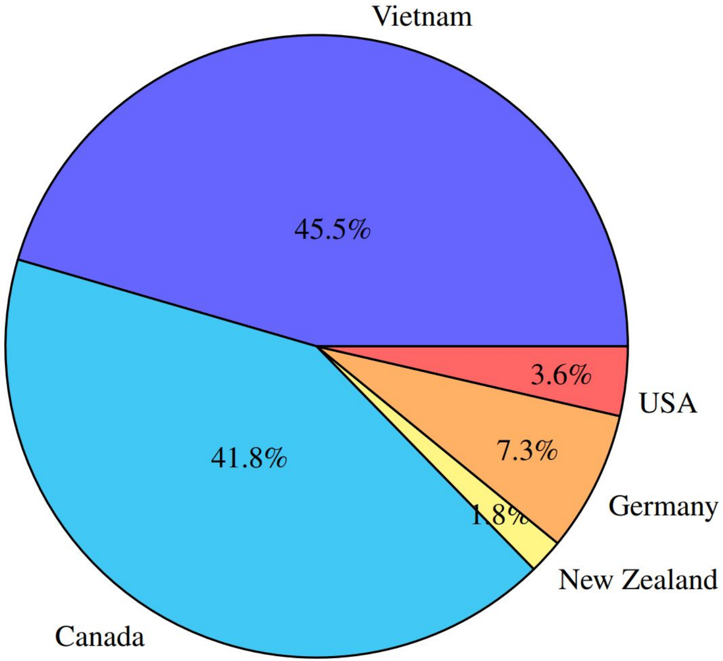
5 - Evaluations - Survey Result - Profession



5 - Evaluations - Survey Result - Experience

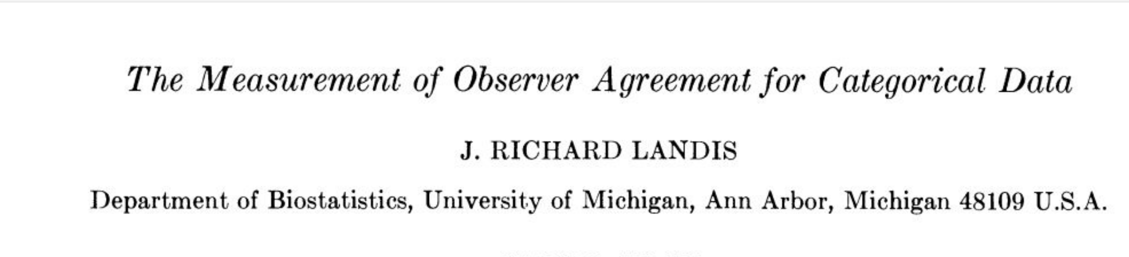


5 - Evaluations - Survey Result - Countries



5 - Evaluations - Survey Result - Positive on Solutions

	Face this problem	Std. dev.	Good solution	Std. dev.
Content Negotiation	52.40%	2.289	76.20%	1.952
Endpoint Redirection	45%	2.225	75%	1.936
Entity Linking	47.40%	2.177	57.90%	2.152
API Versioning	72.20%	1.901	72.20%	1.901
Server Timeout	77.80%	1.763	66.70%	1.999
POST-PUT-PATCH return	58.80%	2.029	82.40%	1.57



