Chapter 3

Components

Theodore Gerontakos

Application profiles (APs) have four major components: (1) the *application*, (2) the *entities* that the AP aims to describe, (3) the *properties* that describe those entities, and (4) the *values* assigned to those properties.

APs and their components have been variously described, most notably in the literature of the Dublin Core Metadata Initiative (DCMI).[1] DCMI's work on APs is an outstanding body of work with which all metadata professionals should be familiar. Many creators of APs, however, are not metadata professionals and will likely find this literature difficult to read. Even many metadata professionals look at the Dublin Core Abstract Model, for example, and wonder, "What does this have to do with the actual work I do?" They file it away and tell themselves, "I'll look at this soon," then don't.

Many people not metadata professionals find themselves tasked with creating an AP. Metadata professionals often advise these people in their efforts, but pointing them toward the DCMI literature is not always the best option. Metadata professionals need to know the DCMI literature; others do not. To play an advisory role, metadata professionals should be aware of what is needed to create APs that function well in the current information landscape and simplify the process for those they advise.

The components of APs have not changed dramatically in the past ten years; the way we write them and the sorts of information they contain have changed. The survey of components below explores familiar AP instruments in the new context, made novel mostly by the introduction of linked-data practices. For those comfortable with AP issues, hopefully we will explore some areas of interest; for others it may serve as a launch into the new context, perhaps even help start or continue updating aging models; however, this exploration most frequently addresses the needs of metadata professionals advising those working on projects that need APs but lack the expertise, which includes many librarians who are not metadata librarians but are managing projects.

The Application

An AP is a description of descriptions. It describes to data creators (and others) how to describe something. The application is an application of these descriptions described by the AP. The AP is a set of rules that is completely unnecessary without an application. Strictly speaking, the application is not a component of an AP; rather, it is a determinant, the reason the AP exists.

The use of the word *application* is confusing: there is a distinction between an application for descriptions and the software application that provides an implementation. When we use the term *the application*, we mean the former; when we mean the latter, we use *platform*. Besides the word *application*, more confusion results because many platforms come out of the box with a predetermined application profile. To further complicate matters, we often find that our APs are written for specific platforms. It is not always easy for writers of APs to distinguish between the application and the platform.

All applications have unique needs. The AP is a model that describes descriptions suitable for those unique needs. If the chosen platform comes with an application profile, our job is to understand our metadata needs, choose an appropriate subset of available options, refine the subset, and, if possible, extend it. If we have to create our own AP, we do that by choosing subsets of metadata components that already exist; they just exist outside the chosen platform. In either case, the task of writing an AP is similar: we develop an understanding of the application and design descriptions that will optimize the application's performance.

There is a significant difference between APs written for small projects with relatively low metadata expectations and those for large institution-wide efforts, such as building an institutional repository, where we should expect professional-level data modeling. Small projects can be modeled with short-term visions and informal methods, preferably with some help from metadata professionals. Projects with high expectations involve people across multiple professions, including metadata professionals, who would be deeply involved in modeling the repository. In such cases, professional modeling practices should be scrupulously followed.

Whatever the degree of professionalism needed, the modeling process requires an understanding of the application. Perhaps the fullest description of the AP process is DCMI's Singapore Framework, which can be implemented with varying levels of professionalism.[2] It is an extremely useful recommendation that still has not been widely adopted. Following the Singapore Framework, a metadata application profile, or *description set profile*, is achieved after careful analysis of the application. The instruments for analyzing the application include an evaluation of functional requirements and the creation of a domain model. No specific syntax is required; functional requirements can be a bulleted list of what's required for the application to function well; it can also be written as structured data. Similarly, the domain model could be diagrammed with a pen on an 8½″ × 11″ sheet of paper; it can also be created using data-modeling tools such as the Unified Modeling Language.[3] In both cases, for most projects, the documentation created at this point may never be reviewed after the application is implemented; what we're creating is a full view of the application that allows us to write our most useful AP. The AP is the documentation we would expect to be used—by data creators and data consumers—after the application is deployed.

