Designing an API-based protocol for the interoperability of textual resources

Pascal Belouin and Sean Wang

Max Planck Institute for the History of Science, Germany
Email: <FirstInitialLastName>@mpiwg-berlin.mpg.de

Abstract

Designing a protocol for the interoperability of digital textual resources—or, more simply, a "IIIF for texts"—remains a challenge, as such a protocol must cater to their vastly heterogenous formats, structures, languages, text encodings and metadata. There have been many attempts to propose a standard for textual resource interoperability, from the ubiquitous Text Encoding Initiative (TEI) format to more recent proposals like the Distributed Text Services (DTS) protocol. In this paper, we critically survey these attempts and introduce our proposal called SHINE, which aims to escape from TEI's legacy and prioritize instead the ease for software developers to representation and exchange textual resources and their associated metadata. We do so by combining a hierarchical model of textual structure with a flexible metadata scheme in SHINE, and we continue to define and develop it based on user-centered and iterative design principles. Therefore, we argue that SHINE is a protocol for textual interoperability that successfully balances flexibility of resource representation, consistency across resource representation, and overall simplicity of implementation.

1. Introduction

Internet and the digitization of sources profoundly changed the research process in the humanities. Researchers in text-centric disciplines like history and literature now access primary and secondary sources from different providers online in various formats (such as web pages, PDF files, Microsoft Word documents, or TEI documents). While many could be downloaded and saved locally (or on the cloud), others are locked in online read-only platforms. This diversity in both document formats and access avenues (including differing licenses) creates complications for scholars interested in computational and other large-scale analyses across texts from multiple providers. For instance, digital research tools for textual analyses, such as tagging or entity recognition tools, must ingest texts in specific formats usually by manual uploading documents. Besides technical work to pre-process the texts, many sources are covered by licenses that might make such upload difficult. Thus, while many sources and tools are already digital, researchers cannot easily combine and work across them: the varieties of formats and access rights necessitate significant manual curation; furthermore, certain licenses (especially on sources from commercial providers) make more-than-read analyses legally impossible.

To improve interoperability between digital research tools and textual resources on the web, many have proposed different legal and technical solutions, as well as large infrastructural projects that address a combination of both. As we summarize elsewhere (Wang, Belouin, Ho, & Chen, 2019), with few exceptions these proposals tend to bridge the gap by (re-)centralizing select resources and tools in a contained research environment. Since acquiring text-mining or similar more-than-

read licenses from commercial providers can be expensive, such proposals require strong financial backers and are difficult to implement across institutions. Their centralization also limit flexibility for researchers and make reproducibility more challenging.

Taking a distributed (or decentralized) approach that is inspired by the success of International Image Interoperability Framework (IIIF) as an interoperable protocol for image-based research (Babcock & Di Cresce, 2019), we have designed and implemented a research infrastructure for textual exchange called Research Infrastructure for the Study of Eurasia (RISE).[1] We designed RISE to address these aforementioned issues in a pragmatic manner by combining a robust authentication and authorization mechanism and clearinghouse implemented in a middleware, with a relational state transfer (REST)-like application programming interface (API) text exchange protocol (which we call SHINE) that represents textual resources and their metadata in a flexible manner. While RISE was initially conceived to address specific challenges related to working with commercial textual resources in Chinese studies (Wang, Belouin, Chen, & Ho, 2018), its technical design works with textual resources regardless of language and we believe SHINE can do for textual resources analogous to what IIIF already did for online image collections.

In this article, we focus primarily on the issue of designing a protocol for textual exchange and interoperability. First, we survey past proposals and current attempts for designing such a protocol to highlight some of the key requirements. We only focus on attempts that cover all three aspects of textual exchange and interoperability: discoverability, referencing, and interoperability of the actual resource contents; therefore, protocols that focus on discoverability or metadata only (such as OAI-PMH) are excluded. Then we describe in detail our attempt (which is SHINE) and the rationale behind our design, which focuses on providing a generic and flexible representation for both structure and metadata that would cover the heterogeneity of textual resources. We provide some practical examples to show how various textual resources could be represented in SHINE. Finally, we critically assess the future of interoperable text-based digital research and, considering past failures (Dombrowski, 2014), the role research infrastructures could play in the ecosystem.

