Inter-parameter Constraints in Contemporary Web APIs

Nathalie Oostvogels*✉, Joeri De Koster, and Wolfgang De Meuter

Vrije Universiteit Brussel, Brussels, Belgium
{noostvog, jdekoste, wdmeuter}@vub.ac.be

Abstract. Today's web applications often rely on a myriad of external web APIs, communicating with them through various HTTP requests spread throughout the application. These APIs are often textually described by constraints on the inputs and outputs of their entry points. In this paper we discuss constraints in web APIs that span multiple parameters. We show that these constraints are common in web APIs, but cannot be expressed in existing machine-readable API specification languages. We envision the emergence of *constraint-centric* specification languages which focus on expressing constraints and describe a prototypical language that supports constraints over multiple parameters.

Keywords: Web application · Web API · API specifications · Inter-parameter constraints

1 Introduction

Today it is hard to imagine a web site without cross-website functionality such as a "like" button from Facebook, a video from YouTube or a Twitter feed. Such cross-website functionality is typically provided by a web service that exposes its functionality through an Application Programming Interface (API), which is primarily accessed by means of requests over HTTP(S).

A web API is comprised of a number of entry points (also referred to as endpoints, methods or routes) that are usually described in a publicly accessible textual documentation. For every entry point, it lists inputs (accepted parameters) and outputs (which are returned to the client). Additionally, the documentation often lists constraints on the input parameters of each entry point. Examples of such constraints are the type of a parameter, whether a parameter is required or optional, allowed values, etc.

Satisfying the constraints set by the API providers is essential for a request to succeed. Modern web applications contain many requests to many different APIs. Manually verifying each request in a web application is a difficult and time-consuming task. Fortunately, machine-readable API specifications aid the automatic verification of requests and generation of documentation.

* Funded by a PhD Fellowship of the Research Foundation - Flanders (FWO)

These specifications are written using machine-readable API specification languages [6], which describe the same information found in textual API documentations: general information (location and authentication requirements) and a list of every entry point of the API. The input and output data is described for every entry point, together with constraints for the input data.

As an API evolves, so do the constraints on parameters. This has led to the emergence of constraints over multiple parameters, which are currently not supported by existing specification languages. Table 1 shows an example of such an *inter-parameter* constraint in the Twitter API. The `direct_messages/new` entry point expects up to three parameters: `user_id`, `screen_name` and `text`. As for constraints, Twitter only indicates whether a parameter is required (`text`) or optional (`user_id`, `screen_name`). These *single-parameter* constraints are mostly well-documented and supported by existing machine-readable specification languages.

Table 1. Excerpt from Twitter API documentation[1]

Field name	**Optional?**	**Description**
`user_id`	optional	The ID of the user who should receive the direct message.
`screen_name`	optional	The screen name of the user who should receive the direct message.
`text`	required	The text of your direct message.
Note: One of `user_id` or `screen_name` are required.[2]		

However, as indicated below the table, either the screen name or the user ID should be provided as well. This is an example of a constraint that spans multiple parameters. In contrast to single-parameter constraints, inter-parameter constraints are not captured by existing machine-readable API specifications and can only be verified manually. This limits their usability for expressing more complex scenarios that are commonly found in today's web APIs.

In this paper, we reflect on inter-parameter constraints such as the one above, and we make three contributions:

1. We identify three categories of inter-parameter constraints, and show that they are commonly found in existing web APIs (Section 2);
2. We show where current machine-readable API specification languages fall short (Section 3);
3. We introduce a new *constraint-centric* API specification language, an extension of OpenAPI, that addresses these shortcomings (Section 4).

[1] https://dev.twitter.com/rest/reference/post/direct_messages/new [2] At the time of conducting our survey, the note below the table was explicitly mentioned in the API. Recently, the description has changed — omitting the note — but the constraint still holds.

2 Inter-parameter Constraints

Section 1 already showed an example of an inter-parameter constraint. In this section, we discuss several instances of inter-parameter constraints found in existing web APIs, and classify them into three categories: exclusive constraints, dependent constraints and group constraints. Inter-parameter constraints are as much part of an API as single-parameter constraints, but they are often not properly represented or enforced.

