Review: IEEE Internet Computing submission IC-2016-09-0119.R1

Permalink: https://gist.github.com/stain/a599413a3113912d9eb442141d6e418e
Date: 2017-12-20
Title: A Web API ecosystem through feature-based reuse
Authors: Ruben Verborgh, Michel Dumontier
Call: IEEE Internet Computing (regular article)
Review by: Stian Soiland-Reyes (3/3)
Recommendation: Accept If Certain Minor Revisions Are Made
Outcome: TODO

This review is licensed under a Creative Commons Attribution 4.0 International License.

Appropriate forum for publication

2. What is the most appropriate forum for the publication of this manuscript?

- IEEE Magazine (general interest explanatory article with technical contributions)

Section I. Overview

A. Reader Interest

1. How relevant is this manuscript to the readers of this periodical? Please explain your rating in the Detailed Comments section.

   - Very Relevant

B. Content

1. Please summarize what you view as the key point(s) of the manuscript and the importance of the content to the readers of this periodical.

   This article argues that while Web APIs have become prominent and many, they have also become unnecessarily fragmented, even when functionalities are similar; thus slowing down client development. The authors argue that although the server can change its internal implementation, current Web APIs are effectively implementation-specific for clients, because it is not possible to move from one service provider to another (e.g. posting updates to Facebook vs Twitter) without changing client code.

   The hypothesis is that reusable interfaces can improve the Web API client landscape. However, as even services providing similar functionality differ in other aspects, the proposed solution is to split services into Web API Features, where a particular provider's service is simply combining multiple features; some which may be maintained by communities and others which may be company-specific.
The article convinces the reader to change their thinking for making Web APIs and addresses the community implications. The article is however not a practical guide (although the principles are exemplified with existing technology), but presents more of a vision of where Web API development should go next.

2. Is the manuscript technically sound? Please explain your answer in the Detailed Comments section.
   - Yes

3. What do you see as this manuscript's contribution to the literature in this field?
   This article proposes Features-based design principles for Web API design. This can be seen to extend existing best practices, particularly the ideas expressed in the RESTful Web APIs (Richardson, Amundsen, Ruby; 2013); perhaps this book should be cited as relevant background.

4. What do you see as the strongest aspect of this manuscript?
   - Strong motivation
   - Well-explained idea
   - Principles expressed in a clear and structured fashion
   - Well discussed community aspects

5. What do you see as the weakest aspect of this manuscript?
   - Lacks discussion on need for common data content types, formats or identifiers
   - Mainly motivational ideas (but I think that makes it relevant for an IEEE Magazine)
   - Proposed approach seems to not have been tested in full (but this research angle is not as relevant for IEEE Magazine readers)
   - No discussion of why earlier interface-based approaches failed, e.g. WSDL re-using external XSD elements (but this is not relevant for IEEE Magazine readers)

C. Presentation

1. Does the manuscript contain title, abstract, and/or keywords?
   - Yes

2. Are the title, abstract, and keywords appropriate? Please elaborate in the Detailed Comments section.
   - Yes

3. Does the manuscript contain sufficient and appropriate references (maximum 15)? Please elaborate in the Detailed Comments section.
   - References are sufficient and appropriate

4. Does the introduction clearly state a valid thesis? Please explain your answer in the Detailed Comments section.
   - Yes

5. How would you rate the organization of the manuscript? Please elaborate in the Detailed Comments section.
   - Satisfactory

Focus

6. Is the manuscript focused? Please elaborate in the Detailed Comments section.
   - Satisfactory

Length

7. Is the length of the manuscript appropriate for the topic? Please elaborate in the Detailed Comments section.
   - Satisfactory

Readability
8. Please rate and comment on the readability of this manuscript in the Detailed Comments section.

- Easy to read

9. Please rate and comment on the timeliness and long term interest of this manuscript to IC readers in the Detailed Comments section. Select all that apply.

- Topic and content are of immediate and continuing interest to IC readers

**Evaluation**

*Please rate the manuscript. Explain your choice in the Detailed Comments section.*

- Good

**Recommendation**

- Accept If Certain Minor Revisions Are Made

**Confidential Comments to the Editor**

...