The Measurement of Observer Agreement for Categorical Data

J. RICHARD LANDIS

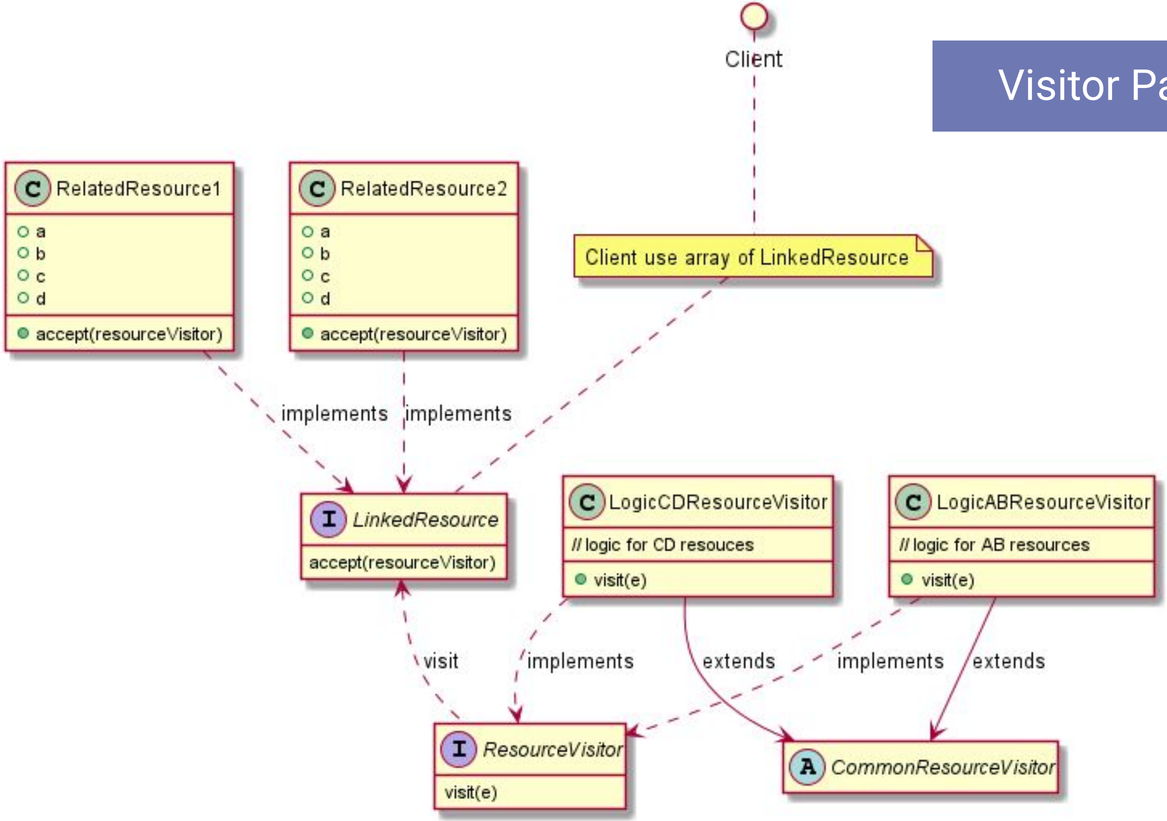
Department of Biostatistics, University of Michigan, Ann Arbor, Michigan 48109 U.S.A.

