An Open Repository Model for Acquiring Knowledge about Scientific Experiments

Martin J. O'Connor, Marcos Martínez-Romero, Attila L. Egyedi, Debra Willrett, John Graybeal, and Mark A. Musen

Stanford Center for Biomedical Informatics Research
Stanford, CA 94305, U.S.A.
martin.oconnor@stanford.edu

Abstract. The availability of high-quality metadata is key to facilitating discovery in the large variety of scientific datasets that are increasingly becoming publicly available. However, despite the recent focus on metadata, the diversity of metadata representation formats and the poor support for semantic markup typically result in metadata that are of poor quality. There is a pressing need for a metadata representation format that provides strong interoperation capabilities together with robust semantic underpinnings. In this paper, we describe such a format, together with open-source Web-based tools that support the acquisition, search, and management of metadata. We outline an initial evaluation using metadata from a variety of biomedical repositories.

1 Introduction

To help tackle the reproducibility challenge in biomedical sciences, many funding agencies and journals are now demanding that experimental data are made publicly available [1]. These requirements have led to a dramatic increase in the availability of data sets derived from scientific experiments. High-quality metadata are seen as crucial to facilitate knowledge discovery with these data sets. The biomedical community has a strong history of tackling this metadata challenge by driving the development of *metadata templates* [2]. These templates focus on addressing the reproducibility challenge by providing detailed checklists of the metadata needed to describe particular types of experimental data sources. The key goal is to provide sufficient metadata to enable the source studies to be reproduced. A large number of public repositories have been built around these templates, greatly enhancing the ability of scientists to discover and share scientific knowledge [3-5].

While individual metadata templates can provide a standard format for a particular data source, they rarely share common structure or semantics. There is also a disconnect between the high-level checklist-based template definitions developed by scientific communities and the submission formats required by metadata repositories. Submission formats for biomedical repositories, for example, are typically spreadsheet-based, with a variety of *ad hoc* formats that require significant user effort to describe even very simple metadata content. These formats typically lack any

standard way of semantically annotating templates. Despite the availability of a large number of controlled terminologies in biomedicine, submission templates have weak or nonexistent mechanisms for linking terms from these terminologies to metadata submissions. These difficulties combine to ensure that typical metadata submissions are poorly described and thus require significant post-processing to extract semantically useful content. There is a pressing need for a standardized approach to representing templates and semantically describing their content.

In this paper, we outline such an approach under development by the Center for Expanded Data Annotation and Retrieval (CEDAR) [6]. We define a lightweight standards-based template model that provides principled interoperation with Linked Open Data. We describe associated tools that use this model to provide an end-to-end workflow for metadata acquisition and management. The ultimate goal is to address the fragmented landscape of metadata submission tools and template formats.

2 Related Work

There are a number of ongoing research efforts to address the metadata challenge in the biomedical domain. In the United States, the National Institutes of Health's Big Data to Knowledge (BD2K) initiative [7] is funding an array of projects to tackle different dimensions of this challenge. The BD2K-funded BioCADDIE [12] project is tasked with building a search engine for metadata that describe a variety of experimental types. Our project, CEDAR [6], also supported by BD2K, focuses primarily on the metadata authoring process, and is building tools to enable users to easily create and submit high quality metadata.

These initiatives build on decades of work creating reusable templates. Early work involved the creation of *minimum information models* [2], primarily centered on describing metadata for laboratory experiments. A minimum information model specifies the core set of required and optional metadata for particular experiment types. These models serve as template building blocks and have spawned a cottage industry for defining community-based templates for a large variety of biological sources. These templates are used by a host of public biomedical repositories [3, 4]. Minimum information models by themselves, however, do not solve the metadata challenge. They often specify only a small core set of information and, typically, their individual fields do not require values from specific ontologies. As a result, the collected metadata can be sparsely populated and have low semantic value.

Several initiatives have concentrated on building tools to increase metadata quality. Foremost among these is the ISA Tools [8], which is a desktop application to allow curators to create spreadsheet-based submissions for metadata repositories. The Linked ISA [9] evolution of this tool provides a means to interoperate with Linked Open Data, effectively adding controlled term linkage to templates.