When creating models or making recommendations, we often overlook the fact that libraries have worked strenuously to create a data model for library data. In 1997, the *Functional Requirements for Bibliographic Records* (FRBR) was formally released and was updated in 2017 by the *Library Reference Model* (LRM).[4] These are high-level conceptual models that provide a common model for application profiles. They are particularly interested in end users and their use of our applications. They offer a range of instruments for describing the entities end users want to find, identify, select, obtain, or explore (those are the five "user tasks" identified in the model). For most applications and for most platforms, these models are too complex; the most obvious problem is that most of our systems do not differentiate the work, expression, manifestation, and item entities. Nevertheless, the models at the very least can provide a solid foundation for AP development. The LRM can serve as the starting point for entities in our domain model, as well as for entity attributes, the entities' relationships with other entities, and their alignment with the user tasks.

LRM is a rich source of AP components and is the basis of our current cataloging code, Resource Description and Access (RDA). Both can be used as the basis for creating an overall model, or at least an understanding, of our application, no matter how small the project. This can include evaluating functional requirements, enumerating use cases, creating a domain model that diagrams the things we are describing and how they are related, and consulting already-existing models for the components that will populate our APs. There is no better place to start the AP *process* than an overall evaluation of the application.

Entities

For many of us, including metadata professionals, the word *entities* is relatively new as an everyday term (despite its use over twenty years ago in FRBR). It refers to the things our applications describe: people, places—everything; concepts (such as subjects); information resources; even knowledge.

We need to identify the things we are describing—that's obvious. Libraries already describe things. The most common things we describe are formats of information resources: a monograph, a map, a sound recording, and so on. We also describe agents, especially in our name authority file. We describe subjects in our subject authority file. Libraries are well acquainted with describing things, just not in a manner fit for the 2020s. There have been several changes in the way things are identified, isolated, described, and brought into relation.

The major reason for these changes is the widespread adoption of linked-data practices. We want to assign Internationalized Resource Identifiers (IRIs) to things. The IRIs act as surrogates for the thing itself. We want to add descriptions of each thing, especially specifying the type of thing, and get the thing, or thing-surrogate, and its descriptions in the web, directly accessible as structured data and not mediated, for example, by a splash page or represented by a record in a library catalog.[5] We want to give our things a web identity or, more precisely, a Semantic Web identity, allowing Semantic Web processors to recognize the thing as a discrete entity.

Everything can be "entified" for the Semantic Web. Not only is the information resource now an entity complex (work/expression/manifestation/item), but it is also related to many other entities. Obvious related entities are the people-entities who produce an information resource, such as the author, the publisher,

and so on; less obvious entities include the title or even the intended audience as things in themselves, things that can be described, identified by an IRI, and assigned a type. Once entified, the entities can then be related using properties.

Application models, especially a domain model, should reveal all types of entities for which our application needs to assign IRIs. Will we be assigning IRIs to works? Expressions? Agents? Keywords? Our APs should describe how to create full descriptions of all things to which we assign IRIs. For all things identified as part of the application but for which we're not assigning IRIs, our APs should identify who has assigned IRIs for those things and how to access those IRIs as well as any additional information about the things.

Unfortunately, most of our current platforms are not optimized for working with IRIs. Some do not create, or "mint," IRIs. Some do not recognize IRIs as actionable links. Some cannot follow an IRI and retrieve external data using that IRI. Then how many systems recognize entities—such as work/expression/manifestation—or allow for entity creation? Very few! This leads to a range of social problems, from administrators reluctant to invest in practices that show little return on investment, to colleagues who are openly hostile to linked data. So what are some responses we can adopt now, while in this transition, especially in regard to modeling entities?

One response is to do little or nothing; just keep creating records. This is not necessarily a poor approach. It represents a confidence that we can give our entities web identities sometime in the future. Our APs in this case look like APs we've been creating the past twenty years. There are very few entities. For example, if we were creating records for data sets, we would likely have only one entity: data sets (see figure 3.1).

If we were describing multiple entities, we would create something that looks more like figure 3.2.

When records are being created, the burden of the AP becomes the description of properties and values, usually for a single entity; in addition, the values will likely be text strings, and the creator of the AP will want to carefully select sources of values. Use of widely adopted standards, best practices, and high-quality well-defined (in the AP) metadata will help ensure future entification. That should not be a revelation: high-quality metadata, using reliable sources for values, has been recommended as a solution to many problems for many decades.

Another response is to start inserting IRIs for entities into our records. This is what libraries are doing that follow the PCC's URIs in MARC Pilot.[6] This can be done in many environments. It does require more work when entering metadata, and the return on investment is not immediately apparent, so it can lead to some resistance. This practice, like the do-nothing approach, represents a confidence that entification and Semantic Web identities will be relatively easily accomplished in the future. The APs we write today would, in this case, need to describe a method for entering entity IRIs (see figure 3.3).

