# smartAPI: Towards a More Intelligent Network of Web APIs

Amrapali Zaveri[1,2(✉)], Shima Dastgheib[1], Chunlei Wu[3], Trish Whetzel[4], Ruben Verborgh[5], Paul Avillach[6], Gabor Korodi[6], Raymond Terryn[7], Kathleen Jagodnik[8,9,10], Pedro Assis[11], and Michel Dumontier[1,2]

[1] Stanford Center for Biomedical Informatics Research, Stanford University, Stanford, USA
[2] Institute of Data Science, Maastricht University, Maastricht, The Netherlands
amrapali.zaveri@maastrichtuniversity.nl
[3] The Scripps Research Institute, San Diego, CA, USA
[4] T2 Labs, Sunnyvale, CA, USA
[5] Imec – IDLab, Ghent University, Gent, Belgium
[6] Harvard Medical School, Boston, MA, USA
[7] University of Miami, Miller School of Medicine, Miami, FL, USA
[8] Icahn School of Medicine at Mount Sinai, New York, NY, USA
[9] NASA Glenn Research Center, Cleveland, OH, USA
[10] Baylor College of Medicine, Houston, TX, USA
[11] Department of Genetics, Stanford University, Stanford, USA

**Abstract.** Data science increasingly employs cloud-based Web application programming interfaces (APIs). However, automatically discovering and connecting suitable APIs for a given application is difficult due to the lack of explicit knowledge about the structure and datatypes of Web API inputs and outputs. To address this challenge, we conducted a survey to identify the metadata elements that are crucial to the description of Web APIs and subsequently developed the smartAPI metadata specification and associated tools to capture their domain-related and structural characteristics using the FAIR (Findable, Accessible, Interoperable, Reusable) principles. This paper presents the results of the survey, provides an overview of the smartAPI specification and a reference implementation, and discusses use cases of smartAPI. We show that annotating APIs with smartAPI metadata is straightforward through an extension of the existing Swagger editor. By facilitating the creation of such metadata, we increase the automated interoperability of Web APIs. This work is done as part of the NIH Commons Big Data to Knowledge (BD2K) API Interoperability Working Group.

**Keywords:** Web API · Web API description · Web services · Linked data · FAIR principles

## 1 API Interoperability

Workflows for data analysis are increasingly employing cloud-based, Web-friendly application programming interfaces (APIs). Thousands of Web tools


E. Blomqvist et al. (Eds.): ESWC 2017, Part II, LNCS 10250, pp. 154–169, 2017.
DOI: 10.1007/978-3-319-58451-5_11

and APIs are available through Web API registries such as ProgrammableWeb[1], BioCatalogue[2] (specifically for life science data), and cloud platforms such as Galaxy[3]. However, sifting through these and other API repositories to define a linkable toolset pertinent to the workflow is challenging. Discovering relevant APIs often requires a precise combination of matching keywords; and once discovered, the API outputs must be examined to determine whether or not they can be connected together. This task is made more difficult since, in general, there is a lack of rich metadata that precisely describe the APIs, their services, and the data on which they operate. Improvements to this task have been achieved slowly since authoring rich metadata is seen as tedious and unrewarding. Providing easy methods for Web API annotation that integrate with shared terminologies could ease this perception and foster a more discoverable environment for API repositories. In turn, users could more precisely find linkable services that meet their functional requirements as the number of APIs grows.

The problem of authoring coherent, comprehensive, and structured API metadata is gaining attention as a pressing matter due to the aforementioned demand and a lack of work in this area that has fully addressed the issues described. The API metadata problem requires an end-to-end solution – from the specification of metadata elements, to developing metadata templates, to filling out such templates using ontology-based terms, to offering developer-friendly solutions to augment API results. All of this needs to occur in a manner that facilitates discovery, exploration, and reuse. While this problem is admittedly large and complex, our objective here is to carve out specific elements and pilot a lightweight software system for the annotation, discovery, and reuse of what we call smart Web APIs. Our approach is innovative because we address first the problem of API metadata authoring, a task generally disliked in the field, by making it easier to generate useful metadata, and by demonstrating concrete benefits of semantic metadata to developers and users alike.

The overall aim of this project is to undertake a pilot effort that investigates the use of semantic technologies such as ontologies and Linked Data for the annotation, discovery, and reuse of APIs. Linked Data[4] involves the creation of typed links between data from different sources on the Web. It has the properties of being machine-readable, having a meaning that is explicitly defined, being linked to other datasets external to itself, and being able to be linked to from external datasets [4]. The Linked Data principles define the use of Web technologies to establish data-level links among diverse data sources. Linked Data is very useful in cases where exchanges of heterogeneous data are required between distributed systems [2]. Web services can similarly benefit from these principles to facilitate integration and composition [20]. The smartAPI specification employs Linked Data with the aim to connect diverse data sources in pursuit of improved API discovery, interoperability, and reuse.