A key shortcoming is the absence of an open, interoperable format for metadata exchange. While the ISA Tools support the standard MAGE-TAB format [10], it is a spreadsheet-based representation with limited expressivity and extensibility. There is a need for an open format built on Web-based standards. The recently developed FAIR Principles [11], which specify desirable properties of metadata, provide a set of important targets for this format to meet. Standard mechanisms of interoperating with Linked Open Data vocabularies are central to such a format.

3 Model and Implementation

We first developed a lightweight, abstract template model to specify the key aspects of template construction. This model represents the core structural characteristics of templates—the common entities and compositional patterns that define a template.

We then produced a concrete representation of the template model, emphasizing the addition of semantic markup and constraints. The concrete template model provides a consistent, interoperable information framework for defining templates and for creating and filling out metadata instances that correspond to those templates.

Finally, we developed a set of tools for creating metadata templates and for acquiring metadata to generate metadata instances.

3.1 Abstract Template Model

Our system needs to recursively compose templates from existing, more granular templates. In our model, we term these sub-templates *template elements*. Template elements constitute the building blocks of metadata templates. Template elements may contain one or more atomic pieces of information, such as a text or date field, or may be recursively composed from other template elements. *Template fields* are used to represent these atomic pieces of metadata. For example, a template field could be used to indicate the date at which a measurement was made for a particular scientific experiment. Template elements are used to recursively combine template fields or template elements to create more complex descriptions. For example, template fields Phone and Email could be contained in a template element called Contact Information, which could itself be contained in a template element called Person.

Figure 1 presents a basic overview of this model. The `Template`, `TemplateElement`, and `TemplateField` entities represent their namesake concepts. All entities have an `@type`[1] field that indicates the entity type, and are uniquely identifiable via an `@id` field. They also contain `title` and `description` fields. Template fields contain an `@value` field, which stores the field's value. A variety of built-in template field types are provided. These include a `TextField`, which represents a free text field, and a `ListField`, which represents a multiple-choice field. This set can be extended to incorporate additional field types. Both templates and template elements can optionally have fields or elements nested inside them. Template elements and fields can be grouped together in a `Template` to provide an overall description of a collection of metadata.

A *template instance* is created from a template. A template effectively serves as a structural specification of metadata instances conforming to that template.

This abstract model provides a structural specification of templates, elements and fields. In the next section we will outline the development of a concrete representation of this model. This representation extends beyond structural constraints and specifies how elements in the model are linked to controlled terms and how controlled-term–based value constraints can be specified for template instances.

[1] The @-prefixed notation follows JSON-LD; see Section 3.2.

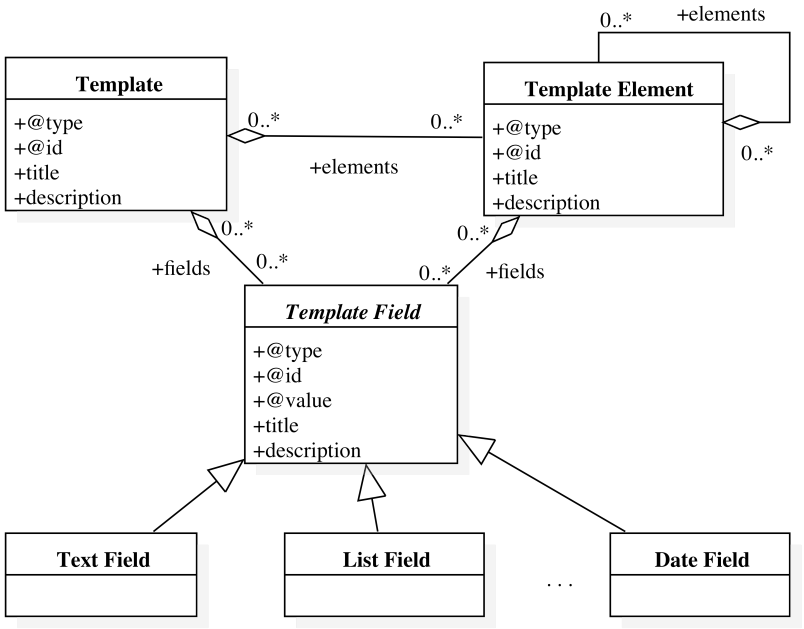

Figure 1. Abstract template model showing main model components

3.2 Template Model Concrete Representation

The template model requires a machine-interpretable representation for software systems to work with the model programmatically. This representation must meet a variety of goals. Primarily, it must describe the structure of templates and the instances generated from these templates. It must also describe and constrain the various relationships between the entities in the model. Template representations must be conveniently serializable so that they can be provided via REST APIs and persisted to storage media. Ideally, the representation should be based on standard formats so that existing tools can be used to manage model entities. The representation should also permit easy validation, and easy indexing to support search. To enable interoperation with controlled terms, a standardized means to annotate templates with controlled terms is key. Finally, the template format must interoperate with Linked Open Data technologies such as RDF and OWL, and allow metadata to be represented as RDF graphs.