70%

5 - Evaluations - Survey Result - Explained

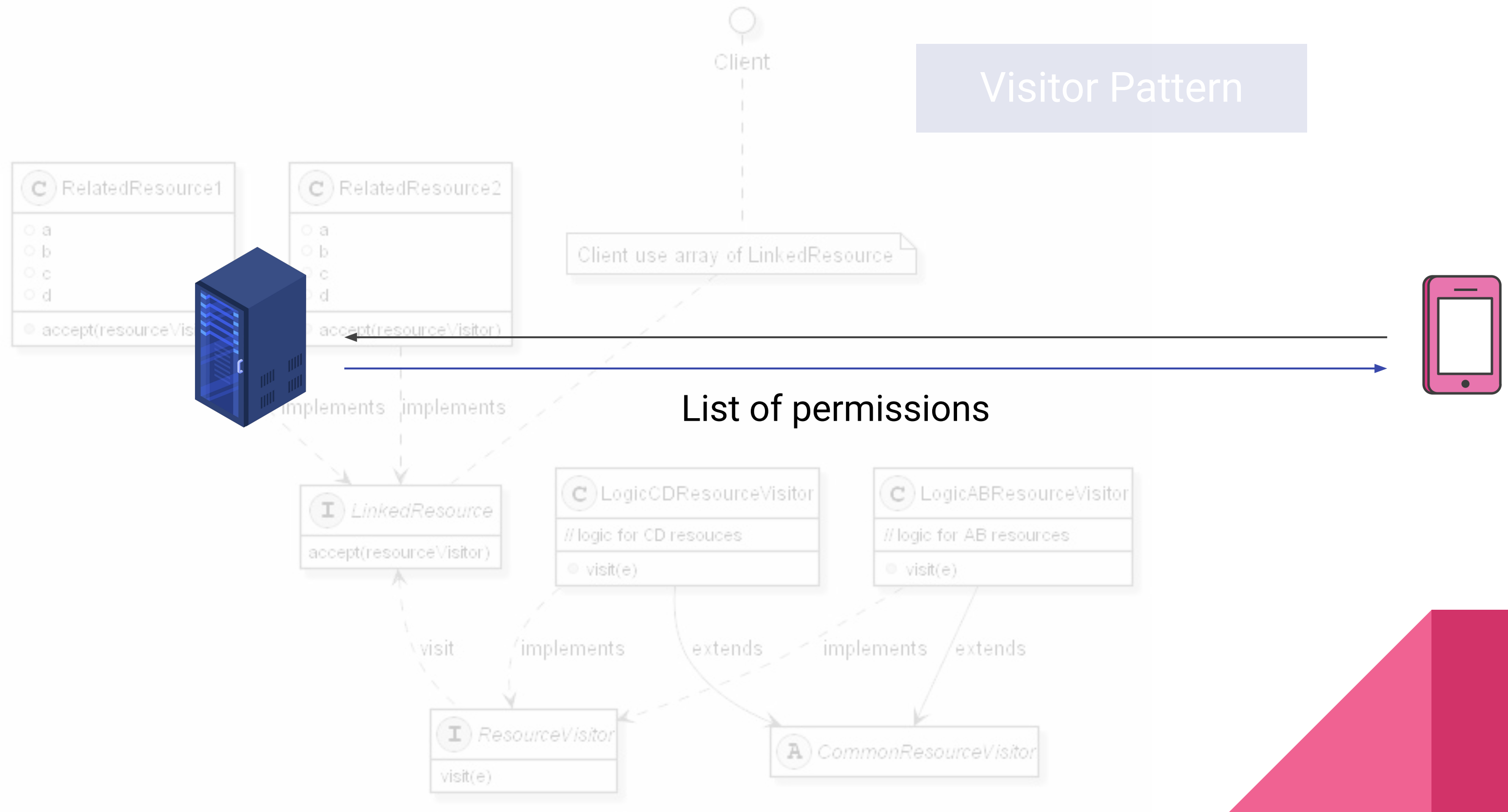
Entity Linking - 57.9% agreed good solution