[1] programmableweb.com.
[2] https://www.biocatalogue.org/.
[3] https://galaxyproject.org/.
[4] https://www.w3.org/DesignIssues/LinkedData.html.

Our main objective is to develop and evaluate a lightweight software system for the discovery and reuse of smart Web APIs. Smart Web APIs have the advantages that they (i) are easier to discover due to rich semantic annotations, (ii) can be readily connected together without additional data wrangling, and (iii) eliminate data silos by providing Linked Data. Our proposed system will consist of two key components: (a) a coordinated facility for the intelligent annotation of smart Web APIs; and (b) an application to discover smart APIs and how they connect to each other. Essentially, smartAPI helps make APIs FAIR [26]: Findable with the API metadata and the registry; Accessible with the detailed API operations metadata; Interoperable with the responseDataType metadata (profiler); and Reusable with the access to existing APIs stored in an open repository. This work is done as part of the NIH Commons Big Data to Knowledge (BD2K) API Interoperability Working Group (WG)[5] and is available at http://smart-api.info/.

This project has four main contributions:

- Development of the smartAPI metadata specification, based on the results of survey of API metadata guidelines and metadata-in-use in API repositories (Sect. 4).
- Development of an intelligent tool that supports the composition and validation of API metadata that conforms to the smartAPI specification (Sect. 5).
- Development of a profiler that automatically annotates the API response data with semantic identifiers (Sect. 5).
- Development of a repository and smartAPI-conformant API to submit, search, and browse API descriptions (Sect. 5) and obtain field-specific metadata suggestions.

We list the different use cases and projects, specifically in the biomedical domain, that are actively participating in the API Interoperability WG and in the process of annotating (or plan to annotate) their APIs using the smartAPI specification (Sect. 6). We then conclude with a discussion on the future direction of this work in Sect. 7.

## 2 Related Work

Currently, there exist several challenges in finding relevant APIs as well as reusing those APIs. We discuss both of these challenges in this section. Also, when discussing the annotation and description of Web APIs, we need to distinguish two main groups that interact with these APIs [24]. First, there are annotations targeted at *developers*, with the main aim of facilitating development. Second, there are efforts to describe Web APIs in such a way that *automated clients* can access and compose them. In this section, we will provide a brief overview of both kinds of annotations.

[5] https://bd2kccc.org/index.php/working-groups/?v=commons&h=front.

### 2.1 Challenges of Finding APIs

Finding relevant APIs is a challenging task for developers for diverse reasons. Extensive collections of useful and representative code and data are still lacking [10] despite the quick proliferation of APIs that makes the discovery of resources relevant to individual developers and users difficult [22]. The most visible and accessible APIs are often those that are currently most used, relegating newer and potentially more useful, but less popular, APIs to obscurity [22]. Application frameworks and software libraries often lack proper documentation [9,21], and more sophisticated algorithms need to be developed to facilitate the identification of useful resources [10]. The discovery of relevant APIs can be facilitated by enhancing rich metadata that describe APIs and the services and data associated with them. The smartAPI initiative contributes toward improved discoverability by providing methods that permit simple and intuitive annotation of Web APIs and that are integrated with standard ontologies.

### 2.2 Challenges of Reusing APIs

*Reuse* in the context of Web APIs can mean multiple things [24]. First, an API itself is a means to enable reuse of the functionality offered by a certain server. Second, the client-side code for interacting with an API can be (partially) reused across applications. Third, the interface of an API – independent of its implementation – can be (partially) reused by other servers, as is the case with standardized APIs. This third form of reuse is unfortunately rare, since many Web APIs are designed from scratch. The resulting heterogeneity leads to a steep learning curve for the integration of existing Web APIs in applications [10,24], which is the fourth and most common meaning of "Web API reuse". The smartAPI initiative aims to tackle this challenge by developing a profiler that features automatic annotation of API response data. This profiler is integrated with the smartAPI editor to facilitate the semantic annotation of APIs. These features enhance reusability as well as interoperability.

### 2.3 Annotations for Developers

The XML-based Web Service Description Language (WSDL) provided one of the first models to describe Web services [5,6]. However, WSDL only provides the mechanisms to characterize the technical implementation of Web services; it does not provide the means to capture the functionality of a service. Furthermore, the module source code is generated automatically using a WSDL description, which is then compiled into a larger program. Then, if the description changes, the program no longer works, even if such a change leaves the functionality intact. This prevents WSDL from being used for automatic service discovery at runtime. Furthermore, WSDL is limited by proprietary vendor-specific implementations, being bound to a specific programming language. Swagger[6], on the other hand,

[6] http://swagger.io/.

provides an editor for authoring HTTP API documents, and is widely used by API developers[7]. Swagger uses the OpenAPI specification[8], which defines a standard, language-agnostic interface to HTTP APIs. However, each API developer annotates his API in isolation, which results in less interoperable and reusable APIs. The current Web API landscape is hindered by the problem of scalability as every API requires its own hardcoded clients, which only benefits the developers. In particular on the current Web, there is a one-to-many relationship between Web APIs and clients: a single API often has clients for one or more programming languages, but none of these clients work with other APIs. As such, individual clients do not scale with the number of APIs. This makes each API unusually short-lived with a tightly coupled relationship of highly subjective quality. This directly leads to increase in development costs and prevents the design of a more intelligent generation of clients that provide cross-API compatibility [24]. Annotating APIs is an important step in making them accessible for more generic clients.