Because many of our systems are not well-suited at present for managing entities, the problem of entities

Entity we are describing: Data sets

Properties we will use to describe the data sets:

Creator (Agent)

Instructions: Enter the name of the creator.

Keyword

Instructions: Enter a keyword.

Description/Abstract

Instructions: Enter a description of the data set.

Identifier

Instructions: Enter the identifier for the data set.

Figure 3.1
Imaginary AP for creating records describing data sets

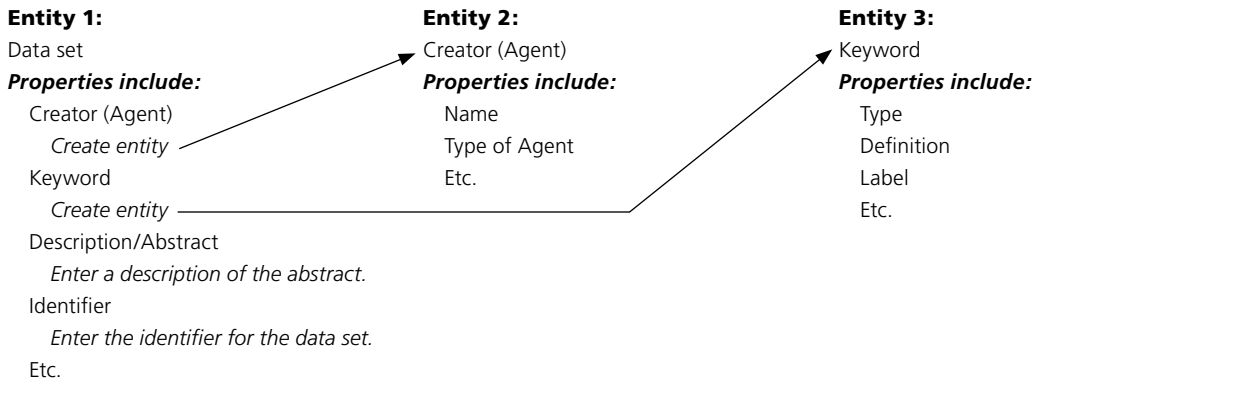

Figure 3.2
Imaginary AP for creating data describing multiple entities associated with data sets

Entity we are describing: Data sets

Properties we will use to describe the data sets:

Creator (Agent) 1

Instructions: Enter the name of creator 1 as it appears in English in Wikidata.

IRI for creator 1

Instructions: Enter the Wikidata IRI for creator 1.

Creator (Agent) 2

Instructions: Enter the name of creator 2 as it appears in English in Wikidata.

IRI for creator 2

Instructions: Enter the Wikidata IRI for creator 2.

Keyword 1

Instructions: Enter a keyword as it appears in LCSH.

Keyword 1 IRI

Instructions: Enter the LCSH IRI for keyword 1.

Keyword 2

Instructions: Enter a keyword as it appears in LCSH.

Keyword 2 IRI

Instructions: Enter the LCSH IRI for keyword 2.

Description/Abstract

Instructions: Enter a description of the data set.

Identifier

Instructions: Enter the identifier for the data set.

Figure 3.3
Imaginary AP for creating records including IRIs for selected entities

can be seen as a future problem. Another thing to do now is to get involved in projects and initiatives that incorporate the new models. This is an excellent way to stay current with many practices, including new ways of writing APs. The projects under "Linked Data for Production," featuring the linked-data platform Sinopia, are a good example.[7] PCC has had several groups that are working in this arena.[8] A related approach is to maintain, especially in academic libraries, a culture of exploration and research. We should avail ourselves of current information about the benefits and pain points of linked data. Another possibility, something more concrete, is to form an understanding of RDA and LRM as RDF models and start writing application profiles for RDA entities that can be used in actual practice. This last suggestion can apply to other ontologies beyond RDA; many organizations, for example, are writing profiles using the BIBFRAME ontology. This sort of current practice will hopefully ease the difficulty of future entification projects.