2. Existing protocols for representing and exchanging textual resources

Various attempts have been made over the years to facilitate the interoperable exchange of textual resources since as early as 1987 (which predates the World Wide Web). The Text Encoding Initiative (TEI), initiated that year, is an early attempt and has since been used by many projects, especially in literary studies. TEI guidelines have evolved and increased significantly in complexity; they are currently at their fifth iteration.[2] Although most research projects today do not implement TEI with this as their main purpose, an early goal of TEI's was to "provide a standard format for data interchange in humanities research."[3] As Burnard (2013) noted, in an alternate universe we might have a more expansive TEI called "Text Encoding for Interchange." However, this ambition has fallen away and TEI instead focuses primarily on in-text markup rather than as a standard protocol for interoperable textual exchange. Therefore, in practice, any research software that automatically ingests TEI documents will have to confront this non-interoperability (Schmidt, 2014). Furthermore, since TEI's inception predates the concept of web services, such use cases were not considered in its native design.

To mitigate TEI's shortcomings with regards to interoperability on the web, more recent solutions have been proposed to allow for discoverability, referencing, and interoperability of textual resources (either in the TEI format or in plain text). The most historically significant effort in this direction is the Canonical Text Services protocol (CTS). Originally developed in 2010 as part of the Homer Multitext Project, CTS is a protocol designed to serve TEI texts,[4] and it allows well-written software clients to pull these texts from the Homer Multitext Project and, subsequently, from the Perseus Digital Library.

Digital historians who automatically consume texts from the Perseus Digital Library have identified lingering issues with CTS. For example, CTS relies on a Uniform Resource Name (URN) scheme, which can be slightly inconsistent across resources made available by the Perseus Digital Library.[5] Clérice (2018) argued that one of CTS's main issues is the fact that it is "tightly coupled to text identifier syntax", which entails a number of drawbacks such as a lack of pagination for catalogs. Furthermore, CTS was initially developed by digital classicists as a project-specific standard, which makes wider adoption more challenging; Despite these shortcomings, CTS remains in use by Perseus and a few other projects such as Paraphrasis.org, but its uptake as a standard is limited.

A more recent development in the quest to create a protocol facilitating textual resource interoperability is the Distributed Text Services (DTS) initiative. DTS started in 2015 aiming explicitly to create a "IIIF for text":

> The Distributed Text Services effort has been inspired, informed and influenced by the Canonical Text Services protocol (CTS). CTS has allowed many classical, canonical texts encoded in TEI to be made available in a machine-actionable, linked open data fashion. However, the CTS API is tightly coupled to the CTS URN identifier system which does not support citation systems used by more modern content or other forms of writing, such as papyri or inscriptions. The API also does not adhere to modern community standards for Web APIs.[6]

In response, DTS adopts specifications of the Hydra Core Vocabulary, which combines the REST architectural style and Linked Data principles "to provide a vocabulary which enables a server to advertise valid state transitions."[7] An API compatible with DTS provides three operation endpoints:

1. Collections Endpoint for navigating the text collection contents;
2. Navigation Endpoint for navigating within a single text document; and
3. Documents Endpoint to retrieve complete or partial texts.

The first two endpoints return JSON-LD (adhering to Hydra), and the third returns TEI-XML of the requested text or fragment. This results in a protocol that adheres strictly to the two most influential principles for modern web service design, REST and "hypermedia as the engine of application state" (HATEOAS; Fielding, 2000), and arguably makes the DTS protocol future-proof, self-documented, and sufficiently flexible to represent any type of catalog of textual sources. Furthermore, its use of JSON-LD, a format increasingly popular among digital humanists, is

another strong point. Therefore, DTS is gaining more traction in the digital humanities community and has a better chance than all previous attempts to become the "IIIF for text" it aims to be.

Given our positive assessment, it begs the question why we developed separately another protocol for textual resource interoperability rather than adopting DTS. There are several reasons for this decision. The first is contextual. As we described elsewhere (Wang et al., 2018), we initially developed RISE as a distributed research infrastructure for Chinese studies and, as the necessity of an interoperable protocol became obvious, designed SHINE accordingly. This contextual legacy provided a certain degree of freedom; since we tackled the problem of textual representation and exchange from the ground up and focused in the first instance only on structured plain text, quite serendipitously we were able to avoid TEI's specter.