### 2.4 Descriptions for Automated Clients

Many approaches for service description exist with different underlying service models. OWL-S [18] and WSMO [16] are the most well-known Semantic Web Service description paradigms. They both allow the description of high-level semantics of services whose message format is WSDL [7]. Though extension to other message formats is possible, this is rarely seen in practice. Semantic Annotations for WSDL (SAWSDL [14]) aim to provide a more lightweight approach for bringing semantics to WSDL services. Composition of Semantic Web Services has been well documented, but all approaches focus on Remote Procedure Call (RPC) interactions and require specific software [19].

In recent years, several description formats for the more lightweight Web APIs have emerged [25]. Several methods aim to enhance existing technologies to deliver annotations of Web APIs. HTML for RESTful Services (hRESTS, [12]) is a microformats extension to annotate HTML descriptions of Web APIs in a machine-processable way. SA-REST [8] provides an extension of hRESTS that describes other facets such as data formats and programming language bindings. MicroWSMO [13,17], an extension to SAWSDL that enables the annotation of RESTful services, supports the discovery, composition, and invocation of Web APIs, but requires additional software.

The description of *hypermedia APIs* is a relatively new field. Hydra [15] is a vocabulary to support API descriptions, but does not directly support automated composition. RESTdesc [23] is a description format for hypermedia APIs that describes them in terms of resources and links. The Resource Linking Language (ReLL, [1]) features media types, resource types, and link types as priorities for description.

With our smartAPI specification, we build upon the already existing widely used OpenAPI specification to provide richer metadata that precisely describes

[7] 10M+ downloads according to http://swagger.io/, last accessed Dec 14, 2016.
[8] https://github.com/OAI/OpenAPI-Specification.

the APIs, their services, the data on which they operate, and the data they return. Our smartAPI editor, which is also an extension of the popular Swagger editor, makes it easier to generate useful metadata and indicates which terms are most widely used to annotate Web APIs. The editor also supports suggestion of metadata elements and values along with their usage frequency to the next API provider while she is annotating her API. Furthermore, the *smartAPI profiler* (c.f. Sect. 5), integrated within the editor, provides automatic annotation of the API response data. Finally, the smartAPI registry serves as a repository to save, search, and browse the created API descriptions. Consequently, the smartAPI framework helps to make APIs FAIR.

## 3 Survey of API Metadata in the Wild

We conducted a survey of existing metadata repositories and specifications that describe APIs. In particular, the following eight resources were surveyed:

- Repositories:
  - Biocatalogue [3][9], a registry of biological Web APIs with 1,184 entries.
  - Programmable Web[10], a directory of internet-based APIs with over 15,000 API descriptions.
  - Tools & Data Services Registry [11][11], a registry with information about analytical tools and data APIs for bioinformatics with 2,331 entries.
- Specifications:
  - OpenAPI Initiative[12], created by a consortium of forward-looking industry experts who recognize the immense value of standardizing how HTTP APIs are described.
  - Minimal Information About a Software (MIAS)[13], a key set of minimal fields can that provide maximum value when describing a software.
  - Prototype smartAPI Specification[14], a specification describing semantically annotated Web APIs that facilitates discovery and reuse of Web-based APIs.
  - Semantic Automated Discovery and Integration (SADI)[27][15], a set of design patterns defining the behavior of data retrieval and/or analysis resources that must interoperate on the Semantic Web.
  - schema.org API Reference[16], reference documentations for APIs as described by schema.org.

[9] https://www.biocatalogue.org, Accessed April 9, 2016.
[10] http://www.programmableweb.com/apis/directory, Accessed April 10, 2016.
[11] https://bio.tools/, Accessed April 9, 2016.
[12] https://www.openapis.org/, Accessed April 11, 2016.
[13] http://www.softwarediscoveryindex.org/, Accessed April 11, 2016.
[14] http://smart-api.info/website/docs/specification/, Accessed April 11, 2016.
[15] https://rawgit.com/wilkinsonlab/SADI-Specification/master/SADI-W3C-Member-Submission.html#service-metadata, Accessed April 12, 2016.
[16] https://schema.org/APIReference, Accessed April 12, 2016.