Having moved from creating records to creating data signals a significant change in what we do and creates uncertainty in our APs about how to handle entities. At present, we know we need to continue identifying the things we plan to describe and, at an early stage in the modeling process, establish a general idea of how to describe them. In the new context, the most pressing problem revolves around IRIs and (1) identifying the entities for which we create IRIs and (2) finding already-existing IRIs for the other entities in our data; however, the implications are much broader. Exactly what metadata is, is up for grabs. Our *records* describe information resources (primarily manifestations), agents, and subjects as discrete things, then include textual references to lots of related things. Now when we create data, the list of things we describe, or seek descriptions for, has grown. Our direction is toward the creation of, or linking to, data describing many types of entities and bringing them into relation. In other words, we're now creating first class data and not only metadata records. Perhaps soon we will not call ourselves metadata professionals but library data professionals?

Properties

Identifying properties and the correct way to use them is arguably the main purpose of an AP. Most APs are a list of properties and the rules for using them. Although few people look at APs, almost all our users are acquainted with properties. They've been in use for a really long time. In recent times past, we have called them by other names: *field name*, *data element*, *attribute*, *predicate*. Here we call them *properties* and assume a general understanding.

We also favor a current trend in thinking of properties as relationships; specifically, relationships between entities or, more expansively, between resources, because the value of the property may be a literal and is not necessarily an entity. To further complicate matters, LRM and RDA, libraries' primary data models, distinguish relationship-properties from attribute-properties.[9] Nevertheless we remain content to consider the property a relationship between two resources or, otherwise stated, two nodes.

In application profiles, we generally do not define properties; that is done elsewhere, most notably in ontologies or in element sets. Sometimes our AP will require a property that apparently is not defined anywhere; in that case, preferred practice is to define the property, publish the definition, then use it in the AP. The AP assembles predefined properties from multiple sources and clarifies their local use. There are not that many areas that APs clarify for properties, usually ten or so. Figure 3.4 shows five commonly seen properties of properties that we see in almost every AP.

These properties of properties are not terribly complex. What is difficult is deciding. Why is a particular property required? That is an organizational problem. In the miniature AP in figure 3.4, the most complex property is usage notes. It could be called by another name, say *input instructions*; these are specific rules on creating values, which can get quite complex. Often, we find our initial rules do not accommodate all cases, which can lead to a labyrinth of rules in our APs. It can be difficult to find a balance between simplicity and complexity.

Other complexities keep AP creation mostly an endeavor of specialists. These include the need to understand the original context of a property. A property's original context is usually a syndetic structure that can get quite complex. For example, the original context may describe a hierarchy of properties; when we climb the hierarchy, we find our property is in a chain of properties intended to describe humans only. If we use that property to describe dogs, our APs will contribute to the creation of lower quality data.

The complexity we will focus on here concerns changes due to the widespread adoption of linked-data practices. Despite those changes, we can view current efforts as a continuation of what we have always done.

Required: yes

Repeatable: no

Data Type: date

Definition: date associated with the item

Usage Notes: enter the date the item was created; enter the year only

Figure 3.4
Common properties of properties

We still need metadata professionals (catalogers) experienced in data creation—people who know good sources of values, understand the complexities of particular fields like a date field, and so on. The same people, however, were educated in the MARC format; now we need similar expertise in RDF *and* in a handful of additional data models (RDA, BIBFRAME, etc.).

The most notable change is that several properties of properties have become commonplace. Sometimes these properties of properties will be explicit in a source ontology, in which case we can just repeat them in the AP for convenience; other times selected properties of properties needed in the local application will not appear in the source but should be included in the AP. These properties of properties include the following:

IRI

Properties now have IRIs. The IRI should be explicit in the property's original context. We repeat it in the AP for convenience. If a property does not have an associated IRI, we should consider not using it.

Label

Human-friendly labels are better than IRIs for display purposes, both within the application and in the AP itself. A label may be recommended in the original context, which is the preferred label, but a local application can use a local label.

Sub-property-of

Sub-property-of would appear in the property's original context, where it is part of the syndetic structure. We cannot change this information in the AP; we can only repeat it. Care should be taken, when assembling the AP, that the hierarchically inherited properties of properties are respected.

Domain

Domain may or may not be asserted in the property's original context. If it is, we could simply repeat it in

our AP. Alternatively, we could narrow the domain to a subclass. We should not change the class so that it will result in instances that are no longer members of the class asserted in the original source. If the domain is not asserted in the original source, we are free to assert a domain in the AP, if required by the application.[10]

Range

Range may or may not be asserted in the property's original context. As with domain, we could choose to refine the range for our purposes, but not change it. If the range is not asserted in the original source, we are free to assert a range in the AP, if required by the application.[11]

Node Type

The node type concerns a property's value (whether it is an IRI, a blank node, or a literal); we'll describe it in the section on values. However, information about values in our APs is usually documented as part of the description of a property. Node type is a new addition to our APs. It may or may not be defined in the original context of the property.