Ontology languages such as RDF and OWL are not particularly suited to representing the structural constraints required for this sort of data-centric problem [20]. The current SHACL [21] effort aims to address this shortcoming by providing an RDF-based constraint language to structurally describe RDF data. Its goal is to "communicate information about data structures associated with some process or

interface, generate or validate data, or drive user interfaces". However, SHACL is not yet standardized and has almost no current tool support.

We identified two key JSON-based technologies that can be combined to meet many of the goals outlined above, while retaining full interoperation with semantic resources: JSON Schema [13], and JSON-LD [14]. Both are supported by a large variety of Web-centric tools.

JSON Schema is a technology for describing and validating the structure of JSON data. Its directives—themselves represented as standard JSON elements—can be used to provide a structural description of a JSON document. JSON documents that are specified with JSON Schema can be structurally validated against their associated schemas via off-the-shelf tools.

JSON Schema provides a structural specification only—it does not describe the semantics of JSON documents. A recent technology called JSON-LD ("Linked Data") was developed to meet this goal. JSON-LD provides a lightweight syntax to add semantic annotations to JSON documents. The key goals of JSON-LD are to support the use of Linked Data in Web-based programming environments, to build interoperable Web services, and to store Linked Data in JSON-based storage engines. JSON-LD effectively allows JSON documents and their contents to be made available as Linked Data, offering the potential for machine-interpretable RDF semantics [22].

We first outline how we use JSON Schema to describe the structure of templates and to constrain and validate the template instances generated from those templates. We then describe how we use JSON-LD to mark up these structural specifications, adding semantic content to these templates and instances. We show how this combination of JSON Schema and JSON-LD provides the capabilities to fully represent the template model and provide a strong bridge to semantic technologies[2].

3.2.1 Representing Template Structure using JSON Schema

With JSON Schema we define the structure of the primary entities in the CEDAR template model. We first outline its use to define the three core entities in the model: template fields, template elements, and templates.

Representing Template Fields using JSON Schema. Template fields are used to describe an atomic piece of metadata. Informally, they correspond to a single field in a form, which when filled out contains a single value. In principle, a template field could be stored as a simple JSON property value. However, in many cases we would like the option to add additional metadata to describe template fields. At a minimum, we want users to record a name and description of each field. Hence, we use a JSON object to describe template fields.

The template field representation includes a value field, in addition to the other descriptive information. We first define a simple `value` field of type string to hold the raw value. We can also use the standard JSON Schema `title` and `description` fields to hold a name and description for the field.

[2] A complete template model specification can be found at http://metadatacenter.org/cedar-template-model.

For example, here is the definition of a Study Title template field, which contains the full name of a study represented as a single string[3]:

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "title": "Study Title", "description": "Study title template field",
  "properties": { "value": { "type": "string" } },
  "required": [ "value" ], "additionalProperties": false
}
```

A conforming instance of this template field could look as follows:

```
{ "value": "Immune Biomarkers" }
```

Representing Template Elements using JSON Schema. Template elements support composition and can include a combination of multiple template fields and elements. They are represented using an approach equivalent to the one used to represent template fields. Again, we specify that a template element must be represented as a JSON object. We can then restrict each nested template field or template element using nested JSON Schema specifications.

For example, the definition of an Investigator template element is shown below. It contains one nested template field called `fullName`.

