Hydramata Works

A focus on modeling Predicates and dynamic composition of Works

Introduction

A Response to Shared Development

- We want the same things…
  - But we can't agree on the details.
- We have the same things…
  - But they are just different enough.
- We do the same things…
  - But not quite and sometimes at different times.

Customization and Flexibility are Mandatory

- Create new resource types on the fly
- Create and rearrange predicates on the fly
- Expose command line ready services
- Leverage existing views for greater reusability
- Internationalize all the things

Methods

Ruby on Rails Engine

- A logical container for models, views, and services
- Exposes configuration points and generators
- Separates concerns from application

S.O.L.I.D. Design Principles

- Single responsibility; Data Structure or Behavior
- Open for extension, Closed for modification
- Liskov's substitution; Subclass behavior
- Interface segregation; Clear expectations of object interaction
- Dependency inversion; Allow for customization of collaborators

Test Driven Development

- Specify a feature by providing examples
- Write the integration test
- Iterate on units and tests to complete feature

Deliberate Decisions

- Tests must be fast because so Test Driven Development is possible
- Objects are of two major forms
  - Data Structures: Buckets of data, immutable
  - Functions: Behavior that takes an input and returns an output
- Craft services with an eye towards the command line
- Clear entry points for adopters of Hydramata::Works because others will be helping
- Automatic verification of Documentation

Scenario: Hydra Development Today

Given an existing Work Type of Article (i.e. a subclass of ActiveFedora::Base)
And the Abstract predicate is defined for other work types
When I want to add the Abstract predicate to Articles
Then I must add the corresponding Datastream class
And I must update the Article class
And I must update the Form view
And I must update the Show view
And I might update the Index view
And I must commit my changes
And I must push my changes
And I must deploy to a Staging environment to verify changes
And I must deploy to Production to see my changes

Scenario: Hydra Development with Hydramata Works

Given an existing Work Type of Article
And the Abstract predicate is defined for other work types
When I want to add the Abstract predicate to Articles
Then I must make a database entry associating Abstract to Articles

Customization via Diminishing Specificity

Look for *views* and *translations* first in the most specific case, then less general and so forth.
For example, rendering the show view of an Article's Abstract predicate, Hydramata looks for:

1. ./hydramata/works/properties/article/abstract/_show.html.erb
2. ./hydramata/works/properties/abstract/_show.html.erb
3. ./hydramata/works/properties/_show.html.erb

You can also specify that a given Work Type or Predicate renders or translates as another thing (i.e. Abstract will render as a Description, but translate as Abstract)

Persisted Data is the Canonical Data Model; Not the Ruby Model

Hydramata::Works interrogates your Fedora object to build the in-memory data structure. This may mean you need different interrogation and sniffing functions for your objects. It also means you have options for handling Fedora objects not ingested via a traditional Hydra application.

ActiveRecord::Base leans on the schema of the database to expose the attributes of the data structure. Migrations enforce this. ActiveFedora::Base relies on in-code representation of schema, which leads to migration challenges.

Don't Repeat Knowledge

Since Work Types and Predicates are defined in the database, it is possible to automate the build of documentation regarding the predicates that make up your work types; And use the internationalization to further explain what this is about.

Numerous Creases as You Move Your Data

From Fedora to In Memory it is Feasible and Trivial to:

- Only load the predicates that the user can see
- Load predicates based on the object's last updated date

From In Memory to Output Buffer:

- Define custom rendering options by Work Type, Predicate (Time of Year? Object Identifier)
- Define new buffers: To File System, To HTML, To JSON, To RDF

From Input to Persistence

- Allow trusted sources (i.e. Metadata librarians) to attach additional predicates and values to a given subject.
- Persist all user input without sending it straight to Fedora

From Web Request to Services:

- Extend existing behavior without Ruby tomfoolery

Conclusions

Hydramata::Works is in the pre-alpha stages of development. However, by adhering to the methodology, feature exploration and completion has been fast and flexible. And while there are many moving parts and configuration points, there are guideposts for entry both adoption and development.

By separating structure from behavior – that is to say focusing on an object's single responsibility – the path of upgradability and maintainability will be less than traditional Rails and Hydra application development.

Further Reading

- Avdi Grim's Naught - https://github.com/avdi/naught
- Avdi Grim's "Confident Ruby"
- Corey Haines "Understanding the Four Rules of Simple Design"
- Eric Freeman & Elisabeth Freeman's "Head First Design Patterns"
- Jim Weirich's Wyriki - https://github.com/jimweirich/wyriki
- Robert Martin's "The Clean Coder"
- Robert Martin's "Clean Code"

Acknowledgements

Thank you to the **Curate team** at: Data Curation Experts, Indiana University, Northwestern University, University of Cincinnati, University of Virginia
Thank you to the **Orcid House Development team**: Carolyn Cole (Penn State), Eric James (Yale), Glen Horton (University of Cincinnati), Jim Halliday (Indiana University), Mike Stromming (Northwestern University), Patrick Burke (University of Cincinnati), Paul Clough (Northwestern University), Sue Richeson (University of Virginia)

More on Hydramata Works

Follow along at ndlib.github.io/hydramata-works
Contribute at github.com/ndlib/hydramata-works
Example at github.com/ndlib/hydra_connect_demo
By email jfriesen@nd.edu

UNIVERSITY OF NOTRE DAME