"Use Values From" or "Lookup"

"Use values from" or "lookup" will likely appear in an AP only. It is more than an instruction to use headings from a controlled source; it states we should search a controlled vocabulary and, if a match is found for our heading, retrieve additional data from the external data set, especially the IRI. It also may include a search across multiple data sets. It involves new practices for us, and we still don't have a commonly used method to represent these practices in our APs.

Another change is that common practices for creating APs, which may become standards, are emerging. What distinguishes these new practices is the adoption of entities: entities can be specifically described, as well as the profile itself in some cases; nevertheless, the heart of these profiles remains the enumeration of properties used to describe a given entity. The Library of Congress led the way by creating the LC BIBFRAME Profiles specification, which describes a way to describe properties of properties, properties of entities, and properties of the profile itself for unlimited resources.[12] This was adopted and refined by the Linked Data for Production 2 project (LD4P2) for use in creating APs in Sinopia. DCMI is also working on new representations of APs, mostly through its DCMI Application Profiles Working Group, which includes some novel ways to describe and utilize properties of properties but in a familiar spreadsheet environment.[13] RDA requires a narrowing of its immense number of properties for use in specific applications, and APs will do this work; the RDA literature, notably the RDA Toolkit, includes some sample APs, but it is still unclear what the exact structure will be.[14] Whatever the case may be, a shared structure for RDA profiles would be extremely useful, especially if the structure is to be machine-actionable. Hopefully, PCC will contribute to several of these efforts.

All these efforts at the various leading organizations include the new RDF instruments as well as instruments more traditional to our APs, such as a property's cardinality. Even if we still plan only to create an AP that is a list of properties, we would do well to create APs that take these changes into account. AP creation was already a specialist endeavor that, with a little help, could be handed off to a nonspecialist. With the emergence of linked data, even that may prove difficult.

Values

With few exceptions, properties have values. In APs, the property-value pair is treated as a unit. Sometimes this unit is called a field. We can distinguish, sometimes with difficulty, a property of a property from a property of a value. For example, stating that a property is repeatable is a property of the property; stating that the property uses only terms from the Art and Architecture Thesaurus is a property of the value. Nevertheless, if we are creating a tabular AP, with each row describing a property, the properties of the value are described in the same row as the properties of the property.

Here we will endeavor to discuss the properties of values. Until recently, these properties were almost exclusively ways to create textual values ("strings"). This is still an important part of the properties of values, but with the introduction of linked-data practices, we have not only some new ways of describing string values, but also some new value properties.

Node type is a new property of values mentioned above. It is a springboard into many current issues, so we will take a close look at node type, and then, due to space constraints, take only a cursory look at some other properties of values.

A *node* is derived from the domain of graph data modeling; it comes to most library metadata professionals from RDF specifications, as RDF is a graph data model. It's simple: every thing or string is a node joined to another node by a property, also called a "relation," or an "arc." The RDF core structure is often represented as the "triple": (1) a *subject* node that can be described by a property or "arc" called (2) a *predicate* that has a value that is (3) the *object* node. Many of us are familiar with the RDF triple

subject-predicate-object, but most people we collaborate with will not be familiar with graphs and RDF, so talking about nodes may be a little confusing. The triple-subject and the triple-object are nodes.

When we talk about the node type, we are describing the triple-object. Although not all data for which we will want to write APs will be RDF data, it is useful to keep in mind that, when we describe values, we're describing the triple-object, the value of the predicate, the node toward which the arc in the graph is directed.

There are not many node types: IRI, blank node, literal, or some combination of the three. Distinguishing the node type is useful. An IRI and a blank node represent actual things, and things require additional modeling and description. If the node is a literal, then rules for entering the literal should be included in the AP. However, a literal is not a thing. It can be turned into a thing, like the RDA Nomen, but, as a string, it cannot be further described.

Of course an IRI is itself a text string distinguished because it is situated in larger data models. In the context of the World Wide Web, it is an actionable string that follows a particular syntax; the action is that it "dereferences" (the IRI is an IRI "reference" that references a resource on the web). This points to another complexity that makes AP authoring a task of specialists. Our APs are miniature models that require a handful of skills. These miniature models are part of larger models, such as OWL ontologies. Those models themselves are situated in a mega-model, RDF, which provides a common model for Semantic Web data. Then all of this is situated in a super-mega-model, the World Wide Web. Understanding the full stack is more than a full-time endeavor.

