Responsible and Sustainable Overrides in Ruby and Samvera in General

My Presentation for Samvera Connect 2022

Jeremy Friesen

[2022-10-18 Tue 10:34]

Contents

1 Metadata

Presentation Title Responsible and Sustainable Overrides in Ruby and Samvera in General

Name Jeremy Friesen

Pronouns he/him/his

Job Title Senior Lead Software Engineer

Organization Software Services by Scientist.com (aka SoftServ)

Conference Samvera Connect 2022

Date October 25, 2022 (2022-10-25)

2 Abstract

The Samvera **stack is deep**; and we often need to make **localized adjustment(s)** to address either an underlying bug or to **extend existing behavior**. The code-base has places for configuration, but sometimes that might not be enough. Join me on a foray into how you can make the Ruby/Rails changes you need now and not make things (too much worse) for your future self and others.

3 Starting with some Marginalia

I started preparing for this presentation by writing a **reference repository** on Github: jeremyf/responsible_overrides to demonstrate **some override strategies**.

But I got ahead of myself.

For this presentation, we won't look there. Instead we're going to take a slightly different approach, that ~~would~~ could lead to that aforementioned code-base.

We'll draw up some blueprints and prepare a foundation, if you will.

4 First, What do I Mean by Override?

Typically this is **re-opening** the upstream class/object and **replacing/adding/removing** functional logic in **your local instance**.

It can also be **copying and amending** the file into the same relative **load path** location and letting Rails pick this new file.

4.1 Overrides in Tension with Community

Using overrides is a natural extension of our oft invoked aphorism:

> If you want to go fast, go alone. If you want to go far, go together.

4.2 Consequence of Overriding

When we override something, we are **deviating from the anticipated path** and are now forging ahead **on our own**.

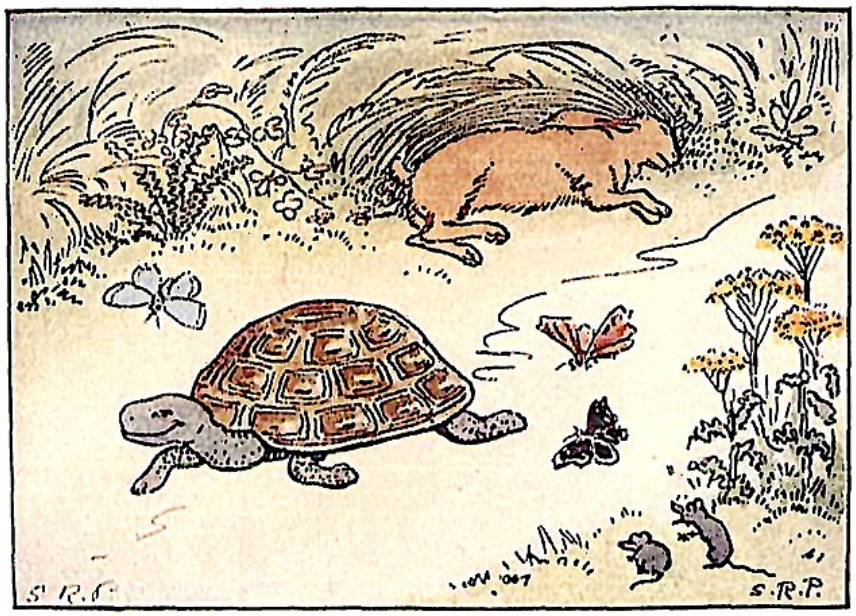
In that moment, we are choosing to go *fast*-er. But also *alone*-er.

Attribution: No machine-readable author provided. Nchele~commonswiki assumed (based on copyright claims)., CC BY-SA 4.0, via Wikimedia Commons

4.3 Lest We Forget

I want to offer Milan Kundera's observation from *Slowness*:

> There is a secret bond between slowness and memory, between speed and forgetting.



THE TORTOISE AND THE HARE

Attribution: The Tortoise and the Hare from Project Gutenberg

4.4 Background and Guidance

In this talk, I want to provide some **background and guidance** for our override journey.