Visitor Pattern



5 - Evaluations - Survey Result - Explained

Entity Linking - 57.9% agreed good solution

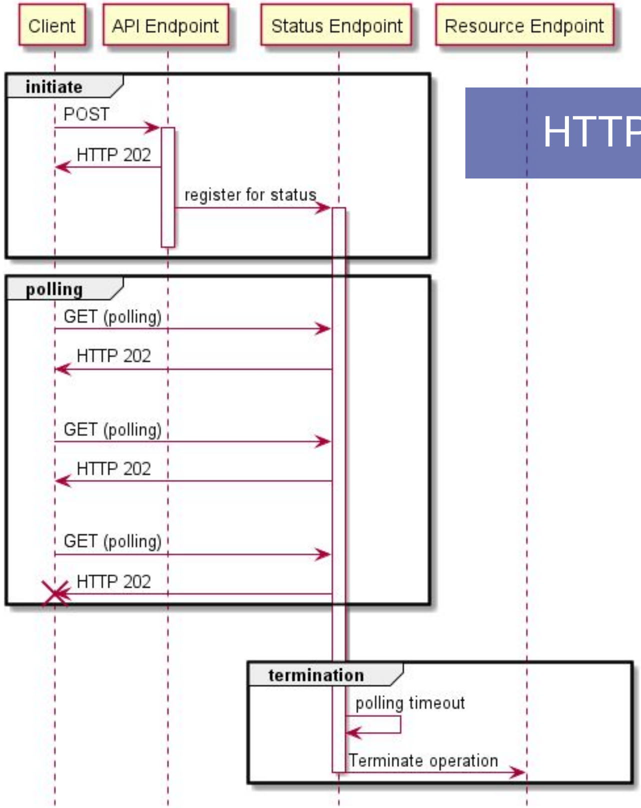


Visitor Pattern

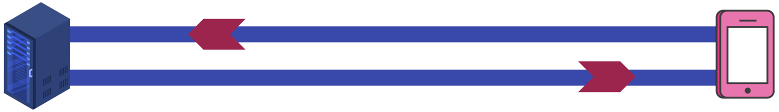
List of permissions

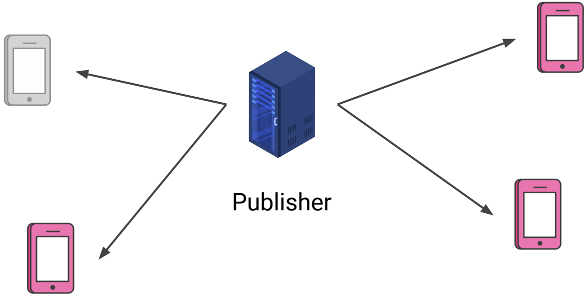
5 - Evaluations - Survey Result - Explained

Server Timeout - 66.7% agreed good solution



HTTP Polling



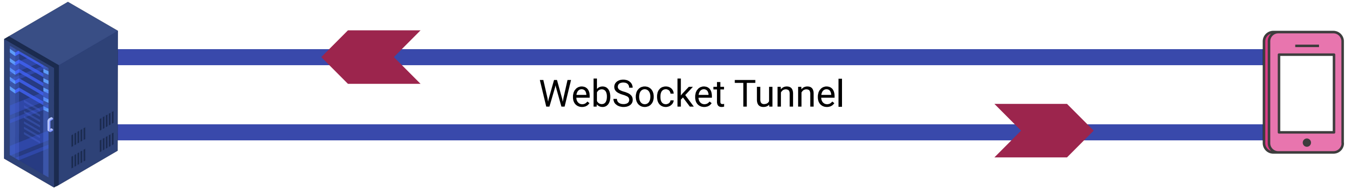


HTTP/2

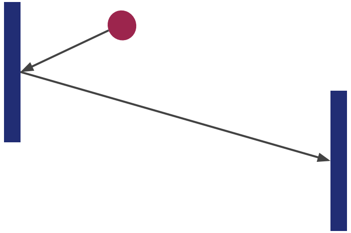
gRPC

5 - Evaluations - Survey Result - Explained

Server Timeout - 66.7% agreed good solution



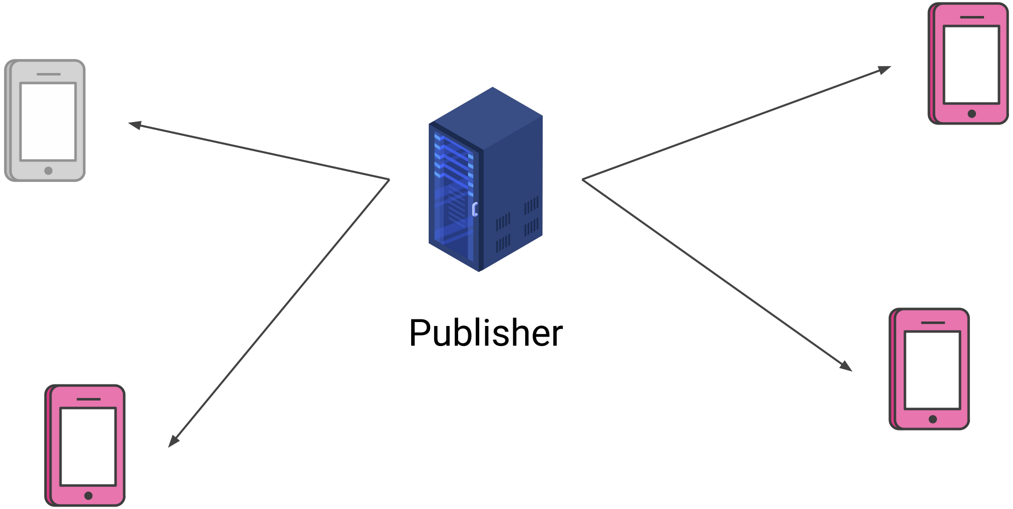
WebSocket Tunnel



HTTP Polling

5 - Evaluations - Survey Result - Explained

Server Timeout - 66.7% agreed good solution



5 - Evaluations - Survey Result - Explained

Server Timeout - 66.7% agreed good solution

HTTP/2

gRPC

Forced Encryption

TPC Head-of-line Blocking

https://engineering.salesforce.com/the-full-picture-on-http-2-and-hol-blocking-7f964b34d205