**Public Comments**

Hi, I am Stian Soiland-Reyes http://orcid.org/0000-0001-9842-9718 and believe in open reviews.

I would appreciate if you could contact soiland-reyes@manchester.ac.uk if you agree on me publishing this as an Open Review.

This review is licensed under a Creative Commons Attribution 4.0 International License http://creativecommons.org/licenses/by/4.0/ and is also available at the secret URL https://gist.github.com/stain/a599413a3113912d9eb442141d6e418e

My view is that this article presents an interesting motivation for changing how we design Web APIs. While the underlying ideas and technologies are not new (as the authors point out), what this article argues is needed are changes in best practices and community building.

I have some questions and thoughts, detailed below, some which I think merits minor revision of the text:

- Stronger emphasis on content types and common data structures
- Emphasize challenges in formalizing “a feature” description
- Avoid bottoms/top-up terminology (confusion with Software Engineering)
- Mention that challenges of Web API Feature re-use have already been visited in earlier technologies (SOAP WSDL, CORBA IDL) and that we need to learn from their lessons
- Suggest that while community effort is key there will probably still be fragmentation of feature ecosystems, e.g. due to technology stack differences
- Minor typo/grammar and reference issues (see below)

From this my conclusion is Accept If Certain Minor Revisions Are Made.

**Detailed review per section**

**Introduction**

The authors motivate well that while Web interface components for humans are conceptually reusable across providers (e.g. social media status update); but somehow this portability is not true in the Web API landscape.

**Issues with the current Web API Landscape**
I agree on the main argument that current Web APIs have no explicit interface definitions and that the effect for clients is tight coupling with a particular provider.

This section touches on previous developments with SOAP and WSDL. Perhaps an overlooked design feature there was the support to reuse XML Elements and WSDL service definitions across implementations, as WSDL has separated out endpoint binding information to be loaded at runtime.

However in practice this rarely happened, perhaps as code generation use transitioned from wsdl2code to code2wsdl on the server side (e.g. with Java annotations), forcing the monolithic service to be described in one go (always different); and where actual type and service reuse became hampered by different XML namespaces or differences in code generation. Here perhaps notable exception was the caBIG ecosystem which insisted on shared data types and service interfaces, at a large development cost to its community.

The reviewed article touches on the challenges on code generation.

I feel however that as an IEEE Internet Computing magazine article it might be out of scope to go into such deeper reflection of history - but still I think we have missed a lesson there. Perhaps the lack of focus on separate service features did hamper earlier development, and so I agree with the authors on trying to revisit the idea of interface-described service.

**Bottom-up instead of top-down**

This section rather confusingly exchanges top-down and bottom-up development, claiming that current Web API development is done top-down.

Using traditional Software Engineering terminology here I would argue this is opposite - Web API development now is only **bottom-up** - making only what the particular provider needs to expose (interfaces emerge from the required features as they are built); and what the authors are proposing is actually a partial return to traditional **top-down development** where the interfaces are broken down (Principle 1), defined in advance and reused (Principle 2) by several implementations.

Although the text uses perhaps opposite terminology, this section is still understandable, helped by Figure 1 which argues that the traditional monolithic APIs are “top down” with respects to its exposed functionalities and interfaces (binding the client), and the proposed “bottom up” alternative puts the features on top and the API emerges by the selection of implemented features.

Perhaps one suggestion to avoid the confusion against Software engineering terminology here would be to use other words, e.g. “API-centric” and “Feature-centric”.

**Feature-based interface principles**

This section provides in a clear fashion the proposed principles for Feature-based Web API design. The structure of Principle/Before/After/Tech/Examples works well and makes the text readable. This also suits well the magazine format.

**Principle 1**

This principle basically recommends defining a service by splitting it into features. I would say this is equivalent to Separation of Concern using modules/packages/namespaces in programming languages. Existing APIs might implicitly have such separation of concern; as indicated by their URI patterns or by documentation sections (see for instance https://developer.twitter.com/en/docs); but I agree with the authors this should be explicit. The provided examples (OpenSearch, AtomPub) are good and general.

**Principle 2**

My main question is from where a developer would find an existing “feature”. The article touches on this later with community building, but it should be clear that if many small features are made, it will become a large ecosystem; facing many of the same challenges previously encountered with reuse of XML Schemas, RDF vocabularies or OWL ontologies.