2.1 Exclusive Constraints

We call a constraint an *exclusive constraint* when **exactly one of a set of parameters is required**. Table 1 illustrates an exclusive constraint: Twitter expects either a `user_id` or a `screen_name` to indicate the recipient of the private message. In this case the textual representation does not truly reflect the correct requirements for the parameters of the request. Both parameters are tagged as optional in the API documentation, which is contradicted by the requirement that one of them must be supplied for the request to succeed.

Our study of existing API documentation (in Section 2.5) reveals that many of them contain exclusive constraints. Examples include entry points in

- Facebook for publishing a status update[6] where *"either link, place or message must be supplied"*;
- Stripe[3], where *"either source or customer is required"* when creating a charge;
- YouTube[4], where you may only provide one filter when retrieving a playlist (*"specify exactly one of the following parameters"*).

2.2 Dependent Constraints

The second category of inter-parameter constraints are *dependent* constraints, where **constraints on a parameter depend on a property of another parameter** (which we call the *base parameter*). This dependency can be on either the presence of a parameter or its value. There are three sub-categories of dependent constraints[5]:

- Present-Present (PP) dependent constraint: the presence of a parameter depends on the presence of the base parameter;
- Present-Value (PV) dependent constraint: the presence of a parameter depends on the value of the base parameter;
- Value-Value (VV) dependent constraint: the accepted set of values for a parameter depends on the value of the base parameter.

Table 2 shows an example of a PP-dependent constraint in the Facebook API. When posting a `link` on someone's wall, you can specify a `picture`, `name`, `caption` and `description` for that link. These four parameters may only be included when `link` (the base parameter) itself is also present.

[3] https://stripe.com/docs/api/node#create_charge [4] https://developers.google.com/youtube/v3/docs/playlists/list

[5] Value-Present dependent constraints are omitted, because no example was found in this survey.

Table 2. Dependent constraints in the Facebook API[6]

Field name	Optional?	Description
`link`	optional	The URL of a link to attach to the post. Additional fields associated with `link` are shown below.
`picture`	optional	Determines the preview image associated with the link.
`name`	optional	Overwrites the title of the link preview.
`caption`	optional	Overwrites the caption under the title in the link preview.
`description`	optional	Overwrites the description in the link preview

Our study of existing API documentation (in Section 2.5) also revealed dependent constraints in other web APIs. For example, in the "add a member to a list" entry point in the Twitter API[7], the parameters `owner_screen_name` or `owner_id` are only taken into account if the `slug` parameter is also present. This is actually a combination of two inter-parameter constraints: next to the PP-dependent constraint, there is also an exclusive constraint on the parameters `owner_id` and `owner_screen_name`. The Google Maps API has a PV-dependent constraint for rendering the directions[8]: the property `infoWindow` is ignored when `suppressInfoWindows` is true. Finally, the Amazon API for product advertisement contains an example of a VV-dependency: when searching for an item[9], `condition` cannot be set to "`new`" when the `availability` parameter is set to "`available`".

2.3 Group Constraints

We classify inter-parameter constraints as *group* constraints when **a set of parameters should be either all excluded or all included in the request**. Table 3 shows a group constraint found in the Twitter API: when creating a new tweet, the user's current location can be provided via the `lat` and `long` parameters. However, it is an error to pass along only `lat` *or* only `long`: both parameters must be included to specify the location of the resulting tweet.

In Section 2.5, we show that group constraints are found in many APIs. In Flickr[11], for example, all coordinates of a person in a picture (`x`, `y`, `width` and `height`) must be provided. The YouTube API[12] requires that when creating a playlist, the parameter `onBehalfOfContentOwnerChannel` may only be present when there is a value for the `onBehalfOfContentOwner` parameter. In addition, `onBehalfOfContentOwnerChannel` can only be used in conjunction with `onBehalfOfContentOwner`.