Our specific origin also resulted in important technical differences from DTS. Since RISE focuses on improving linkages between distributed textual collections and digital research tools, SHINE's initial design focused on this specific type of interoperability and, accordingly, the corresponding target users who are not humanities researchers but are instead software developers. Our focus on the ease of implementation for developers (especially of digital research tools that consume textual resources) distinguishes SHINE from other attempts and results in more pragmatic design choices. In our opinion, software development practices usually develop organically and iteratively; they are informed by, but not fully adherent to, academic design and best practices. Indeed, the history of computer science is full of great ideas and concepts that were only marginally adopted by software development practitioners, such as the OSI model[8] and the Semantic Web.[9] Unfortunately, the strict constraints and principles from HATEOAS and pure RESTful web services are, at least for the time being, victims of the same fate. Practitioners have argued, for example, that "when designing a hypermedia API, you're really designing for a client that does not, and will never, exist";[10] one even went as far as calling HATEOAS "useless."[11] Cognizant of this history, we decided to design SHINE in a "REST-like" manner to better facilitate adoption and implementation by both resource providers and research tool developers.

In practice, this "REST-like" manner for developing APIs is already common. Most APIs by software development practitioners now consist of a set of routes that allows a client to perform "Create, Read, Update, and Delete" (or CRUD) operations on domain entities of a particular web service or application.[12] Based on user-centered design principles (Gulliksen et al., 2003), SHINE includes a rather small number of entities that represent the structure of a textual resource, to which any type of metadata properties can be attached. These entities are then exposed by API routes which permit CRUD operations along the common industry guidelines (Hansson, 2006). We also made a distinction between the act of "cataloging" textual resources and the simple representation of resources in terms of structure, content, and metadata.

In summary, although our goals are similar, SHINE and DTS represent resources and catalogs very differently in the API routes, and this difference stems primarily from development contexts. In designing SHINE, we have decided to sacrifice some strict adherence for a better balance between academic priorities and ease of uptake. SHINE and DTS remain complementary, however. As we design and develop SHINE in an agile, iterative manner, we have over time adopted DTS's many positive features, particularly its extensive use of JSON-LD to describe resources and the API scheme itself. One of us (Belouin) is also an observing member of the DTS Technical Committee,

and there is an API adaptor (in beta version) that can pull resources via DTS and make them available via SHINE endpoints (and vice versa). Given this complementarity, in the next section we describe SHINE in detail, focusing specifically on how it models and represents textual resources.

3. Overview of the SHINE protocol

Given our specific focus and from surveying existing attempts, we identified three criteria for evaluating a protocol for text interoperability: (1) flexibility of resource representation; (2) consistency across resource representation; and (3) overall simplicity of the protocol. These three criteria stand in tension, and in designing SHINE we have strived to achieve a balance among the three (with some trade-offs from each), rather than privileging one and sacrificing the others.

We also identified the following minimum aspects of textual resources that a protocol for interoperability must be able to model. First, a discrete textual resource (e.g., a book) is usually composed of hierarchical sub-parts down to the level of a single word; we refer to a resource's internal hierarchy as its "structure." Textual resource structures can vary greatly between genres (e.g., compare a newspaper article in 20th century England to a measured poem from the 7th century in Arabic), and a protocol must be flexible enough to represent these different structures. Second, a resource and its hierarchical sub-parts might have different properties or attributes, such as the language they are written in, their author, or other information relevant to a particular humanities discipline. One could also consider annotations made by a particular researcher about (a part of) a textual resource as an attribute. We refer to these types of information about a resource and its sub-parts "metadata." While some information might be considered as both structure and metadata (as seen in some TEI markups), we enforce clear distinction between these two types of information, as well as the "content" of a textual resource (i.e. plain texts), in SHINE.

Based on the above, we devised three main principles to guide the design of the SHINE protocol:

1. SHINE should be able to represent, with a limited number of entities, the structure of any genre of textual resources;
2. SHINE should be able to represent any type of metadata, whether associated with a textual resource or any of its sub-parts; and
3. SHINE should be simple enough so that any developer familiar with current industry standards in web services can implement it easily.

The SHINE protocol, in its current form, conform to these principles and results from an iterative design process where the resource representation model was continually refined as more resource providers and research tools developers adopted SHINE. We favor this pragmatic approach, as opposed to having researchers dictate design elements, in order to have SHINE closely reflect actual software development needs and encourage wider adoption.