We retrieved and listed the metadata elements from each of the resources and also analyzed the degree to which each field was actually employed in practice by its frequency of usage. For instance, in the case of Programmable Web, which contains over 15,000 API descriptions[17], all of the entries use the Title and Description fields. However, only 90% of them supply details about the API provider and the primary category to which the API belongs. Results of the survey are available at https://goo.gl/F4OLnW. Thereafter, we aggregated all the metadata elements from the full set of eight resources to produce a common list of 54 API metadata elements (as discussed in Sect. 4).

## 4 SmartAPI Metadata Specification

This standard is the result of a survey conducted by the NIH Commons Big Data to Knowledge (BD2K) API Interoperability Working Group of existing metadata repositories and specifications that describe APIs. The smartAPI specification implements the FAIR principles: Findable, Accessible, Interoperable, and Reusable. In particular, we aggregated all the metadata elements from the eight surveyed resources to produce a common list of 54 API metadata elements. We subsequently divided these elements into five categories:

- API Metadata (Table 1[18]): 20 elements
- Service Provider Metadata (Table 2): 6 elements
- API Operation Metadata (Table 3): 12 elements
- Operation Parameter Metadata (Table 4): 10 elements
- Operation Response Metadata (Table 5): 6 elements

The smartAPI Specification includes 21 metadata elements beyond those included in the OpenAPI specification. Examples of the 21 elements are the category to which the API belongs; metadata format and access mode at the API metadata level; the parameter type and parameter value type at the operation parameter level; and the conformance to a specified response profile at the operation response level. The metadata elements marked with a * in the tables are those specific to the smartAPI specification.

Next, we re-evaluated each of the metadata fields according to its applicability and relevance, and further determined whether each MUST, SHOULD, or MAY be included in the API description. The cardinality and datatype of metadata fields were further specified along with a description and example [19]. The smartAPI Specification along with cardinality, datatype, and an example of each metadata element is available at https://websmartapi.github.io/smartapi_specification/.

[17] last accessed April 2016.

[18] Note: The tables only contain the elements which are MUST and SHOULD. All other elements can be found on the website https://websmartapi.github.io/smartapi_specification/.

[19] The keywords “MUST”, “SHOULD”, and “MAY” in this document are to be interpreted as described in RFC 2119 http://www.ietf.org/rfc/rfc2119.txt.

**Table 1.** smartAPI specification metadata elements: API metadata.

| Element | Description | Level | # | Type | Example |
|---|---|---|---|---|---|
| Name | A human-readable label for the API | MUST | 1..1 | URI | `MyGene.info API` |
| Access point | The base URI for interacting with the API | MUST | 1..1 | URI | http://mygene.info/ |
| Description | A human-readable description of the API functionality | SHOULD | 0..1 | string | `MyGene.info Gene Query Web APIs. Learn more at` http://mygene.info/. |
| Response MIME-type | A list of media types the APIs can produce. Can be overridden on specific API calls | SHOULD | 0..n | string | application/json |
| Documentation | Documentation page URL for the API | SHOULD | 0..n | URI | http://docs.mygene.info/en/ |
| Version | The version of the API | SHOULD | 1..1 | string | `3.0.0` |
| Terms of service | A document that describes the terms of use for the API | SHOULD | 0..1 | string | http://mygene.info/terms/ |
| Support* | Indication of whether SSL Support is present or absent | SHOULD | 0..1 | bool | `yes` |
| Authentication mode | Lists the required security schemes to execute this operation | SHOULD | 0..1 | URI | `none` |

**Table 2.** smartAPI specification metadata elements: Service Provider Metadata.

| Element | Description | Level | # | Type | Example |
|---|---|---|---|---|---|
| Responsible organization | The identifying name of the contact person-/organization | MUST | 1..1 | string | `The Scripps Research Institute` |
| Responsible developer | Name of the developer (User ID/Name) | MUST | 1..1 | URI | http://orcid.org/0000-0002-2629-6124 |
| Contact email | An e-mail address where the provider of the service may be contacted | MUST | 1..1 | email | `cwu@scripps.edu` |

**Table 3.** smartAPI specification metadata elements: API Operation Metadata.

| Element | Description | Level | # | Type | Example |
|---|---|---|---|---|---|
| Operation title* | Title of the operation. A unique identifier of the operation | MUST | 1..1 | string | `q` |
| Operation description | Description of the operation | MUST | 1..1 | string | `Query string. Examples: "CDK2", "NM_052827", "204639_at".` |
| Consumes | A list of MIME types the operation can consume | SHOULD | 0..n | string | `application/json` |
| HTTP method | The base path on which the API is served, which is relative to the host | SHOULD | 0...1 | string | `GET` |
| Authentication mode | Lists the required security schemes to execute this operation | SHOULD | 0..n | string | `none` |
| Transfer protocol | The transfer protocol of the API | SHOULD | 0..1 | string | `http` |