```
{
 "$schema": "http://json-schema.org/draft-04/schema#",
 "type": "object",
 "title": "Investigator",
 "description": "Investigator element",
 "properties": {
  "fullName": {
   "type": "object",
   "title": "Full Name",
   "description": "Full name template field",
   "properties": { "value": { "type": "string" } },
   "required": [ "value" ], "additionalProperties": false
   }
  },
  "required": [ "fullName" ], "additionalProperties": false
}
```

A conforming *template element instance* could look like the following:

```
{ "fullName": { "value": "Dr. P.I." } }
```

Representing Templates using JSON Schema. The representation of templates follows the same principles as template elements. Like template elements, templates can have nested elements and fields. A JSON Schema-encoded CEDAR template specification effectively defines the complete structure of a template instance.

[3] A JSON Schema validator can be found at http://www.jsonschemavalidator.net.

Here is an example of a template containing a template field called `studyTitle` and a nested template element `pi` describing a principal investigator:

```
{
 "$schema": "http://json-schema.org/draft-04/schema#",
 "type": "object",
 "title": "Study", "description": "Study template",
 "properties": {
  "studyTitle": {
   "type": "object",
   "title": "Study Title", "description": "Study title template field",
   "properties": { "value": { "type": "string" } },
   "required": [ "value" ], "additionalProperties": false
  },
  "pi": {
   "type": "object",
   "title": "Investigator", "description": "Investigator element",
   "properties": {
     "fullName": {
      "type": "object",
      "title": "Full Name", "description": "Full name template field",
      "properties": { "value": { "type": "string" } },
      "required": [ "value" ], "additionalProperties": false
     }
   },
   "required": ["fullName" ], "additionalProperties": false
  }
 },
 "required": [ "studyTitle", "pi" ],
 "additionalProperties": false
}
```

A corresponding *template instance* could look like the following:

```
{
  "studyTitle": { "value": "Immune Biomarkers"  },
  "pi": { "fullName": { "value": "Dr. P.I." } }
}
```

3.2.2 Representing Template Semantics using JSON-LD

JSON Schema is useful for defining structural restrictions on JSON documents. It can also be used to specify basic type restrictions on field values. However, it provides a very basic set of built-in type restrictions. It also does not provide a way to add additional types or to interoperate with types defined in external sources.

As mentioned, JSON-LD [14] was developed to meet this goal. JSON-LD provides a lightweight syntax to add semantic annotations to JSON documents that can restrict the types and values of fields using terms from external vocabularies. Like JSON Schema, it adds some custom fields with well-known names to a JSON document to provide additional markup information.

JSON-LD provides four core fields to add semantic markup to JSON documents: `@context`, `@type`, `@id`, and `@value`. The `@context` field is used to define prefixes for controlled vocabularies and to map JSON properties to controlled vocabularies; the

`@type` field indicates the semantic type of a JSON object; the `@id` field gives a unique identifier to a JSON object; finally, the `@value` field holds literal values, and can optionally be given a data type.

Here, for example, is a JSON-LD–enhanced template instance representing a study (with JSON-LD clauses in bold):

```
{
  "@type": "http://semantic-dicom.org/dcm#Study",
  "@id": "https://repo.metadatacenter.org/template_instances/55417",
  "@context": {
    "studyTitle": "https://schema.org/title",
    "pi": "https://mschema.org/property/hasPI"
  },
  "studyTitle": { "@value": "Immune biomarkers study" },
  "pi": {
    "@type": "https://schema.org/Person",
    "@context": {"fullName": "https://schema.org/name" },
    "fullName": { "@value": "Dr. P.I" }
  }
}
```

Note that we have added JSON-LD `@context`, `@type`, `@id`, and `@value` fields to provide semantic markup. The `@context` field maps JSON properties to properties in controlled vocabularies; the `@type` field indicates the semantic type of the instance, which in the case above is the Study class in the Radiation Oncology Ontology; the `@id` field gives a unique identifier to the template instance; finally, the `@value` field is used to store literal values. We use this field in our JSON-LD–enhanced content to replace the previous *ad hoc* `value` field we introduced earlier.

The JSON Schema specification can ensure that conforming instances are marked up with JSON-LD, both by demanding that specific fields are present and by restricting the content of those fields. For example, here is a JSON Schema template specification for the above study instance with clauses (marked in bold) ensuring that conforming instances carry appropriate JSON-LD markup:

```
{
 "$schema": "http://json-schema.org/draft-04/schema#",
 "@type": "https://repo.metadatacenter.org/core/Template",
 "@id": "https://repo.metadatacenter.org/templates/4353",
 "title": "Study", "description": "Study template",
 "properties": {
  "@context": {
    "properties": {
      "studyTitle": { "enum": [ "https://schema.org/title" ] },
      "pi": { "enum": [ "https://mschema.org/property/hasPI" ] }
    },
    "required": [ "studyTitle", "pi" ], "additionalProperties": false
  },
  "@type": { "enum": [ "http://semantic-dicom.org/dcm#Study" ] },
  "@id": { "type": "string", "format": "uri" },
  "studyTitle": { ... }, "pi": { ... }
 },
 "required": [ "@context", "@type", "@id", "studyTitle", "pi" ],
 "additionalProperties": false
}
```

As can be seen in this example, the JSON Schema template specification can ensure that template instances contain a significant amount of JSON-LD–encoded type information. Here, we are forcing the `@context`, `@type`, and `@id` fields in an instance to carry specific controlled terms. These instances can be automatically checked for conformance against the template specification. This use of JSON Schema is completely standard and instance validation can be performed with off-the-shelf tools. We also developed a JSON Schema-based validation schema that can be used to validate template, elements, and fields.

Generation of RDF from Template Instances. Since JSON-LD is effectively an RDF serialization, we can automatically produce RDF from JSON-LD marked up documents. Here, for example, is the RDF generated from the earlier study template instance (in Turtle syntax, with prefixes defined for clarity):[4]

```
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dcm:    <http://semantic-dicom.org/dcm#> .
@prefix schema: <https://schema.org/> .
@prefix mscp:   <https://mschema.org/properties/> .
@prefix tinst:  <https://repo.metadatacenter.org/template_instances/> .
@prefix teinst: <https://repo.metadatacenter.org/element_instances/> .

tinst:55417 rdf:type       dcm:Study ;
            schema:title   "Immune biomarkers study" ;
            mscp:hasPI     teinst:5423 .
teinst:5423 rdf:type       schema:Person ;
            schema:name    "Dr. P.I" .
```

The ability to automatically generate RDF graphs from template instances provides a seamless path to interoperate with RDF-based Linked Data resources.

3.2.3 Value Constraints

JSON Schema allows us to express a very limited set of value constraints. It can, for example, state that the value of a field should be a particular value, or selected from a set of values. It can also restrict a field value to be of a particular type or format. We require more advanced constraints on field values, to specify that values must come from controlled terminologies. For example, we may specify that the value of a field should be selected from a set of URIs identifying classes from a particular ontology.

We provide five main constraint types for controlled terminology fields. We can constrain the possible values for a particular field to (1) classes from specific ontologies, (2) a set of classes, (3) ontology branches, (4) value sets, and (5) literals. Here, value sets are non-hierarchical collections of controlled terms. The possible values of a field can also be composed of some combination of the above five constraint types; the union of all five represents the set of possible field values.

[4] A useful online JSON-LD tool can be found at http://json-ld.org/playground.

To represent these value constraints, we use a `_valueConstraints` field that is placed inside a template field. The `_valueConstraints` field has five possible subfields for the five value constraint source types. We also include the ability to specify that the user may supply multiple entries when filling out template fields.

The top level JSON format adopted for our metadata instances is as follows:

```
{
  "_valueConstraints": {
    "ontologies": [ ... ],
    "classes": [ ... ],
    "branches": [ ... ],
    "valueSets": [ ... ],
    "literals": [ ... ],
    "multipleChoice": true
  }
}
```

The `ontologies` field specifies ontologies as controlled term sources for field values. Similarly, the `valueSets` field specifies values sets as term sources. The `branches` field is analogous to the `ontologies` field, but restricts values to branches within ontologies. The `classes` field indicates a set of classes as acceptable values. Finally, the `literals` field specifies a set of literals.

Here is an example showing a branches constraint for a field:

```
{
  "_valueConstraints": {
    "branches": [
      {
         "uri": "http://purl.obolibrary.org/obo/OBI_0000070",
         "includesRoot": false
      },
      {
         "uri": "http://www.co-ode.org/ontologies/galen#Assay",
         "includesRoot": false
      }
    ],
    "multipleChoice": true
  }
}
```

It restricts the possible values to classes in branches rooted in assay classes in the Ontology for Biomedical Investigations and in the GALEN ontology.

The four other types use a similar approach to constrain possible field values.

3.3 Template Design and Metadata Acquisition Tools

Using this model as a foundation, we built a set of tools for the acquisition, storage, search, and reuse of machine-readable metadata templates and instances [6]. We collectively refer to these tools as the CEDAR Workbench.[5]

[5] The CEDAR Workbench is available at https://cedar.metadatacenter.net.

Template Designer, Metadata Editor, and Metadata Explorer. We developed three highly interactive Web-based tools to simplify the process of managing metadata templates and instances (Figure 2). The Template Designer allows users to create metadata templates. These templates are stored in CEDAR's template repository. The Metadata Editor tool uses these templates to automatically generate a forms-based acquisition interface for entering metadata. Entered metadata are stored as template instances in CEDAR's metadata repository. The Metadata Explorer tool can be used to perform faceted queries on the metadata stored in this repository.

A key focus is on interoperation with ontologies. Using interactive look-up services linked to the BioPortal ontology repository [15], the Template Designer allows template authors to find terms in ontologies to annotate their templates and to restrict the values of template fields. Users entering metadata using the Metadata Editor are prompted in real time with drop-down lists, auto-completion suggestions, and verification hints, significantly reducing their errors while speeding metadata entry and repair. This lookup is driven by the value constraints specified in templates.

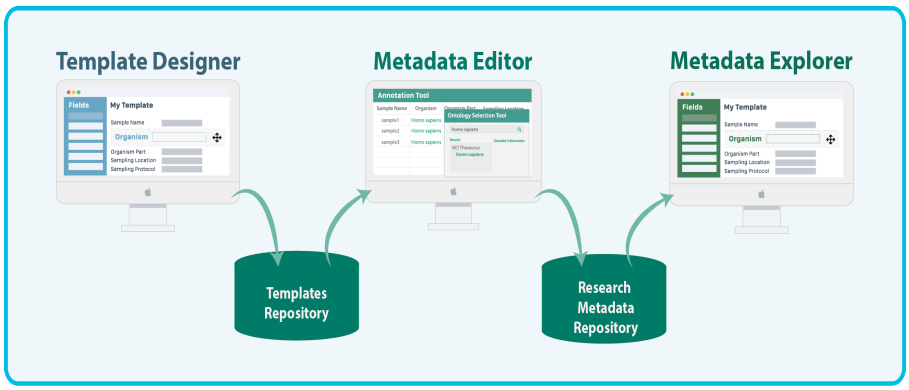

Figure 2. Template and Metadata repositories, and Template Designer, Metadata Editor, and Metadata Explorer tools

Intelligent Authoring. To ease the burden of authoring high quality metadata, we developed a recommender framework that learns associations among data elements to suggest context-sensitive metadata values [19]. In the Metadata Editor the system can recommend possible values for metadata fields during submission. These value suggestions are based on analyses of instances in our metadata repository.

Web Services. We developed an array of REST-based Web services to manage model artifacts. These services provide functionality to store and query templates and instances. We also provide authentication and authorization services to restrict access to artifacts and to allow users to group their artifacts into collections.

4 Initial Evaluation

CEDAR is working with several major community-based groups to perform initial evaluations of our metadata model and tools. Core collaborating groups include (1) ISA [8], which develops tools for submitting metadata for biomedical experiments, (2) ImmPort [5], a data warehouse of immunology-related datasets, and (3) the Human Immunology Project Consortium (HIPC) [17], which designs new metadata templates and supplies experimental datasets to the ImmPort repository.

Our first evaluation assessed the ability of our model to represent real-world metadata models and instances from these collaborating groups. We also evaluated the model's ability to comprehensively link metadata elements and value constraints to controlled vocabularies and ontologies. We evaluated several other essential criteria: the use of the model-driven tools to support the development of model-specified templates; the instantiation of detailed metadata using the model; and the final submission of metadata to external repositories.

Modeling and Template Creation. In collaboration with the ISA, ImmPort, and HIPC teams, we jointly developed a template model (the 'CEDAR Study Model') to serve as a common representation for metadata in the ImmPort system and in a selection of ISA-populated repositories. The structure of this common model was heavily based on the ISA Model [8], which provides a rich description of metadata typically collected when performing experimental studies. The resulting model provides a comprehensive description of metadata for all studies in the ImmPort system, and of metadata in public repositories populated by the ISA tool chain.

We successfully used the Template Designer tool to create a representation of this common CEDAR Study Model. We also represented models from the LINCS Consortium [18] and from the Gene Expression Omnibus (GEO) [3] data repository using our template model. We further generated metadata instances following the corresponding instance model, as described below.

