bf: BIBFRAME

2019 BIBFRAME Workshop - Stockholm, Sweden

https://ld4p.github.io/bf-workshop-2019/lde-edit-rdf/

# Sidestepping the graph - Sinopia Linked Data Editor's approach for editing RDF

## Background

The Sinopia's public facing linked data editor, available at https://sinopia.io/, constructs forms for creating and editing RDF based on resource templates' properties defined in the Library of Congress derived Profiles. Using JSON Schema validation that is versioned and available at https://github.com/LD4P/sinopia/tree/master/schemas.

The editor's use of a more modern Javascript React user interface library coupled with the Redux library for application-state management allows for the dynamic creation of valid RDF triples that are then saved through an API call to the Linked Data Platform Trellis. This approach simplified the implementation of the editor by eliminating the need for complex SPARQL statements for querying and updating a RDF triplestore.

## Profiles and Resource Templates

Sinopia generates HTML forms for creating and editing linked data that extends the Library of Congress Profiles used in the BIBFRAME Editor and Profile Editor projects. Profiles, as implemented in the BIBFRAME Editor, are JSON files that contain one or more resource templates.

Profiles also contain metadata that is not persisted within the Trellis but is still validated using JSON Schema when a Profile is uploaded in Sinopia's linked data editor. Defining and testing these Profiles across the different Sinopia cohort institutions and organizations is a community-lead collaborative effort with the cohorts requirements and suggestions driving the development priorities of the Sinopia Development team.

Here is a snippet of a Profile with metadata like **id**, **title**, **description**, and a Sinopia specific **schema** field:

```
{
    "Profile": {
        "resourceTemplates": [
           .
           .
           .
        ],
        "id": "ld4p:profile:bf2:Item",
        "title": "LD4P BIBFRAME 2.0 Item",
        "description": "Item for all formats (testing), based o
        "date": "2019-08-19",
        "author": "LD4P",
        "schema": "https://ld4p.github.io/sinopia/schemas/0.2.0
}
```

### Resource Templates

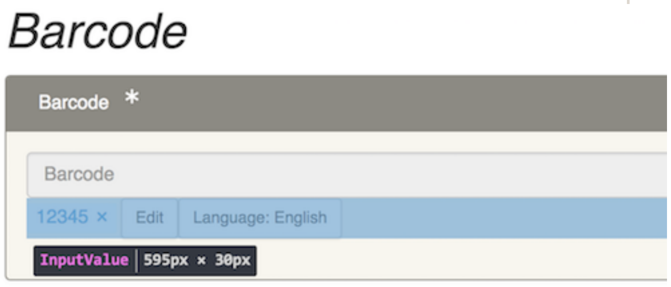
Each Profile contains one or more resource templates with the resource template including an identifier, information on who

## React

An open-source project sponsored by Facebook, React is a very popular Javascript module for building user interfaces. Early on, Sinopia adopted React as a way to dynamically generate the HTML elements for creating and editing linked data.

### Components

Most of the React components in Sinopia are pure functions that either generate HTML elements, css classes, and behavior or provide a collection-level container for other React components. For example the `InputValue` component, pictured below is an example of a **literal** component that is **mandatory**, not **repeatable**, and has a default value:



The source code for this component is available at https://github.com/LD4P/sinopia_editor/blob/master/src/components/

In this code snippet from that Javascript module, the `InputValue` component is defined as a `const` type variable with an important data structure `props` that are properties of the component. The next two lines set two constants, `isLiteral` and `label` that are themselves one-line functions that return conditional values when the component is rendered in the client web browser. Similarly, the `const handleEditClick` wraps two function calls that change the language and remove an item.

```
const InputValue = (props) => {
  const isLiteral = typeof props.item.content !== 'ur
  const label = isLiteral ? props.item.content : prop

  const handleEditClick = () => {
    props.handleEdit(label, props.item.lang)
    props.removeItem(props.reduxPath)
  }
```

Next these functions are tied and rendered in HTML with the **return** statement below:

```
  return (<div id="userInput">
    <div
      className="rbt-token rbt-token-removeable">
      {label}
      <button
```

created the resource template, when it was created, a description, a URI used to create a triple for a RDF type predicate, the JSON schema to use for validating, and a list of property templates.

```
{
  "propertyTemplates": [
   .
   .
   .
  ],
  "id": "ld4p:RT:bf2:Item:Use",
  "resourceLabel": "Use or Reproduction Policy",
  "resourceURI": "http://id.loc.gov/ontologies/bibframe/UsePoli
  "author": "LD4P",
  "date": "2019-06-11",
  "schema": "https://ld4p.github.io/sinopia/schemas/0.2.0/resource-template.json"
}
```

## Property templates

Each resource template must have at least one property template. A property template contains fields that determine if the property in the UI is **mandatory** or **repeatable**, with the **propertyURI** field used to determine the predicate for one or more triples. The property template can have default literal or URI values as well. The property template also contains a **type** property for determining what eventual [React][REACT] component uses to construct the editor UI.