6 - Discussion

6 - Discussion - Internal Threats

JavaScript

Gin

Go

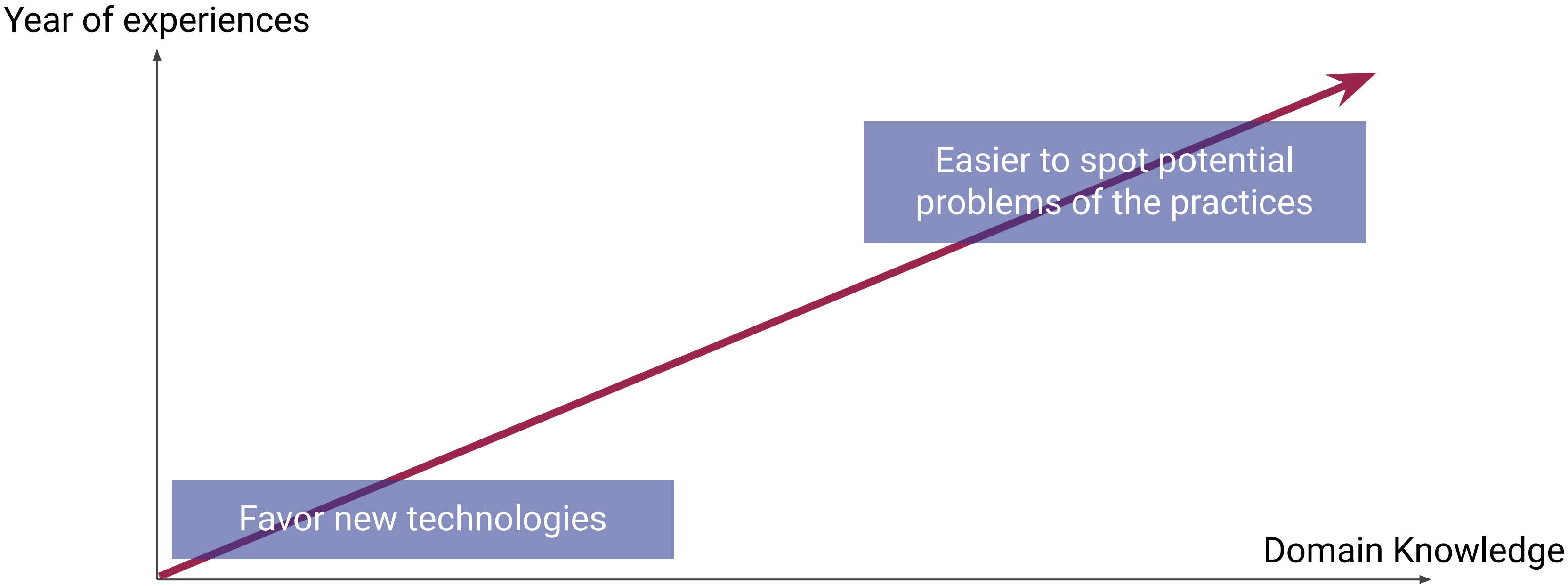
Grails

Groovy

Apache Struts

Java

6 - Discussion - External Threats



7 - Conclusion

7 - Conclusion

Our contributions:

1. Review REST API practices in both academic and gray literature.
2. Provide concrete solutions to 6 technical practices.
3. Validate the solutions with professional developers.

Thank you
Q & A

The 19th International Conference on Service Oriented Computing (ICSOC 2021)

Early submitted and get positive feedback in July
Resubmitted on August 22nd. Notification on Sept 20th.