**Table 4.** smartAPI specification metadata elements: API Parameter Metadata.

| Element | Description | Level | # | Type | Example |
|---|---|---|---|---|---|
| Operation parameter name | Name of the operation parameter | MUST | 1..1 | string | `q` |
| Operation parameter description | Description of operation parameter | MUST | 1..1 | string | `multiple query terms separated by comma (also supports "+" or white space), but no wildcards, e.g., "q=1017,1018" or "q=CDK2+BTK"` |
| Location | Determines the location of the parameter | MUST | 1..1 | string | `query` |
| Parameter type* | Type of parameter | SHOULD | 0..n | string | `inputParameter` |
| Parameter value type* | Type of the value of the particular parameter | SHOULD | 0..n | string | `enterzgene` |

**Table 5.** smartAPI specification metadata elements: Operation Response Metadata.

| Element | Description | Level | # | Type | Example |
|---|---|---|---|---|---|
| Response format | A list of MIME types the APIs can consume | MUST | 1..n | string | `application/json` |
| Example response value | An example of the response value | SHOULD | 0..n | integer | `200` |

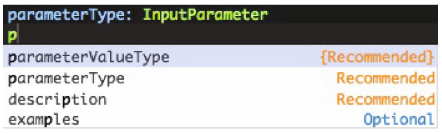
## 5 SmartAPI Implementation

smartAPI serves both the API providers and API users. The framework consists of three modules: the editor that facilitates the API metadata authoring for API providers; the searchable API registry where the created API documents are stored and indexed; and the profiler that annotates the API response data.

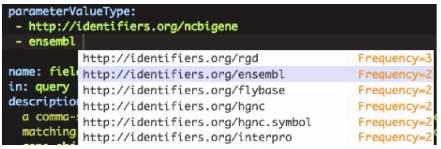
The smartAPI editor is an extension of the Swagger editor[20], which is widely used by API providers. The Swagger editor uses the OpenAPI specification and provides a framework for creating interactive HTTP API documentation. First, we extended the OpenAPI specification JSON file to incorporate the newly added smartAPI metadata. We extended the auto-completion functionality of the Swagger editor, by suggesting not only the list of predefined metadata and values, but also the values retrieved from the indexed API documents previously created and saved in the registry[21], along with the frequency of their usage. Every new API document added to the registry is indexed using Elasticsearch query engine[22], and their metadata elements and values along with their usage frequency are suggested to the next API provider (Fig. 1b). The conformance level (Required, Recommended, or Optional) of the suggested metadata element is also provided (Fig. 1a).

The smartAPI profiler, shown in Fig. 2a, provides automatic annotation of the API response data, i.e. responseDataType (Fig. 2b). To do this, the API response data (e.g. http://mygene.info/v3/gene/1017) is recursively traversed to provide a keypath/value pair where the keypath consists of one or more labels concatenated together and the value is either a single value or list of strings. The resource annotation is provided by comparing the keypath labels to resource names and synonyms from Identifiers.org[23]. In cases where a match is not found, an example value for the keypath is then compared against resource identifier patterns from Identifiers.org and resulting matches are displayed as suggested annotations. The user may also add his own resource annotation if one does not exist. The annotated API response data is stored in the responseDataType element (Fig. 2b).

[20] http://swagger.io/swagger-editor/.
[21] http://smart-api.info/registry/.
[22] https://www.elastic.co/products/elasticsearch.
[23] http://identifiers.org/.




(a) Metadata elements are suggested from the specification, based on the cursor position, along with their conformance level (Required, Recommended, Optional).




(b) Values for "parameterValueType" are suggested from both identifiers.org and previous APIs saved and indexed in the registry, and sorted by their usage frequency.

**Fig. 1.** Auto-suggestion functionality for API metadata elements and values.


(a) The API response is annotated using the profiler


(b) The annotated response data returned by the profiler is inserted to the editor.

**Fig. 2.** Semantic annotation of the API response (e.g. http://mygene.info/v3/gene/1017) using the smartAPI profiler.

The "parameterValueType" and "responseDataType" elements are added to the specification to semantically annotate the input (parameter) and the output (response) of the API respectively. As shown in Fig. 1b and Fig. 2b, the values of these metadata elements are semantic identifiers from identifiers.org, prefixcommons[24], and other relevant ontologies.

The code, full documentation, and tutorial are available at https://github.com/WebsmartAPI/swagger-editor. A live demo is also available [25].