[6] https://developers.facebook.com/docs/graph-api/reference/v2.8/user/feed

[7] https://dev.twitter.com/rest/reference/post/lists/members/create

[8] https://developers.google.com/maps/documentation/javascript/reference#DirectionsRenderer

[9] http://docs.aws.amazon.com/AWSECommerceService/latest/DG/ItemSearch.html

[10] https://dev.twitter.com/rest/reference/post/statuses/update

[11] https://www.flickr.com/services/api/flickr.photos.people.add.html

[12] https://developers.google.com/youtube/v3/docs/playlists/insert

Table 3. A group constraint in the Twitter API[10]

Field name	Optional?	Description
`lat`	optional	The latitude of the location this Tweet refers to. This parameter will be ignored unless it is inside the range −90.0 to +90.0 (North is positive) inclusive. It will also be ignored if there isn't a corresponding `long` parameter.
`long`	optional	The longitude of the location this Tweet refers to. The valid ranges for longitude is −180.0 to +180.0 (East is positive) inclusive. This parameter will be ignored if outside that range, if it is not a number, or if there is not a corresponding `lat` parameter.

2.4 Identifying Unsatisfied Constraints in API Requests

When only a textual representation of an API documentation is available, an IDE is unable to automatically verify whether requests comply with a specification. Developers are then forced to rely on the API provider to respond with a meaningful error message in case of a malformed request, or are forced to manually verify each request in the application. The problem with the former is that this means that bugs can only be identified after deployment of the application. Additionally, this approach requires full coverage of every API request by the application's test suite. Furthermore, every API provider responds differently — and not always with an error message — to requests that do not satisfy its constraints. We can classify the responses to unsatisfied inter-parameter constraints in three categories:

The API provider returns an *error message*: in the best-case scenario the API provider returns a meaningful error message whenever inter-parameter constraints are not satisfied. Unfortunately, this is not always the case. For example, when the exclusive constraint from the YouTube API is not met by supplying more than one filter for a playlist, the following error message is returned: *"Incompatible parameters specified in the request"*. Twitter returns a more detailed error message when a dependent constraint is not satisfied: *"You must specify either a list ID or a slug and owner"*. For unsatisfied group constraints, Flickr returns as error message: *"Some co-ordinate parameters were blank"*.

The API provider makes a *silent choice*: API providers can opt to tolerate certain malformed requests in order to be compatible with a wider variety of clients. For example, Twitter does not complain when both the screen name and user ID are passed along when sending a direct message. However, when the screen name and the user ID belong to different users, Twitter chooses the screen name and silently ignores the user ID instead of raising an error. The same applies for group constraints present in the Twitter API: if not all group parameters are present, all incomplete groups are ignored. Similarly, Facebook just silently ignores all the dependent parameters when the *base* parameter is not

provided. These kinds of errors are very difficult to debug, because the developer does not receive any feedback about the incorrect requests.

The API documentation is incorrect: in the case of Facebook, where their API documentation mentions the exclusive constraint *"either link, place or message must be supplied"* for publishing a status update, supplying all parameters results in a sensible status update, where all provided values are combined.

2.5 Inter-parameter Constraints in the Wild

To investigate how frequently the three categories of inter-parameter constraints occur in web APIs in the wild, we analysed the six most popular APIs of ProgrammableWeb[13] (based on usage in mashups). Other catalogs and metrics exist, e.g. API Harmony[14] checks usage on GitHub. Table 4 summarises our results.

In every API documentation, we looked for keywords that indicate an inter-parameter constraint. Exclusive constraints are often indicated with *either* or *one of*, dependent constraints with keywords such as *additional* and *providing*, and keywords for group constraints include *corresponding* and *providing*.

Table 4. Inter-parameter constraints in web APIs

	Exclusive	Dependent	Group	# entry points
Google Maps JavaScript API	10	3	3	117
Twitter REST API	32	14	6	97
YouTube Data API	11	3	5	50
Flickr API	12	0	1	206
Facebook Graph API	11	4	1	209
Amazon Product Advertising API	2	5	2	9

Table 4 shows that every inter-parameter constraint occurs in every API, except for Flickr. Exclusive constraints are the most common inter-parameter constraint in web APIs with a total of 78 occurrences. Although less frequent, group constraints occur in all the APIs we investigated. Apart from Flickr, every API has dependent constraints in their API documentation.