Metadata Acquisition. Using these templates we tested the ability to acquire metadata. The ImmPort team generated metadata instances conforming to our CEDAR Study Model for all 146 public studies in their system, and the ISA team performed a similar process for 300 publicly available studies. After correcting the model to address minor representation issues, we found all these target metadata instances could be represented in the core model.

Using the GEO template, we ingested all public GEO metadata as instances of this template. We indexed these instances using our Elasticsearch-based recommender system, and generated distribution profiles for all metadata fields. We then manually created experimental GEO submissions via this tool chain, and confirmed the relevance of the field-based recommendations that resulted [19].

Controlled Term Linkage. In addition to representing the structural constraints of the common studies model, we performed an analysis of all elements, fields, and value constrains in the model to identify appropriate controlled-term linkages for ImmPort metadata. In collaboration with members of the HIPC and ImmPort teams,

we comprehensively annotated the model with appropriate controlled terms from a variety of biomedical ontologies. We also specified value constraints for controlled fields to ensure that the generated acquisition interfaces restricted acquired metadata to appropriate vocabularies. In the cases where custom value sets were required for fields, we used recently developed BioPortal functionality to create user-defined value sets. The final model (and tools that used it) successfully represented all type and value constraint information for the ImmPort metadata.

Metadata Submission. The final stage in the evaluation process involved assessing metadata submission to public repositories. We first targeted the NCBI BioSample [4] repository, which captures metadata about biological samples. In addition to a spreadsheet-based submission process, BioSample has an experimental XML-based submission portal. The structure of this submission is described in an XML Schema document, and an online validator is available to validate submissions.

For this test, we used the Template Designer to develop a template for the metadata described in the BioSample XML Schema. Starting with this template, we used our Metadata Editor to create sample metadata. We then transformed this metadata to the BioSample XML-based format and submitted it to the validator to ensure that it validated. In collaboration with NCBI, we have begun the process of developing a BioSample submission portal that can optionally be used by metadata submitters.

We are also working with the ImmPort representatives on our team to develop a similar submission pipeline for the ImmPort repository. Instead of manually generating a template using the Template Designer, the ImmPort team programmatically generated templates that conform to our template model. These templates form the basis of automatically generated user acquisition interfaces. With the ImmPort team, we are pursuing incorporation of our submission interface into their existing pipeline, toward acquiring real user submissions in the future.

Once these end-to-end submission pipelines are in place in production workflows, we will rigorously evaluate the quality of metadata collected compared to existing spreadsheet-based approaches.

5 Conclusion

We have described a standards-based template representation model that provides a common format for describing metadata templates and instances. The representation focuses on meeting the goals of the FAIR principles [11] and on interoperating with Linked Open Data. It also provides mechanisms to support template composition, with the aim of increasing reuse of descriptive metadata fragments across templates.

A key focus throughout is strongly linking metadata templates with controlled vocabularies and ontologies. In addition to semantically marking up the templates themselves, thereby specifying a rigorous link between template fields and well-specified semantic concepts, designers can specify ontology-based value constraints to ensure that metadata conforming to these templates are linked to controlled terminologies.

We developed a set of tools that leverages the template model to support the end-to-end management of metadata templates and instances, from initial template creation to final metadata submission. Two core tools focus on simplifying the process of managing templates: (1) the Template Designer allows users to create, search, and author metadata templates; this tool automatically produces a user interface specification from a template. (2) The Metadata Editor tool uses this interface specification to generate a forms-based acquisition interface for acquiring metadata conforming to the template. The system also provides an array of REST APIs, enabling access to all templates and metadata collected using those templates.