## 6 SmartAPI Use Cases

One of the main use cases in which we will examine the usefulness and usability of the smartAPI system is to find and explore connections pertaining to cardiovascular pharmacogenomics. Our use case begins with a set of genes that are differentially expressed in hypertrophic cardiomyopathy (HCM), a leading cause of death among young athletes. HCM arises from genetic defects in close to 20 different genes, although the most common forms of HCM result from mutations in genes encoding proteins of the cardiac sarcomeric apparatus. One concern is that young athletes may be increasing their risk of HCM through

[24] http://prefixcommons.org/.
[25] https://www.youtube.com/watch?v=EQpUEiOu1ng&t=3s.

pharmacogenomic interactions. Our objective is to use the smartAPI platform to i) discover which, if any, differentially expressed genes in HCM are targeted by FDA-approved drugs, and ii) identify which HCM genes are also differentially expressed in other published cardiovascular studies. Information about drug targets and pharmacogenomics is already available as Linked Data, through the open source Bio2RDF project[26]. Bio2RDF provides nearly 11 billion Linked Data points from 35 life science databases including DrugBank[27] (a source of drug targets) and PharmGKB[28] (a source of pharmacogenomic interactions). Users of the smartAPI system can gain access to Bio2RDF data by following the Linked Data generated by the MyGene.info and MyVariant.info smartAPIs to Identifiers.org, which in turn will provide links to these Bio2RDF data.

The API Interoperability group is a Working Group[29] in the NIH Commons Framework project. The *NIH Commons* is defined as "an initiative which is essentially a shared virtual space where scientists can work with the digital objects of biomedical research, i.e., it is a system that will allow investigators to find, manage, share, use and reuse data, software, metadata and workflows."[30]. A series of Commons *pilots* has been initiated to develop and test these components in order to understand and evaluate how well they will contribute to an ecosystem that will effectively support and facilitate sharing and reuse of digital objects.

Below, we list the projects that are actively participating in the WG and are in the process of annotating (or plan to annotate) their APIs, specifically in the biomedical domain, using the smartAPI specification:

- **MyGene.info** [28][31] provides Web APIs for both gene queries and gene annotation retrieval. MyGene.info services are being used in Web applications that require querying genes, e.g. BioGPS[32], as well as in an analysis pipeline to retrieve regularly updated gene annotations. MyGene.info has a Swagger-based API document that was loaded into the smartAPI Swagger editor for being validated against the smartAPI specification and saved into the smartAPI registry[33]. The validation process provided a list of missing required, recommended, and optional metadata elements. As a result,"contact" info was added as a required element and the "parameterType", "parameterValueType", and "responseDataType" were recommended. These additions semantically enrich the API document and increase its interoperability with other relevant APIs.
- **MyVariant.info** [28][34] provides simple-to-use Web APIs to query/retrieve variant annotation data, aggregated from many popular data resources.

[26] bio2rdf.org.
[27] http://download.bio2rdf.org/release/3/drugbank/drugbank.html.
[28] http://download.bio2rdf.org/release/3/pharmgkb/pharmgkb.html.
[29] https://bd2kccc.org/index.php/working-groups/?v=commons&h=front.
[30] https://datascience.nih.gov/commons.
[31] http://mygene.info/.
[32] http://biogps.org.
[33] http://smart-api.info/registry/.
[34] http://myvariant.info/.

MyVariant.info was modified and saved into the smartAPI registry through the same process as MyGene.info.

- The **National Institutes of Health Library of Integrated Network Cellular Signatures (NIH LINCS) Data Portal**[35] provides access to a diverse array of novel bioassay data that has been curated and packaged with rich metadata for the assay entities. These metadata conform to the NIH LINCS metadata standards[36] that enable integration and interpretation of LINCS data. The LINCS Data Portal API[37] provides programmatic access to all datasets, dataset entities, and metadata within the LINCS Data Portal.
- The **BD2K PIC-SURE HTTP API** facilitates platform-agnostic programmatic access to disparate patient-level heterogeneous datasets to authenticated users. The API provides a selection of methods to access, query, and interrogate data in diverse formats[38]. To test the PIC-SURE API, a demo with the National Health and Nutritional Examination Survey (NHANES) is available online[39]. NHANES is a publicly available epidemiological survey conducted by the US CDC, recording over 1,100 variables from more than 41,000 respondents across the US; it is essentially a snapshot of patients' exposomes and phenomes. The exposome is composed of collections of environmental, behavioral, and dietary factors that are associated with health and disease, and phenomes include clinical and physiological phenotypes that are predictive of health.
- The **Alliance of Genome Resources (AGR)**[40] is an initiative formed in 2016 that has the goals of providing better support for the biological sciences via an integration of shared data; standardization of data models and interfaces; and unified outreach to researchers, educators, and the public. The initial members of AGR are the Gene Ontology Consortium[41] and six model organism databases: Saccharomyces Genome Database[42], WormBase[43], FlyBase[44], Zebrafish Model Organism Database[45], Mouse Genome Database[46] and Rat Genome Database[47]. This integration will provide the best visualizations and tools currently in use and allow efficient development of new tools in a collaborative manner. As the project moves toward deeper integration of content and software, we will provide easy-to-use cross-organism queries of the extensive data available in the component resources. The data