In summary, inter-parameter constraints are present in modern web APIs and the way services respond to requests that do not satisfy constraints is not always well-defined. These diverse ways of responding to invalid requests stem from a disconnect between the documentation of an API and its implementation. Ideally, a specification is available that defines every constraint in a machine-readable manner, including inter-parameter constraints. This specification enables automatic verification of every constraint on the client side and reject invalid requests before they are sent. However, inter-parameter constraints are currently second-class concepts in specification languages and can therefore only be described textually.

[13] http://www.programmableweb.com/apis/directory [14] https://apiharmony-open.mybluemix.net/

3 Machine-Readable Specification Languages for Web APIs

Machine-readable specification languages for web APIs have been around since 2000, with the introduction of WSDL (Web Services Description Language)[2]. Since then, many new languages have emerged, such as WADL[4] (Web Application Description Language), OpenAPI specification (formerly known as Swagger), MSON (Markdown Syntax for Object Notation) and RAML (RESTful API Modeling Language). These languages primarily form the input for tools that generate human-readable documentation, but also enable automated testing of APIs and code analysis. If a machine-readable API is not available or cannot fully represent the constraints of an API, developers have to resort to manual verification of each API call.

Existing specification languages already support single-parameter constraints such as types, minimum and maximum values and whether parameters are required or optional. However, we argue that inter-parameter constraints are not supported by specification languages for web APIs. Table 5 shows four specification languages, but there exist many more specification languages for web APIs such as WSDL, WifL [3], Web IDL [1] and hRESTS [5]. However, to the best of our knowledge, none of these deal with inter-parameter constraints.

Table 5. Constraints in web API specifications

	Exclusive	**Dependent**	**Group**
OpenAPI	×	×	×
MSON	×	×	×
RAML	×	×	×
WADL	×	×	×
JSON Schema	×	✓	✓

We also include JSON Schema in our discussion — even though it only validates one object at a time — because it supports some inter-parameter constraints.[15] However, it is not a good fit for describing web APIs, as there are several mismatches between JSON Schema and web API specifications. First, JSON Schema by default allows fields that were not described in the schema. This is undesired behaviour while validating web APIs: the list of parameters should be exhaustive and extra parameters should be rejected. To ensure that unmentioned fields are rejected, the field `additionalProperties` must be added and set to `false` in *every* JSON Schema object.

Note that parameters in OpenAPI are described using a *subset* of JSON Schema, which excludes the features below.

JSON Schema does not directly support the specification of inter-parameter constraints, but programmers are able to express group constraints and Present-

[15] JSON Hyper-Schema is an extension of JSON Schema for describing APIs. However, it does not add additional expressiveness for describing web APIs.

Present dependent constraints. The following snippet shows an encoding in JSON Schema of the PP-dependent constraint specified in Table 2. In this constraint we use `dependencies` to specify that the preview picture and the title name can only be present if a link was also provided with the request.

```
{type: 'object', properties: {link:    {type: 'string'},
                              picture: {type: 'string'},
                              name:    {type: 'string'}},
                 dependencies: {picture: 'link', name: 'link'}}
```

The snippet below shows an encoding in JSON Schema of the group constraint specified in Table 3 as a mutual dependency between both fields.

```
{type: 'object', properties: {long: {type: 'number'}, lat:  {type: 'number'}},
                 dependencies: {long: 'lat', lat: 'long'}}
```

For humans creating or interpreting the specification, it can be difficult to see which dependency maps to which logical constraint. Likewise, it is difficult to combine multiple constraints on parameters in JSON Schema. Readability and maintainability could be improved by separating constraints from the structure of the object, eg. by having language constructs for defining custom constraints.