Stating that a value is an IRI has easily understood implications for our instance data. Any values to be entered by data creators for a given property should be IRIs; any values seen in a data set by data consumers can be recognized as IRIs. If sound linked-data practices are followed, the IRI represents a thing, and the data consumer should be able to dereference the IRI and retrieve useful information about the thing.

There are nodes that are things but are not represented by IRIs and are not text nodes; we call these *blank nodes* or *bnodes*. They can be described—statements can be made about them—but they are not given an identity on the web. As usual, it is more complicated than that: usually local identifiers are assigned (by any software used to parse the data), but these do not persist beyond the local context.

Stating that a value is a blank node has easily understood implications for our instance data. Any values to be entered by data creators for a given property should be a blank node and should not be assigned an IRI. This is not straightforward for our systems, however; blank nodes require systems that permit a layered data structure. Blank nodes result in data "nested" in other data. As writers of APs, we are not always at liberty to state that a value should be a blank node. It depends on our model developed for the overall application within the confines of a specific platform (our "implementation model").

The string is the most complicated of the three node types: it can be "structured," "unstructured," or an identifier; it can be "typed"; its language can be stated; it can be from a controlled vocabulary.

Structured and *unstructured* are terms taken from RDA to describe literals, but the problem they represent is not just an RDA problem. An unstructured literal is a blob of text entered however, without any rules; it is uncontrolled. A structured literal, on the other hand, follows data-entry rules; for example, the elements of the textual information may need to be entered in a particular order. The rule followed is called a *string encoding scheme* in RDA; elsewhere, most notably in DCMI, it is called a *syntax encoding scheme*. Nowadays it is common to call it an *SES*, and it is another property of values. In our AP, we would somehow state that the value of a given property is a literal and that the literal is either structured or unstructured; if structured, we would specify the SES.

RDA also distinguishes literals that are identifiers. Although these literals are similar to structured values, they are distinguished not only because they always have meaning in a particular external context, but also because they are considered machine-readable. Although not a common feature of APs outside RDA, identifier-literals will be essential to our RDA profiles (when we get around to writing them).

Another node type is the *typed* literal. Usually, in an AP, we state that the node type is a literal and that the data type is a particular data type. Because of the overuse of the word *type*, there is some confusion with this property of values. In this case, we're referring to traditional data type, such as string, date, dateTime, number, integer, and so on. When our instance data follows RDF, our typed literals are most frequently typed using the data type vocabulary in "W3C XML Schema Definition Language Part 2."[15] Our entry in the AP could look something like the fragment in figure 3.5.

When the node type is a literal, there is yet another variation: the literal with its language identified. This presents a particular problem for the AP; we would state that the node type is a literal, then state somehow (it is not a fixture of APs at present) what the language expectations are. It could be that a value must be in a particular language; that is easy to represent in the AP, say in the input instructions. It could be that the language of the value must be explicit in our instance data; that is more difficult: where do we enter that information? Our systems do not customarily allow us to attach a language to a value (see figure

Property: http://purl.org/dc/terms/date

Label: Date

Node Type: rdfs:Literal

Datatype: xsd:gYear

Input instructions: Enter the date created; enter the year only

Figure 3.5
Property/values for a date property plus property/values for values, including data type

Application Profile:
Property: Label
Node Type: Literal
Language: Greek, Ancient *[this is a property of the value and its value]*

Display 1:
Label: Ἀνδρομάχ

Display 2:
Label: Ἀνδρομάχ [Language: Greek, Ancient]

Stored as:
Label=47;
47.labelString: Ἀνδρομάχ
47.labelLanguage: GreekAncient

Figure 3.6
Difficulties making language of a value explicit

3.6).

This "meta" problem, which applies also to data type, is a structural complexity inherited partially from RDF. We want our systems to represent a value of a property of a value. These seemingly simple needs cause trouble at many levels. Where does the information go in the AP? Where in the data entry form? Should all values have a language requirement? Nevertheless, we widely acknowledge the importance of language identification and would do well to create data points for language in our APs and demand systems that feature elegant representation of language.