### Literal Type Property

The most basic type of component is the Literal, that allows the cataloger to add a literal value in the object position for the RDF triple. The subject is either a URI or a blank node and is determined by the context in which the resource template is used.

Example **Literal** type property template:

```
    {
      "propertyTemplates": [
        {
          "mandatory": "false",
          "repeatable": "true",
          "type": "literal",
          "propertyURI": "http://id.loc.gov/ontologies/b
          "propertyLabel": "Your cataloger ID",
          "resourceTemplates": [],
          "valueConstraint": {
            "valueTemplateRefs": [],
            "useValuesFrom": [],
            "defaults": []
          }
        }
      ]
  }
```

### Lookup Type Property

Sinopia has a three different types of lookup components depending on the source for the lookup.

#### Library of Congress Lookup

```
{
    "mandatory": "false",
    "repeatable": "true",
    "type": "lookup",
    "valueConstraint": {
        "useValuesFrom": [
            "http://id.loc.gov/vocabulary/mstatus"
        ],
        "valueDataType": {
            "dataTypeURI": "http://id.loc.gov/ontologies
```

```
          onClick={() => props.removeItem(props.reduxPa
          className="close rbt-close rbt-token-remove-b
          <span aria-hidden="true">×</span>
        </button>
      </div>
      <button
        id="editItem"
        onClick={handleEditClick}
        className="btn btn-sm btn-literal btn-default">
        Edit
      </button>
      { isLiteral ? (<LanguageButton reduxPath={props.r
    </div>)
}
```

## Redux

The transformation the Profiles with Resource Templates being rendered with a client-side editor is accomplished with React but we still need a way associate all of changes made by the catalogers so that we can do such things as generation of RDF validations, and updating the backend Sinopia Server.

To capture the current data of the application's React components and to build a RDF representation based on the values of the components, the Javascript Redux project was used for managing the current state of the Sinopia editor application running in the client web browser of the user. The Redux state in Sinopia is a Javascript object that includes a number of top-level properties that store different aspects of the application. For example, loading the **resourceTemplate:bf2:Identifiers:Barcode** resource template, creates an **editor** property that reflects general state of the application like errors, validations, what fields have been expanded, what modals are displayed, and a checksum for alerting the user if data has changed but not saved back to the Sinopia server.

```
   editor: {
      displayValidations: false,
      errors: [],
      rdfPreview: {
        show: false
      },
      resourceURIMessage: {
        show: false
      },
      groupChoice: {
        show: false
      },
      expanded: {
        resource: {
          'resourceTemplate:bf2:Identifiers:Barcode':
            'http://www.w3.org/1999/02/22-rdf-syntax-
              expanded: true
            }
          }
        }
      },
      resourceValidationErrors: {},
      lastSaveChecksum: '54527c024d0021784f666c279485
    }
```

Each of component in Sinopia has a prop `reduxPath` that is an array made-up of URIs, resource template IDs, and random IDs, that is used to locate the values of the component in within a hierarchy representing the entire state of the application.

Continuing the example above, the barcode resource template is loaded into a property panel with the following *reduxPath*:
`['resource', 'resourceTemplate:bf2:Identifiers:Barcode',`

```
        }
    },
    "propertyURI": "http://id.loc.gov/ontologies/bibfran
    "propertyLabel": "Incorrect, Invalid or Canceled?"
}
```

### Questioning Authority Lookup

[Questioning Authority](#) is a service from Cornell University and the University of Iowa LIS program that provides an API service that Sinopia queries and either JSON or RDF is returned payload. QA caches all of the RDF from ShareVDE along with other sources like Discogs and some [id.loc.gov](#) linked data service like LCSH.

```
{
  "propertyLabel": "Related Discogs Entity",
  "propertyURI": "http://exampleontology.com/relatedDisc
  "mandatory": "true",
  "repeatable": "true",
  "type": "lookup",
  "subtype": "context",
  "resourceTemplates": [],
  "valueConstraint": {
    "valueTemplateRefs": [],
    "useValuesFrom": [
      "urn:discogs"
    ],
    "valueDataType": {
      "dataTypeURI": "http://id.loc.gov/ontologies/bibfr
    },
    "defaults": []
  },
  "remark": "http://id.loc.gov/authorities/names.html"
}
```

## Resource Type Property

The last property template type is the **resource** type that allows resource templates to be embedded within another resource template.

```
{
    "mandatory": "false",
    "repeatable": "true",
    "type": "resource",
    "valueConstraint": {
        "valueTemplateRefs": [
            "lc:RT:bf2:Identifiers:Barcode"
        ]
    },
    "propertyURI": "http://id.loc.gov/ontologies/bibfran
    "propertyLabel": "Barcode"
}
```