Finally, Table 5 shows that JSON Schema does not support exclusive constraints. Ostensibly, the *oneOf* construct appears suitable, but we show this is not the case with a counterexample in Listing 1.1. This example attempts to encode the constraint given in Section 2.1, where an object may only contain a `screen_name` or `user_id` field. Nonetheless, an object `{screen_name:42, user_id:42}` would be accepted as well: the `screen_name` parameter is not a string, therefore the first schema is not considered valid, and therefore the `oneOf` constraint passes! This is not a good fit for the exclusive constraints found in web APIs: we want to ensure that exactly one of the fields is present.

Listing 1.1. Attempt at using `oneOf` for exclusive constraints

```
1  {oneOf: [
2    {type: "object",
3     properties: {screen_name: {type: "string", required: true}}},
4    {type: "object",
5     properties: {user_id: {type: "number", required: true}}}]}
```

We can conclude that the current API specification languages have very minimal support for inter-parameter constraints. There is a need for a specification language that enables the specification of inter-parameter constraints (next to single-parameter constraints). Moreover, this language needs to be future-proof such that new kinds of inter-parameter constraints can be easily supported in the language as well. In the next section, we present a specification language that embeds support for inter-parameter constraints in its core.

4 OAS-IP: A Constraint-Centric Specification Language

In this section we introduce OAS-IP, a new specification language for web APIs, focused on defining and imposing constraints on parameters of entry points. By defining constraints using propositional logic, writers of API specifications can

factor out patterns in constraints and impose constraints on single or multiple parameters. This enables the discovery and implementation of novel inter-parameter constraints.

OAS-IP is an extension of the OpenAPI specification language, which aims to be a vendor-neutral specification language for web services, and is supported by many companies such as Google and Microsoft. OAS-IP offers two extensions to the specification language, both described below. The first extension enables developers to define predicates for common constraints, and the second introduces a new way to impose constraints on query and payload parameters. Contraints on path and header parameters are not supported, as they were not found in our survey.

4.1 Constraint Definitions

Figure 1 shows the syntax of constraints in OAS-IP: a constraint is a logical formula that consists of operations over parameters, joined together with logical connectives. As usual, precedence can be indicated by parentheses. Parameters target regular and nested fields, as well as "array" fields — constraints on which apply to every element of the targeted array. Operations test properties on these parameters, such as whether it is present, its type, and restrictions on its value or its length. Finally, to promote reusability, constraint definitions enable the abstraction of common constraint patterns.

$$
\begin{array}{rcl}
\mathsf{v} \in \text{Values} & ::= & \text{Number, String, Boolean or Parameter} \\
\mathsf{f} \in \text{Parameters} & ::= & \mathsf{s} \mid \mathsf{f.s} \mid \mathsf{f.[]} \\
\mathsf{t} \in \text{Types} & ::= & \mathsf{string} \mid \mathsf{number} \mid \mathsf{boolean} \mid \mathsf{object} \mid \mathsf{t[]} \mid \mathsf{null} \\
\mathsf{cd} \in \text{Constraint definitions} & ::= & \mathsf{s}(\mathsf{s}_1, ..., \mathsf{s}_n) = \mathsf{c} \\
\mathsf{c} \in \text{Constraint} & ::= & \mathsf{o} \mid \mathsf{lc} \mid \mathsf{s}(\mathsf{v}_1, ..., \mathsf{v}_n) \\
\mathsf{o} \in \text{Operations} & ::= & \mathsf{present(f)} \mid \mathsf{type(f)}{=}\mathsf{t} \mid \mathsf{length(f)} \oplus \mathsf{v} \mid \mathsf{value(f)} \oplus \mathsf{v} \\
\mathsf{lc} \in \text{Logical connectives} & ::= & \mathsf{and(c,\ c)} \mid \mathsf{or(c,\ c)} \mid \mathsf{not(c)} \mid \mathsf{implic(c,\ c)} \mid \mathsf{iff(c,\ c)} \\
\oplus \in \text{Math operators} & ::= & =,\ !=,\ <,\ >,\ <=,\ >=
\end{array}
$$

Fig. 1. Syntax definition for constraints

Listing 1.2 shows the definition of several single-parameter constraints (lines 2–5) as well as the three categories of inter-parameter constraints we identified in Section 2 (lines 6–10). Expressing the exclusive constraint requires that either `present(f1)` or `present(f2)` is true. Dependent parameters are expressed using an implication. Finally, group constraints are expressed with a double implication: `f1` must be present when `f2` is present, and vice versa.