We performed a variety of internal evaluations with collaborators in the ISA [8], ImmPort [5], and HIPC [17] groups. These analyses focused on different stages of the metadata management process, from initial template creation, to final submission and validation. The evaluations demonstrated the fundamental viability of the template model. The next evaluation steps for the template model include supporting and evaluating user submissions to existing public metadata repositories, with the BioSample [4], ImmPort [5], and LINCS [18] repositories as the first targets. Soon, we expect to establish additional markup leveraging the JSON-LD constructs to support more advanced interoperability scenarios.

The use of JSON Schema provided a standard technology to represent all structural aspects of CEDAR's template model. No invention of new constructs was required—all structural constraints were captured directly by core JSON Schema clauses. Off-the-shelf tools proved usable for model validation. The use of JSON-LD provided a robust bridge between our model and semantic technologies. With JSON-LD we retained the full expressivity and power of semantic technologies, while gaining the practical advantage of the massive availability of Web-centric tooling. Since JSON-LD is effectively an RDF serialization, we can go back and forth between the two representations using standard tools. Again, no new JSON-LD constructs were required to represent the semantic markup required by CEDAR's model.

The JSON-based approach eased adoption by groups unfamiliar with semantic web technologies, and provided a low-overhead pathway for users to incrementally add semantic concepts. We have observed how JSON-LD facilitates incremental semantic enhancement of metadata, as our partner teams created simple structural metadata and then gradually added comprehensive semantic markup. We think such an incremental approach is key to bringing semantic web technologies into the wider Web community. We also note that several biomedical initiatives focused on metadata, such as BioCADDIE [12], have begun using JSON-LD to describe their metadata.

All software and schemas described in this paper are open source and available on GitHub [23]. We released a public alpha version of CEDAR in September, 2016.

Acknowledgments

CEDAR is supported by the National Institutes of Health through an NIH Big Data to Knowledge program under grant 1U54AI117925. NCBO is supported by the NIH Common Fund under grant U54HG004028. We appreciate the collaborations offered by the ImmPort, BioSharing, HIPC, and LINCS communities.

References

[1] Borgman CL. The conundrum of sharing research data. J Am Soc Inform Sci Technol, 2012; 63(6):1059–1078.

[2] Tenenbaum JD, Sansone S-A, and Haendel MA. A sea of standards for omics data: sink or swim? JAMIA, 2014; 21(2):200–203.

[3] Edgar R, Domrachev M, and Lash AE. Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. Nucleic Acids Research, 2002; 30(1):207–210.

[4] BioSample. http://www.ncbi.nlm.nih.gov/biosample. Retrieved 15th September, 2016.

[5] Bhattacharya, S., et al. ImmPort: disseminating data to the public for the future of immunology. Immunologic Research, 2014; 58(2-3).

[6] Musen MA et al. The center for expanded data annotation and retrieval. Journal of the American Medical Informatics Association, 2015; 22(6):1148-52.

[7] BD2K. https://datascience.nih.gov/bd2k. Retrieved 15th September, 2016.

[8] Sansone S-A, Rocca-Serra P, Field D, et al. Toward interoperable bioscience data. Nature Genetics, 2012; 44(2):121–126.

[9] Rocca-Serra P, Brandizi M, Maquire E, et al. ISA software suite: supporting standards-compliant experimental annotation and enabling curation at the community level. Bioinformatics, 2010; 26(18):2354–2356.

[10] Rayner TD et al. A simple spreadsheet-based, MIAME-supportive format for microarray data: MAGE-TAB. BMC Bioinformatics, 2006; 489(7).

[11] Wilkinson MD et al. The FAIR guiding principles for scientific data management and stewardship. Scientific Data, 2016; 15(3).

[12] Nosek BA el al. Promoting an open research culture. Science, 2015; 6242(348):1422-1425.

[13] JSON Schema. http://json-schema.org. Retrieved 15th September, 2016.

[14] JSON-LD. http://json-ld.org. Retrieved 15th September, 2016.

[15] Musen MA, Noy NF, Shah NH, et al. The National Center for Biomedical Ontology. JAMIA, 2012; 19(2):190–195.

[16] ELIXIR: A distributed infrastructure for life-science information. https://www.elixir-europe.org. Retrieved 15th September, 2016.

[17] Maecker H et al. Standardizing immunophenotyping for the Human Immunology Project. Nature Reviews Immunology, 2012.

[18] LINCS. http://www.lincsproject.org. Retrieved 15th September y, 2016.

[19] Panahiazar M et al. Context aware recommendation engine for metadata submission. Workshop on Capturing Scientific Knowledge, 2015.

[20] Motik B, Horrocks I, and Sattler U. Adding Integrity Constraints to OWL. OWLED, 2007; Vol. 258.

[21] SHACL. https://www.w3.org/TR/shacl/. Retrieved 15th September, 2016.

[22] JSON-LD Use Cases. https://www.w3.org/2013/dwbp/wiki/RDF_AND_JSON-LD_UseCases. Retrieved 15th September, 2016.

[23] CEDAR GitHub Organization. https://github.com/metadatacenter. Retrieved 15th September, 2016.