In a nutshell, the SHINE protocol consists of several REST-like routes that expose structural components of a textual resource; in addition, we also defined "collection" to group multiple resources. Thus, the building blocks used to represent structure in SHINE are, from the least to most granular, "collection," "resource," "section" (parts of a text, such as chapters or verses), and

"content unit" (the smallest unit of text, which could be a page, a line, or even a word depending on the resource genre). Sections can have multiple hierarchical levels, using a mechanism through which sections can be the children of another section, so that genres with more complex hierarchies (e.g., a novel with chapters, sub-chapter sections, and paragraphs) could still be represented as flat arrays using regular REST-like routes.

Table 1 shows the minimum set of API routes that a resource provider must implement to be SHINE-compatible. These seven routes allow any client to pull the structure, metadata, and content of the textual resources made available by a resource provider. Although the routes shown here are read-only, additional routes can also be implemented to allow POST, PUT, and DELETE requests. Thus, mapping this REST-like API structure to CRUD operations is straight-forward, and a resource provider can implement a software client to manage these additional operations.

Table 1. The minimum set of API routes for a SHINE-compatible resource provider

HTTP Verb & Route	**Description**
GET /collections/	Returns all collections
GET /collections/*{uuid}*/	Returns the metadata for a collection
GET /collections/*{uuid}*/resources	Returns all resources for a collection
GET /resources/*{uuid}*/	Returns the metadata for a resource
GET /resources/*{uuid}*/sections	Returns the sections for a resource
GET /sections/*{uuid}*/	Returns the metadata for a section
GET /sections/*{uuid}*/content_units	Returns the content units for a section
GET /content_units/*{uuid}*/	Returns the metadata for a content unit

Each of these structural entities, or building blocks, can have metadata attached to them. The SHINE protocol represents, stores, and exchanges metadata as data objects that consist of attribute-value pairs and array data types, modeled after the JSON open standard (see Figure 1). Metadata fields are grouped by namespace, and we plan to make SHINE's metadata schema fully JSON-LD compliant in the next release.

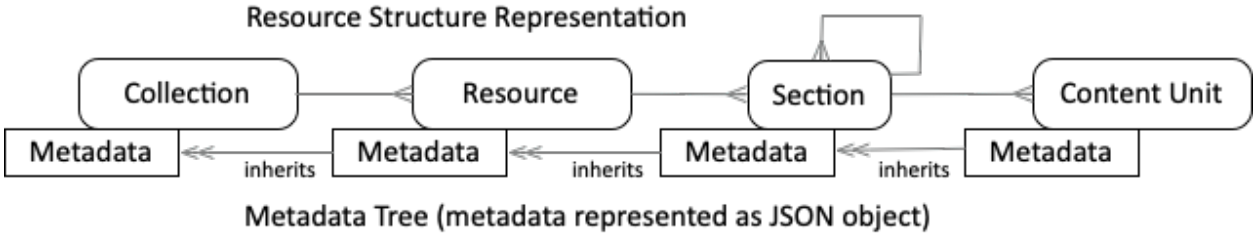

Figure 1. A schematic representation of the SHINE resource representation model.

Furthermore, as illustrated in Figure 2, these metadata objects are inherited from the higher structural entity in the hierarchy of a resource; however, a particular metadata field can be overridden at a lower structural entity. This metadata model featuring inheritance and override mechanisms provides flexibility, and we explore its full implication in the next section.