[35] http://lincsportal.ccs.miami.edu/dcic-portal/.
[36] http://www.lincsproject.org/data/data-standards/.
[37] http://lincsportal.ccs.miami.edu/apis/.
[38] http://bd2k-picsure.hms.harvard.edu.
[39] http://bd2k-picsure.hms.harvard.edu/example-01.html.
[40] http://www.alliancegenome.org.
[41] http://www.geneontology.org/.
[42] http://www.yeastgenome.org/.
[43] http://www.wormbase.org/.
[44] http://www.flybase.org/.
[45] http://www.zfin.org/.
[46] http://www.informatics.jax.org/.
[47] http://rgd.mcw.edu/.

access will be available via an API that will be conformant to the smartAPI specification.

The API Interoperability project is still an ongoing project, and there are a number of BD2K centers that have been actively participating in the WG meetings and have expressed interest in adopting and implementing the smartAPI specification and editor to annotate their APIs. Once we have annotated the APIs using the smartAPI editor, we will store them in the smartAPI registry[48], which will not only provide all of the smartAPI-conformant APIs in one location but will also be integrated into the editor. With this integration, the data and values will be used to suggest related fields and values for new similar APIs during the annotation process (refer to Sect. 5).

## 7 Conclusions and Future Work

In this paper, we have defined a smartAPI metadata template that contains 54 API metadata elements used to describe an API. Results are reported for a survey of eight resources that were used to identify these API-associated metadata. We constructed the smartAPI metadata template for the validation of API annotations. Additionally, we built a Web application for the intelligent annotation of smartAPIs. Since authoring metadata can be tedious and overwhelming, we developed a software built upon the already existing Swagger editor that will help users describe their APIs by (i) indicating highly used fields, (ii) suggesting commonly used values, and (iii) enabling the discovery and reuse of terms authored by others. Moreover, we developed a profiler for automatic annotation of API response data and integrated that within our editor to enable semantic annotation of APIs, which increases their reusability and interoperability.

Our proposal to facilitate the authoring of rich API metadata is especially significant because of the increased emphasis on providing cloud-based APIs. If left unmanaged, a majority of the APIs will lack the proper metadata needed to find APIs. As sketched out by the participants of the Software Discovery Index Workshop[49], our work begins to explore their roadmap to address challenges facing specifically the biomedical research community in locating, citing, and reusing biomedical software. We believe that the semantic tools and technologies developed in this project will form an important cornerstone in the overall vision of the Commons. As future work, we will assess the ease and utility of authoring smart API metadata for biomedical APIs as well as APIs in other domains. Although we have developed our own API repository (http://smart-api.info/registry/), we expect to be able to export to other repositories that generally have fewer metadata requirements, e.g. ProgrammableWeb. Additionally, our main aim for future work will be to focus on use cases that illustrate our aim of making the APIs interoperable.

[48] http://smart-api.info/registry/.
[49] http://www.softwarediscoveryindex.org/.

**Acknowledgments.** The smartAPI pilot project was funded as a supplement to CEDAR (U41HG000131521). Mygene.info and MyVariant.info are supported by U01HG008473 (from NHGRI). The LINCS Data Portal is supported by grant U54HL127624 awarded by the National Heart, Lung, and Blood Institute through funds provided by the trans-NIH LINCS Program http://www.lincsproject.org/ and the trans-NIH Big Data to Knowledge (BD2K) initiative http://www.bd2k.nih.gov.