The last type of literal we will consider here is a literal value taken from a controlled vocabulary. The new term for a controlled vocabulary is *vocabulary encoding scheme* or VES. Ideally values from a VES would have a node type IRI, not literal. The result in most systems, unfortunately, would be a value that displays to users as an IRI. Surely users would prefer we enter the literal instead. In this case, the VES would be identified in our AP, but its IRIs would likely not appear in our instance data (see figure 3.7).

As seen above, other properties of values include the SES and the VES; also mentioned above were domain and range, as well as data type, which can

Property label: Subject

Node Type: literal

SES: LCSH at http://id.loc.gov/authorities/subjects.html

Input instructions: Enter the header exactly as it appears at the top of the page

Figure 3.7
AP fragment with instructions to use a string from the VES

be seen as properties of values. Sometimes we see "shape," which comes to us from RDF validation languages (SHACL and ShEx being the most prominent). Cardinality can be a property of values when a single property allows multiple values separated by a delimiter. Other properties of values might include

- length of the value
- choice of a value from a set
- intersections or unions of value sets
- constant values
- maximum/minimum
- maxInclusive/minInclusive
- base IRIs for IRI values
- patterns that values should follow (like regular expressions).

A lot of this may be present in the ontologies we use as sources for our APs, or we may add it to the APs ourselves. If it is present in the ontology, we would not want to contradict anything in the ontology.

Notes

1. Dublin Core Metadata Initiative, https://dublincore.org/.
2. Mikael Nilsson, Tom Baker, and Pete Johnston, "The Singapore Framework for Dublin Core Application Profiles," Dublin Core Metadata Initiative, January 14, 2008, https://dublincore.org/specifications/dublin-core/singapore-framework.
3. Unified Modeling Language, https://www.uml.org/.
4. IFLA Study Group on the Functional Requirements for Bibliographic Records, *Functional Requirements for Bibliographic Records*, IFLA Series on Bibliographic Control 19 (Munich, Germany: K. G. Saur Verlag, 1998, last updated February 2009) https://www.ifla.org/publications/functional-requirements-for-bibliographic-records; Pat Riva, Patrick Le Bœuf, and Maja Žumer, *IFLA Library Reference Model: A Conceptual Model for Bibliographic Information* (The Hague, Netherlands: International Federation of Library Associations and Institutions, August 2017, last updated December 2017), https://www.ifla.org/publications/node/11412.
5. Tom Heath and Christian Bizer, *Linked Data: Evolving the Web into a Global Data Space* (San Rafael, CA:

Morgan & Claypool, 2011), chap. 3, http://linkeddatabook.com/editions/1.0.
6. "PCC URIs in MARC Pilot," Program for Cooperative Cataloging, Library of Congress, last updated October 24, 2019, https://www.loc.gov/aba/pcc/pilots/URIs-in-MARC-Pilot.html.
7. Sinopia home page, Linked Data for Production 2 (LD4P2), https://sinopia.io.
8. "Task Groups," Program for Cooperative Cataloging, Library of Congress, https://www.loc.gov/aba/pcc/taskgroup/task-groups.html.
9. Riva, Le Bœuf, and Žumer, *IFLA Library Reference Model*, 17.
10. Dan Brickley and R. V. Guha, eds., "RDF Schema 1.1," section 3.2, W3C Recommendation, World Wide Web Consortium, February 25, 2014, https://www.w3.org/TR/2014/REC-rdf-schema-20140225/#ch_domain.
11. Dan Brickley and R. V. Guha, eds., "RDF Schema 1.1," section 3.1, W3C Recommendation, World Wide Web Consortium, February 25, 2014, https://www.w3.org/TR/2014/REC-rdf-schema-20140225/#ch_range.
12. "BIBFRAME Profiles: Introduction and Specification," draft for public review, Library of Congress, May 5, 2014, https://www.loc.gov/bibframe/docs/bibframe-profiles.html.
13. Dublin Core Metadata Initiative, "dcmi/dctap," GitHub, https://github.com/dcmi/dctap.
14. "Application profiles," RDA Toolkit, https://access.rdatoolkit.org/Guidance/Index?externalId=en-US_ala-591ca278-2807-399b-9530-6b44171e6ccc.
15. David Peterson, Shudi (Sandy) Gao, Ashok Malhotra, C. M. Sperberg-McQueen, and Henry S. Thompson, eds., "W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes," W3C Recommendation, World Wide Web Consortium, April 5, 2012, https://www.w3.org/TR/xmlschema11-2.