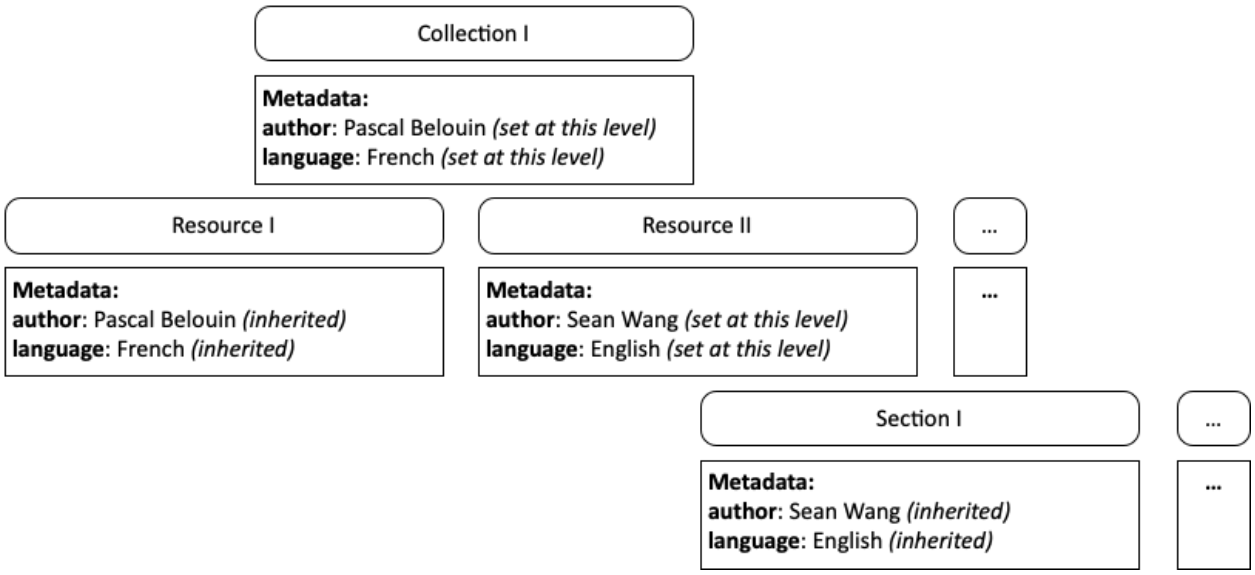

Figure 2. A simplified example of SHINE's metadata inheritance and override mechanism

3.1 Resource modeling

SHINE's modeling of textual resources is generic and flexible, as it allows for any granularity for both structural and metadata representation. In general, the strict hierarchical representation of structure gives SHINE allows for generic representation across genres, while the metadata inheritance and override mechanisms based on that hierarchy gives SHINE its flexibility. We demonstrate below several examples on how to work with this combination of structure and metadata to model multi-lingual resources in SHINE with ease.

Take a collection of Homer's classics, composed of various versions of *The Iliad* and *The Odyssey* (some of which were translated or commented on) as an example. Figure 3 shows how this collection can be modeled in SHINE at higher levels of the structural hierarchy (collection and resources), while Figure 4 shows the same collection at lower levels using one resource—the commented version of *The Odyssey*—and its sections and content units. We implement a basic set of Dublin Core fields in SHINE and fetch them from the resource providers, but any additional metadata information beyond that basic set is still preserved. SHINE's metadata inheritance and override mechanisms work hierarchically alongside the resource structure. Any metadata information set at a higher structural level (e.g., collection) will be automatically inherited by lower levels; for example, once the creator (Homer) is set at the collection level, then all lower levels (resource, section, and content unit) in that collection will inherit the same metadata information. The same process is done with the language (Greek). However, one resource in this collection is an English translation of *The Iliad*. For this particular resource, we can override the inherited language metadata from the collection level (Greek) by changing it to English. This override will not impact any higher level, but all lower levels of this resource (sections and content units) will inherit the overridden language metadata of English.