Yet it is not described anywhere what shape or format an independent feature definition is to take. If a feature is to be reused, it must exist in some form. Is it formal (e.g. Open API, JSON Schema, URI patterns), informal (HTML, PDF), or just a convention (REST principles)? Different forms of “feature descriptions” would mean different forms of reuse possibilities.

How can you validate if an API truly conforms to a reused feature, or just happens to look similar? (e.g. GitHub Repository Listing API implemented by GitLab)

Examples on p6 l50 seems to suggest Twitter already provide Atom collection of tweets, I was unable to find evidence for this. Modify to “instead of the current Twitter-specific API.”
This text shows little of the implication on how reusing a third-party feature could demand an extensive mapping to a provider's internal model, compared to provider-specific APIs which can “cheat” with a 1:1 correspondence to internal model (and just bump the API version when the internal model changes).

**Principle 3**

What's the role of HTTP Content Negotiation for advertising feature support, combined with re-use of hypermedia Content Types? This is one of the main idea of RESTful Web APIs which I don't see reflected on in this article.

In many ways I feel this article lacks attention to content types and common data formats. It would not be particularly interoperable if two services both implement a “search” feature and return JSON, but don't have the same JSON structure.

This article do not cover what is in scope for a feature description; I would suggest at least URI templates, authentication, content types, input/output data structures, hypermedia expectations, status codes, service compositions. But locking all of these down formally could also mean the feature becomes difficult to port across implementations (WSDL scenario).

**Principle 4**

This principle attempts to answer some of my earlier questions, by suggesting abstract interface definitions using technologies like Hydra and JSON-LD. Yet here the question is that there are so many technology choices; each which may be incomplete for describing a feature.

What are the implication of mixing features that are described in different ways? Say a JSON-LD-based hypermedia feature vs HAL feature. Using both features at once might mean an unpleasant mix of potentially incompatible client libraries, which for a client would mean same challenges as traditional cross-service integrations (e.g. service composition using workflows).

**Principle 5**

While I agree on the principle of measuring how and how much a feature is used, the key point here by the authors is that this must be measured across implementations. Yet no existing technology really exist that can help with this; I feel “benchmarking software” is misplaced.

What is really needed here is some kind of distributed “feature repository” ping-back functionality; which come with several privacy concerns. It would probably need to be an pen non-compete Community behind a given feature that needs the measurement.

There could also be softer indirect ways to measure this - for instance reusing a JSON-LD feature which import a community hosted @context means usage requests could be logged across implementations; but with several caveats such as caching, cloning or lack of usage details.

I also think that conformance should be measured - it's not sufficient for a provider just to claim they support e.g. the OpenSearch feature - they need to prove it through validation or conformance tests so clients can make reasonable assumptions across providers.

The latter part of this principle addresses the very important part of composing features (e.g. pagination and sorting). Here again a difficult question arises for API developers on how they should document what feature compositions are recommended or supported, particularly if features are only partially supported, or it is vague what outputs can fit as which inputs.

**Potential adoption obstacles**

This section has a strong discussion on the incentives and challenges of building feature communities.

**Conclusions and future directions**

This section gives a vision and implications of a feature-based Web API ecosystem. It highlights that this article gives the principles, but argues that mostly the technologies already exists, and that obstacles are not technological. I agree that the focus must be on communities.

What I think is the logical conclusion from this is that several communities could emerge - yet each community might agree on a different technology stack or different ways to express/formalize features.

In many ways this is already what is happening with standardizing effort, e.g. some academic dataset providers are supporting DCAT; others are doing OAI-PMH. They may support other "features" as well (e.g. Dublin Core), but disagree on serialization formats.
References

Citation 2: Missing author names: "SR Ponnekanti, A Fox" (https://doi.org/10.1007/978-3-540-30229-2_18)

All citations are missing DOIs/URLs -- I thought the journal was called Internet Computing.

Typos/grammar

- p3 l41: “as as”
- p8 l21: “we new possibilities”
- p8 l29: “The impact of a feature”
- Several locations: Unnecessary use of “...”, particularly when listing just 1 example.