`'http://www.w3.org/1999/02/22-rdf-syntax-ns#value']` and the data is stored in in an `items` object along with a language code for literals. If the user adds a value for the **bf:enumerationAndChronology** property, then an `items` object will be added to the corresponding section in the Redux state.

```
    resource: {
      'resourceTemplate:bf2:Identifiers:Barcode': {
        'http://www.w3.org/1999/02/22-rdf-syntax-ns#v
          items: {
            LuG2ym_Td: {
              content: '12345',
              lang: 'en'
            }
          }
        },
        'http://id.loc.gov/ontologies/bibframe/enumer
      }
    }
```

Sinopia's [Redux](#) state also caches copies of the resource templates that are active in the editor that is used for other functions in the editor like validation and RDF generation and is illustrated below:

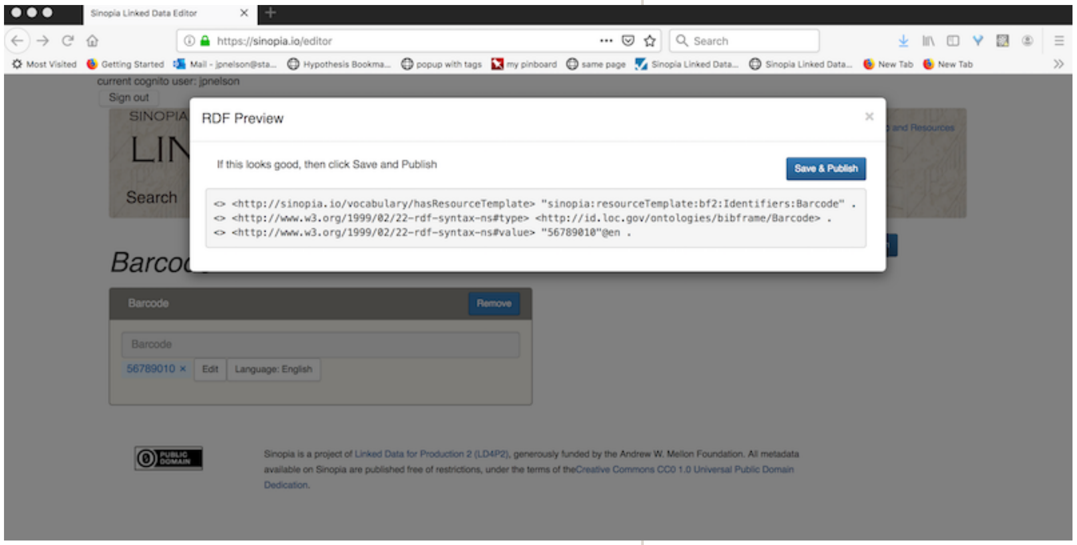
```
 entities: {
      resourceTemplates: {
        'resourceTemplate:bf2:Identifiers:Barcode': {
          id: 'resourceTemplate:bf2:Identifiers:Barco
          resourceURI: 'http://id.loc.gov/ontologies/
          resourceLabel: 'Barcode',
          propertyTemplates: [
            {
              mandatory: 'true',
              repeatable: 'false',
              type: 'literal',
              resourceTemplates: [],
              valueConstraint: {
                valueTemplateRefs: [],
                useValuesFrom: [],
                valueDataType: {},
                defaults: [
                  {
                    defaultLiteral: '12345'
                  }
                ]
              },
              propertyURI: 'http://www.w3.org/1999/02
              propertyLabel: 'Barcode',
              editable: 'true'
            },
            {
              mandatory: 'false',
              repeatable: 'true',
              type: 'literal',
              resourceTemplates: [],
              valueConstraint: {
                valueTemplateRefs: [],
                useValuesFrom: [],
                valueDataType: {}
              },
              propertyURI: 'http://id.loc.gov/ontolog
              propertyLabel: 'Enumeration and chronol
              editable: 'true'
            }
          ]
        }
      }
   .
   .
   .
}
```

Finally, we use the Redux state to store user credentials and session information using the AWS Cognito service and built with the Amazon [Amplify](#) SDK (software development kit).

## State to RDF (and back again)

### Redux to RDF

Using the editor's application state stored in Redux along with the resource templates cache, the Sinopia editor builds a RDF graph upon demand. When cataloging, the user at anytime can see what RDF is being produced by clicking on the **Preview RDF** button:



```
<> <http://sinopia.io/vocabulary/hasResourceTemplate> "sinopia:resourceTemplate:bf2:Identifiers:Barcode" .
<> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://id.loc.gov/ontologies/bibframe/Barcode> .
<> <http://www.w3.org/1999/02/22-rdf-syntax-ns#value> "56789010"@en .
```

### Quick `sinopia:hasTemplate` solution

In developing the Sinopia version **1.0**, we first worked on representing resource templates using React components followed by using Redux. After we had working codebase, we started generating RDF using the N3 RDF javascript module based on the application's Redux state. Using a OpenAPI yaml API configuration definition of the expected interactions with the Trellis backend, we successfully created the expected entity with a newly resolvable minted URI.

### Future Possibility: Machine Learning

We have tentatively started exploring the use of Machine Learning for part of these workflows, especially in trying to map incoming RDF with the existing resource templates in the Sinopia.

## Questions?

## Thank-you!