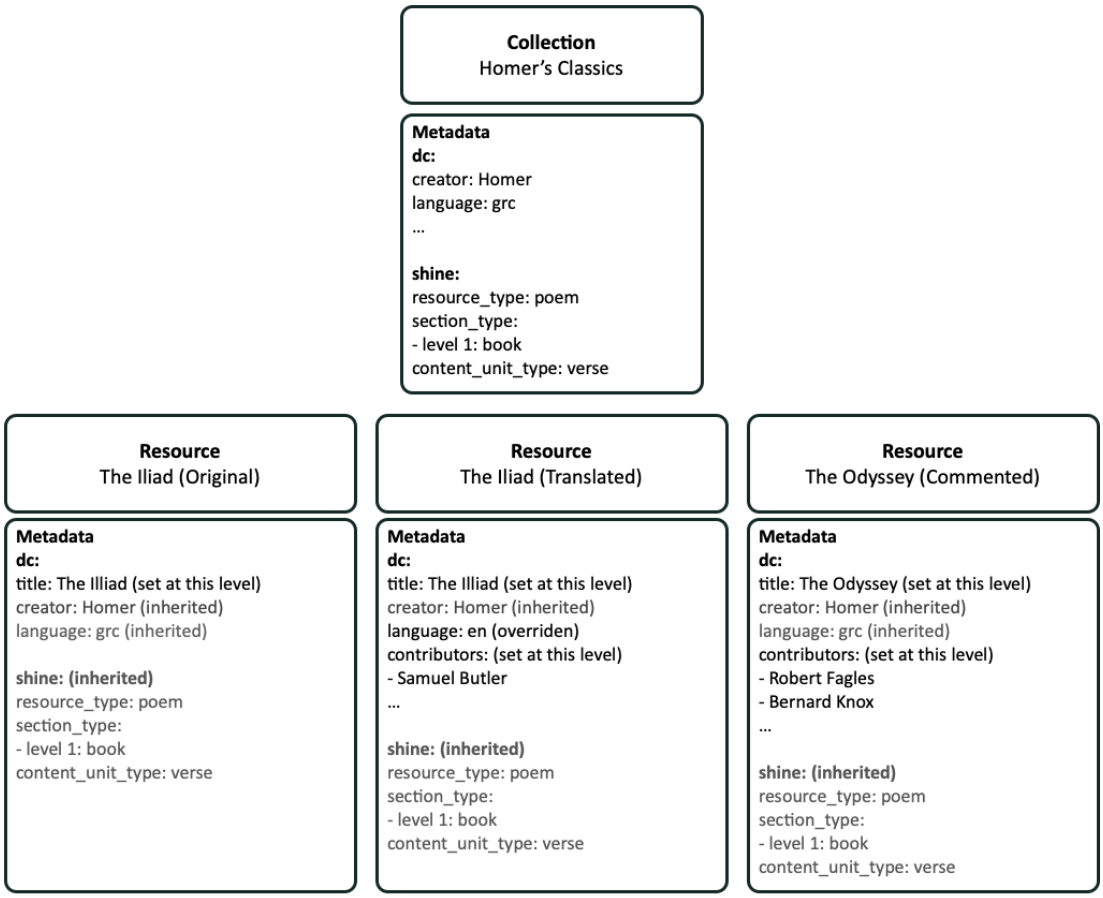

Figure 3. Representing a Homer's classics collection and its resources using SHINE

SHINE's metadata can also be used to store and represent information about how a particular structural entity maps onto specific genres (e.g., poems, paragraphs); this information can be set under the "shine" namespace of the metadata object. Figure 4 shows an example using The Odyssey, where "section type" and "content unit type" are set. Since the structural entity "section" can have multiple hierarchical levels in SHINE (i.e., a section could have a parent section or children section[s]), in some cases it is necessary to describe the type of section level in this metadata namespace.