4.2 Constraints

Listing 1.3 shows how (inter-parameter) constraints are imposed on entry points in OAS-IP. It imposes the exclusive constraint already discussed in Section 2.1.

Listing 1.2. Sample constraint definitions in the YAML syntax

```
x-constraint-definitions:
  - minimum(f, v)      := value(f) >= v
  - required(f)        := present(f)
  - string?(f)         := type(f) = string
  - number?(f)         := type(f) = number
  - xor(f1, f2) := or(and(present(f1), not(present(f2))), and(present(f2), not(present(f1))))
  - pp-dependent(f1, f2)         := implic(present(f1), present(f2))
  - pv-dependent(f1, f2, v)      := implic(present(f1), value(f2) = v)
  - vv-dependent(f1, f2, v1, v2) := implic(value(f1) = v1, value(f2) = v2)
  - group(f1, f2)      := iff(present(f1), present(f2))
```

Listing 1.3. Expressing constraints for an API operation in OAS-IP

```
/direct_messages/show:
   post:
      parameters:
        - { name: screen_name, in: query, type: string }
        - { name: user_id, in: query, type: string }
        - { name: text, in: query, type: string}
      x-constraints:
        - xor(screen_name, user_id)
```

In OpenAPI, and thus in OAS-IP as well, entry points are grouped under the `paths` key, with the different HTTP methods they support nested under the entry point. Lines 4–6 list the parameters for the POST method of this entry point, including single-parameter constraints to impose a type on the parameters. The inter-parameter constraint on these parameters is listed on line 8, indicating that exactly one of screen name and user ID must be supplied.

In OpenAPI, constraints are only specified per parameter, thus limiting it to supporting single-parameter constraints. Such constraints can be trivially translated to the *x-constraints* section, using the single-parameter constraint definitions in Listing 1.2. In Section 4.4 we discuss in detail how OAS-IP supports the constraints that can be expressed in OpenAPI.

4.3 Composing Inter-parameter Constraints

Section 2 showed that inter-parameter constraints are common in the documentation of web APIs. Sometimes these inter-parameter constraints are even nested. We will discuss the composability of inter-parameter constraints by means of an example from the Twitter API:

> *"You can identify a list by its* `slug` *instead of its* `list_id`*. If you decide to do so, note that you will also have to specify the list owner using the* `owner_id` *or* `owner_screen_name` *parameters."*

This sentence denotes a dependent constraint between `slug` and *two* fields (`owner_id` and `owner_screen_name`), which have an exclusive constraint imposed on them in turn. There is also an exclusive constraint between these three fields and the `list_id` field. Using the constraint definitions in Listing 1.2, we would like to write this down as:

```
and(xor(list_id, slug), iff(present(slug), xor(owner_screen_name, owner_id)))
```

However, this is subtly wrong: the constraint is also valid if every field except `slug` is present.[16] Instead, this constraint may be written down as a set of smaller, non-nested constraints:

```
xor(slug, list_id)
implic(present(slug), xor(owner_screen_name, owner_id))
pp-dependent(owner_screen_name, slug)
pp-dependent(owner_id, slug)
```

This set of constraints is not as concise, but it is correct. During the course of developing OAS-IP (and accompanying examples), we found it beneficial to work with sets of singular constraints rather than nesting constraints.

4.4 Comparison with Other Web API Specification Languages

Section 3 discussed several languages for web API specifications. Our main concern with existing languages is the lack of support for inter-parameter constraints: only JSON Schema has limited support for expressing constraints over multiple parameters. More specifically, JSON Schema expresses dependent constraints and group constraints using its `dependencies` construct.