## References

1. Alarcón, R., Wilde, E.: RESTler: Crawling RESTful services. In: Proceedings of the 19th International Conference on World Wide Web, pp. 1051–1052. ACM (2010)
2. Auer, S., Heath, T., Bizer, C., Berners-Lee, T.: LDOW 2016: 9th workshop on linked data on the web. In: Proceedings of the 25th International Conference Companion on World Wide Web, WWW 2016 Companion, International World Wide Web Conferences, pp. 1039–1040 (2016)
3. Bhagat, J., Tanoh, F., Nzuobontane, E., Laurent, T., Orlowski, J., Roos, M., Wolstencroft, K., Aleksejevs, S., Stevens, R., Pettifer, S., Lopez, R., Goble, C.A.: BioCatalogue: A universal catalogue of web services for the life sciences. Nucleic Acids Res. 38, W689–W694 (2010)
4. Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. In: Semantic Services, Interoperability and Web Applications: Emerging Concepts (2009)
5. Chinnici, R., Moreau, J., Ryman, A., Weerawarana, S.: Web Services Description Language (WSDL) Version 2.0. Part 1: Core Language. W3C Recommendation (2007). http://xml.coverpages.org/wsdl20000929.html
6. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.0. W3C Recommendation (2000). http://xml.coverpages.org/wsdl20000929.html
7. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL). W3C Note, March 2001. http://www.w3.org/TR/wsdl
8. Gomadam, K., Ranabahu, A., Sheth, A.: SA-REST: Semantic Annotation of Web Resources. W3C Member Submission, April 2010. http://www.w3.org/Submission/SA-REST/
9. Hsu, S.K., Lin, S.J.: MACs: Mining API code snippets for code reuse. Expert Syst. Appl. **38**(6), 7291–7301 (2011)
10. Ishag, M.I.M., Park, H.W., Li, D., Ryu, K.H.: Highlighting current issues in API usage mining to enhance software reusability. In: Proceedings of the 15th International Conference on Software Engineering, Parallel and Distributed Systems (SEPADS 2016). Recent Advances in Computer Engineering Series (2016)
11. Ison, J., Rapacki, K., Ménager, H., et al.: Tools and data services registry: A community effort to document bioinformatics resources. Nucleic Acids Res. **44**(1), D38–D47 (2015)
12. Kopecký, J., Gomadam, K., Vitvar, T.: hRESTS: An HTML microformat for describing RESTful Web services. In: Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology, pp. 619–625. IEEE Computer Society (2008)
13. Kopecký, J., Vitvar, T.: MicroWSMO. WSMO Working Draft, February 2008. http://www.wsmo.org/TR/d38/v0.1/
14. Kopecký, J., Vitvar, T., Bournez, C., Farrell, J.: Semantic annotations for WSDL and XML schema. IEEE Internet Comput. **11**, 60–67 (2007)

15. Lanthaler, M., Gütl, C.: Hydra: A vocabulary for hypermedia-driven Web APIs. In: Proceedings of the 6th Workshop on Linked Data on the Web, May 2013. http://ceur-ws.org/Vol-996/papers/ldow2013-paper-03.pdf
16. Lausen, H., Polleres, A., Roman, D.: Web Service Modeling Ontology (WSMO). W3C Member Submission, June 2005. http://www.w3.org/Submission/WSMO/
17. Maleshkova, M., Kopecký, J., Pedrinaci, C.: Adapting SAWSDL for semantic annotations of RESTful services. In: Meersman, R., Herrero, P., Dillon, T. (eds.) OTM 2009. LNCS, vol. 5872, pp. 917–926. Springer, Heidelberg (2009). doi:10.1007/978-3-642-05290-3_110
18. Martin, D., Burstein, M., Hobbs, J., Lassila, O.: OWL-S: Semantic Markup for Web Services. W3C Member Submission, November 2004. http://www.w3.org/Submission/OWL-S/
19. Milanovic, N., Malek, M.: Current solutions for web service composition. IEEE Internet Comput. **8**(6), 51–59 (2004)
20. Pedrinaci, C., Domingue, J.: Toward the next wave of services Linked services for the web of data. J-JUCS **16**, 1694–1719 (2010)
21. Scaffidi, C.: Why are APIs difficult to learn and use? Crossroads **12**(4), 4–4 (2006)
22. Torres, R., Tapia, B., Astudillo, H.: Improving web API discovery by leveraging social information. In: 2011 IEEE International Conference on Web Services, pp. 744–745 (2011)
23. Verborgh, R., Arndt, D., Van Hoecke, S., De Roo, J., Mels, G., Steiner, T., Gabarró Vallés, J.: The pragmatic proof: Hypermedia API composition and execution. Theory Pract. Logic Program. (2016). http://arxiv.org/pdf/1512.07780v1.pdf
24. Verborgh, R., Dumontier, M.: A web API ecosystem through feature-based reuse (2016). CoRR abs/1609.07108, http://arxiv.org/abs/1609.07108
25. Verborgh, R., Harth, A., Maleshkova, M., Stadtmüller, S., Steiner, T., Taheriyan, M., Van de Walle, R.: Survey of semantic description of REST APIs. In: Pautasso, C., Wilde, E., Alarcón, R. (eds.) REST: Advanced Research Topics and Practical Applications, pp. 69–89. Springer, New York (2014). http://link.springer.com/chapter/10.1007/978-1-4614-9299-3_5
26. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., et al.: The FAIR Guiding Principles for scientific data management and stewardship. Sci. Data 2 (2016). http://www.nature.com/articles/sdata201618
27. Wilkinson, M.D., Vandervalk, B., McCarthy, L.: The semantic automated discovery and integration (SADI) web service design-pattern, API and reference implementation. J. Biomed. Semant. **2**(1), 5–23 (2011)
28. Xin, J., Mark, A., Afrasiabi, C., Tsueng, G., Juchler, M., Gopal, N., Stupp, G.S., Putman, T.E., Ainscough, B.J., Griffith, O.L., Torkamani, A., Whetzel, P.L., Mungall, C.J., Mooney, S.D., Su, A.I., Wu, C.: High-performance web services for querying gene and variant annotation. Genome Biol. **17**(1), 91 (2016)