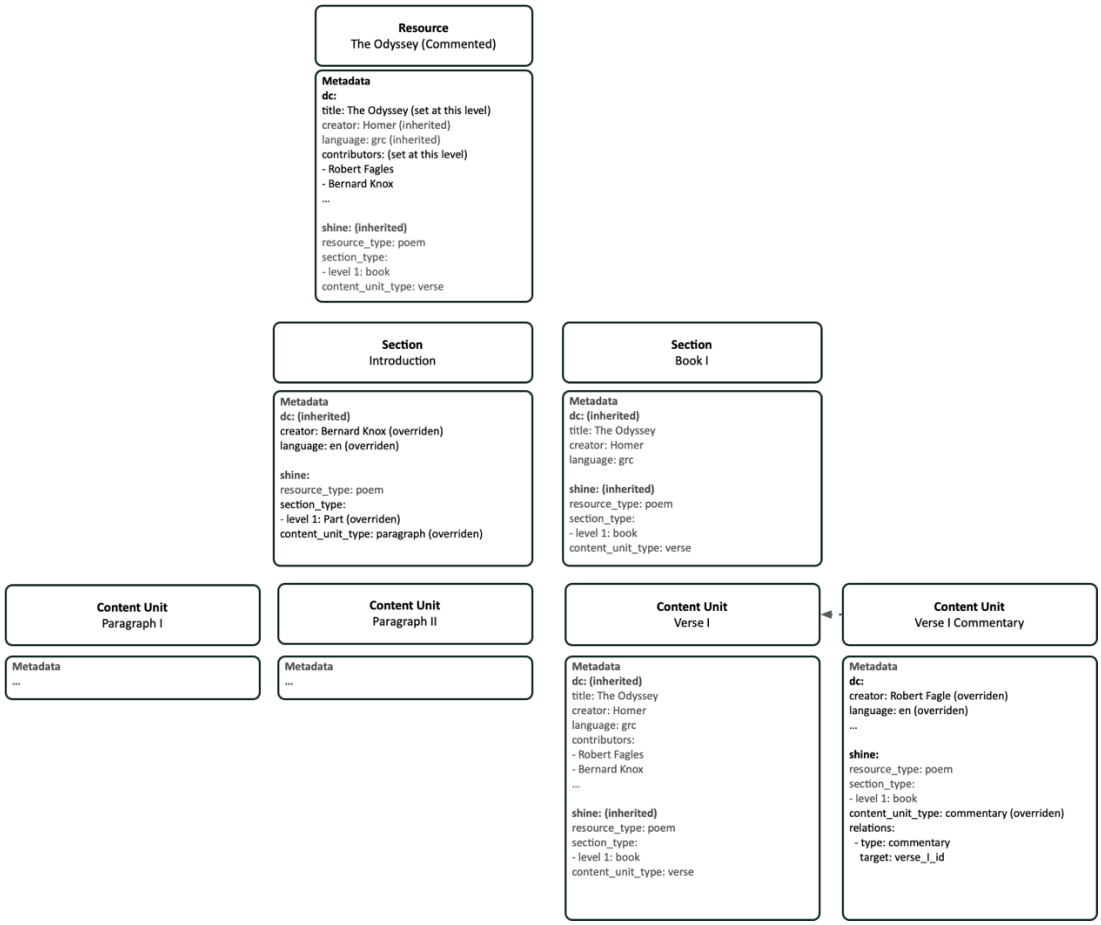

Figure 4. Representing various sections, content units, and their metadata using SHINE

3.2 Filtering and search

Filtering of resources becomes instinctively straight-forward given the REST-like routes provided by the SHINE protocol. Filtering based on a particular metadata field, for example, can be easily done by passing a particular filter string as a parameter to one of the collection routes, allowing for the filtering of collections, resources, or sections. Furthermore, we are currently working on defining a set of additional API endpoints and mechanisms to facilitate more advanced search based on metadata attributes, as well as full-text search where allowed.

3.3 Cataloging

Cataloging at the collection level could either be seen as a structural issue or a filtering based on some metadata information issue. We see many resource providers organize their resource catalog as the former. If we continue with the Homer's classics example above, a resource provider could create a collection called "Homer's classics," catalog various resources into this collection, and organize them in hierarchical trees. Based on the metadata information attached to them, resources could be grouped by theme, language, edition, etc., within this collection. DTS, for instance, employs this sophisticated method of cataloging that makes top-level collection a first-class citizen.

SHINE, on the other hand, employs the latter method and thus only permits one level of collection. We decided to sacrifice some theoretical sophistication for the sake of simple implementation and integration with authorization mechanisms on the collection routes. Said more simply, structurally speaking it is not possible to have a collection of collections in SHINE. Thus, grouping collections in SHINE must be done by filtering some metadata attributes, as opposed to collections being used as a complex structural entity. For example, imagine that in addition to Homer's classics, two other collections are represented in SHINE: Jane Austen's bibliography and *La Comédie humaine*. The latter consists of plain texts made available by Project Gutenberg, while the other two were published as collections by Penguin. It is not possible to have a collection of all collections published by Penguin on a structural level. But it is possible to do this *ad hoc* by filtering based on the metadata attribute "publisher: Penguin" on all collections.

3.4 Expanding the metadata object

Since the metadata object in SHINE is—in theory—infinitely expandable, we can use it for any type of description of a resource or its sub-parts. Thus, metadata can not only be used to store more conventional metadata information, but could also include non-standard information beyond the plain text version of content (such as annotations and research data). Recall that in SHINE, we strictly enforce the distinction between structure and metadata in our resource modeling. This distinction works well with interoperability of plain texts but not so much for non-standard and in-text markups like in TEI. For those who are interested, we can utilize the full flexibility of SHINE's metadata scheme to do these TEI-like actions. Therefore, it is possible to, say, have one resource reference another, or parts of a resource to reference another part, internally or externally, in SHINE. This would allow us to model entities such as commentaries, as seen in Figure 4, where a relations namespace is used in the lower right corner.

3.5 Access rights management

Textual resources are covered by different licenses and resource providers might impose additional restrictions on their access rights. Therefore, in some cases resource access must be moderated. An authentication and authorization mechanism can be implemented relatively easily on top of SHINE's REST-like routes at any level of structural granularity. Indeed, we implemented access rights management in our own hosted middleware instance. One issue we have not resolved, however, is that more-than-read analyses often require transfer of texts. If the texts have very stringent restrictions on sharing, in the long-term we may have to implement some sort of encryption mechanism to fully comply with these kinds of restrictions.