This section introduced OAS-IP, a specification language for web APIs with constructs for defining constraints by means of predicates over parameters. Writers of web API specifications do not have a limited set of constraints to choose from: they can use any combination of the provided operations. Moreover, complicated predicate combinations are abstracted to a custom constraint definition. This is an advantage over JSON Schema, where lack of abstractions gives rise to readability and maintainability issues. For example: a group constraint results in two `dependencies` expressions, which is not always intuitive to the reader.

To gauge the expressiveness of OAS-IP for modeling existing web APIs, we examined the constraint keywords of OpenAPI and JSON Schema and attempted to replicate them with constraint definitions in OAS-IP. Apart from the type and whether the parameter is required, the majority of keywords are single-parameter constraints on either numeric values of parameters or the size of arrays or objects, and thus supported with the existing operations in OAS-IP. Others, such as `pattern`, `uniqueItems` and `multipleOf` can be supported by adding new operations. OAS-IP currently does not support the JSON Schema `items` and `additionalItems` constraints which provide a different schema for each item of an array.

A final difference is the handling of unspecified fields. The `patternProperties` keyword allows constraining parameters whose name matches a regular expression, while the `additionalProperties` keyword either validates or forbids unknown parameters. As we mentioned before, OAS-IP defaults to rejecting requests with unknown fields. Both can be supported with the addition of *patterned fields*: for example, `string?(metadata./^x-/)` would require that unspecified fields of the `metadata` object starting with `x-` must be strings.

[16] It is possible to come up with alternative formulations, but the reader needs to construct a truth table in order to convince him or herself.

With these extensions, OAS-IP can express the same single-parameter constraints as OpenAPI and JSON Schema. In addition, it is capable of describing inter-parameter constraints, which other specification languages cannot.

5 Conclusion and Future Work

Today, APIs of modern web applications are described in a public API documentation. This documentation is often generated using a machine-readable API specification language. Such specifications can also enable automatic verification of constraints in the API. Traditionally such specification languages have focused on *single-parameter* constraints.

We surveyed current API documentation and identified three kinds of *inter-parameter* constraints. These constraints are currently only described textually and are not covered by the existing specification languages. This limits the power of existing tooling around API specification languages. Our survey indicates that inter-parameter constraints are essential to achieving comprehensive, machine-readable web API specifications. Therefore, we designed OAS-IP, an API specification language based on the OpenAPI specification, extended with new language constructs to define and impose both single-parameter and inter-parameter constraints. With this added flexibility, developers can describe *all* constraints in their APIs, which in turn makes tooling more powerful.

As a proof of concept, we have developed a preprocessor which, given an OAS-IP specification, produces a list of constraints for every entry point in the API.[17] We envision that this preprocessor will form the basis for various tools that help developers interact with web APIs. As a first step, existing documentation generation tools can use the preprocessor to help web developers understand the constraints on particular entry points. Going further, we envision that constraint-centric languages will form the basis of new tools that statically analyse interactions with web APIs in web applications.

References

1. S. Bae, H. Cho, I. Lim, and S. Ryu. SAFEWAPI: Web API Misuse Detector for Web Applications. In *Foundations of Software Engineering*, pages 507–517, 2014.
2. E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, et al. Web services description language (WSDL) 1.1, 2001.
3. P. J. Danielsen and A. Jeffrey. Validation and interactivity of Web API documentation. In *International Conference on Web Services (ICWS)*, pages 523–530, 2013.
4. M. J. Hadley. Web application description language (WADL). 2006.
5. J. Kopecky, K. Gomadam, and T. Vitvar. hRESTS: An HTML microformat for describing RESTful web services. In *Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 1, pages 619–625, 2008.
6. R. Verborgh, A. Harth, M. Maleshkova, S. Stadtmüller, T. Steiner, M. Taheriyan, and R. Van de Walle. Survey of semantic description of REST APIs. In *REST: Advanced Research Topics and Practical Applications*, pages 69–89. Springer, 2014.

[17] https://github.com/noostvog/Verify-Request