4. Conclusion

The SHINE protocol described in this paper is a second version, and we are implementing it in the various infrastructural components of RISE. RISE currently links 55,128 resources to an increasing number of research tools. We are refactoring the SHINE protocol to make it fully JSON-LD compliant. Besides plain texts, we are also working on mappers that would allow interoperability and transfer of TEI (and other XML-based textual data) through HTTP content negotiation.

In this design and implementation process, we have solicited feedback from many stakeholders who work with digital texts and research infrastructures. As we advocate for a user-centered design, additional feedback from the community is always appreciated to help us refine our technical products iteratively. At the moment, we are especially looking for library partners. Our survey also shows that groups such as DTS have similar goals as us, though our designs and scopes of implementation might be different. We have made SHINE-DTS mappers, and in our ongoing work with RISE it is possible that—depending on community uptake—we eventually replace SHINE with DTS entirely. Regardless of what might happen, we hope that the approach proposed here will shape the design of existing and future protocols so that we can together realize a "IIIF for text" within the digital humanities community.

---

[1] For more information, see project websites https://www.mpiwg-berlin.mpg.de/research/projects/rise-and-shine-research-infrastructure-study-eurasia and https://rise.mpiwg-berlin.mpg.de/.
[2] See https://tei-c.org/guidelines/p5/.
[3] Retrieved from https://tei-c.org/Vault/ED/edp01.htm#b2b1b3b3b3.
[4] See http://cite-architecture.github.io/cts_spec/.
[5] See http://inlustre.net/2013/04/how-to-retrieve-ancient-text-data-from-perseus/.
[6] Retrieved from https://distributed-text-services.github.io/specifications/.
[7] Retrieved from http://www.hydra-cg.com/spec/latest/core/.
[8] See https://spectrum.ieee.org/tech-history/cyberspace/osi-the-internet-that-wasnt.
[9] See https://twobithistory.org/2018/05/27/semantic-web.html.
[10] Retrieved from https://jeffknupp.com/blog/2014/06/03/why-i-hate-hateoas//.
[11] Retrieved from https://medium.com/@andreasreiser94/why-hateoas-is-useless-and-what-that-means-for-rest-a65194471bc8.
[12] See http://spec.openapis.org/oas/v3.0.2 for an example of this type of API commonly implemented for web services.

References

Babcock, K., & Di Cresce, R. (2019). Impact of International Image Interoperability Framework (IIIF) on digital repositories. In K. J. Varnum (Ed.), *New top technologies every librarian needs to know* (pp. 181-196). Chicago: ALA Neal-Schuman.

Burnard, L. (2013). The evolution of the Text Encoding Initiative: From research project to research infrastructure. *Journal of the Text Encoding Initiative, 5*. https://doi.org/10.4000/jtei.811

Clérice, T. (2018). *From file interoperability to service interoperability: The Distributed Text Services*. Paper presented at the annual conference of the Text Encoding Initiative, Tokyo. Retrieved from https://hal.archives-ouvertes.fr/hal-02196659/

Dombrowski, Q. (2014). What ever happened to Project Bamboo? *Literary and Linguistic Computing, 29*(3), 326-339.

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* (Unpublished doctoral dissertation). University of California, Irvine. Retrieved from https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Gulliksen, J., Göransson, B., Boivie, I., Blomkvist, S., Persson, J., & Cajander, Å. (2003). Key principles for user-centred systems design. *Behaviour & Information Technology*, *22*(6), 397-409.

Hansson, D. H. (2006). *Resources on Rails*. Keynote presented at RailsConf, Chicago. Retrieved from https://youtu.be/GFhoSMD6idk

Schmidt, D. (2014). Towards an interoperable digital scholarly edition. *Journal of the Text Encoding Initiative, 7*. https://doi.org/10.4000/jtei.979

Wang, S., Belouin, P., Chen, S.-P., & Ho, H. I. (2018). *Research Infrastructure for the Study of Eurasia (RISE): Towards a flexible and distributed digital infrastructure for resource access via standardized APIs and metadata.* Paper presented at the 9th International Conference of Digital Archives and Digital Humanities, Taipei. Retrieved from https://pure.mpg.de/pubman/faces/ViewItemOverviewPage.jsp?itemId=item_3033461

Wang, S., Belouin, P., Ho, H. I., & Chen, S.-P. (2019). *RISE and SHINE: A modular and decentralized approach for interoperability between textual collections and digital research tools.* Paper presented at the annual Digital Humanities Conference, Utrecht. Retrieved from https://dev.clariah.nl/files/dh2019/boa/0607.html