To help us enter into memory and conversation **on both process and approach**.

Attribution: Yale University Press, Public domain, via Wikimedia Commons

5 Approach

Let's walk through a **responsible approach** for doing so:

1. First, **assess** your available options.
2. Second, work to **contain** the changes you'll be making.
3. Third, **document** what you've done.

Attribution: Unknown authorUnknown author, Public domain, via Wikimedia Commons

5.1 Assessment

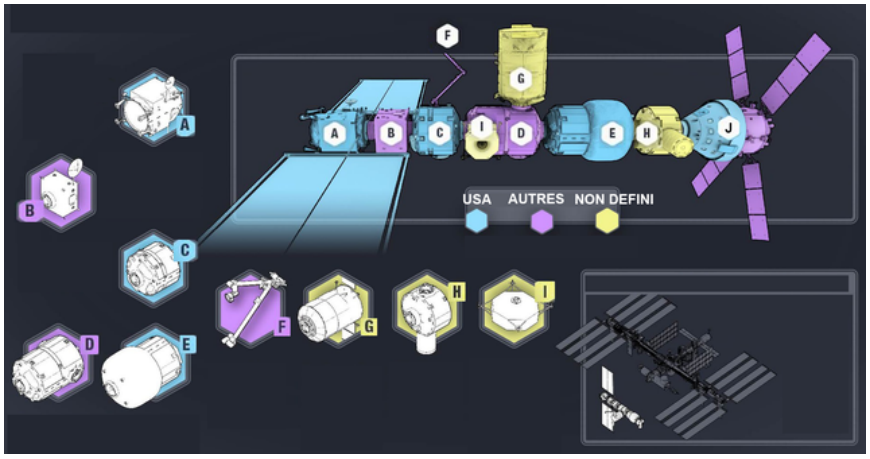
Before you begin the copy/paste journey, look for **places in the code** where upstream developers might have created **creases for customization**:

- Configurations
- Class variables
- Other sundry items

Attribution: Peter Trimming from Croydon, England, CC BY 2.0, via Wikimedia Commons

5.1.1 Configurations

Look to the **configuration file(s)**. Take some **time** to orient to what all the developers have indicated is configurable; and even encourage you to configure.



Attribution: NASA, Public domain, via Wikimedia Commons

1. Hyrax::Configuration

   Below is code that I recently added to Hyrax::Configuration:

```
class Hyrax::Configuration
  attr_writer :derivative_services
  # The registered candidate derivative services.  In the
  # array, the first 'valid?' candidate will handle the
  # derivative generation.
  #
  # @return [Array] of objects that conform to
  #         Hyrax::DerivativeService interface.
  # @see Hyrax::DerivativeService
  def derivative_services
    @derivative_services ||= [
      Hyrax::FileSetDerivativesService
    ]
  end
end
```

   Note: For presentation purposes I have made changes to the code formatting.

2. Hyrax::Configuration Continued

   In your application's initializers you can add either of the following:

   The first example will **replace the existing** derivative_services.

```
Hyrax.config.derivative_services = [
  MyDerivativeService
]
```

The next example will **prepend the new service** to the array of existing services.

```
Hyrax.config.derivative_services
  .unshift(MyDerivativeService)
```

Note: I chose the above code because it helps lead to the second point of assessment.

5.1.2 Class Methods

Throughout Hyrax you might find `class_attribute` or `mattr_accessor` calls. These are potential points of configuration.

These are "advanced configuration" options.

Image: Le bal paré; Ausschnitt aus Stich von Antoine Jean Duclos, 1774

1. Hyrax::DerivativeService Revisited: Part 1

   Let's delve a bit deeper into `Hyrax::DerivativeService` class; as of October 18, 2022 (2022-10-18) on the `main` branch.

```
class Hyrax::DerivativeService
  # @deprecated favor Hyrax.config.derivative_services=
  def self.services=(services)
    Deprecation.warn(
      "Hyrax::DerivativeService.services= is deprecated; " \
"favor Hyrax.config.derivative_servies="
    )
    Hyrax.config.derivative_services = Array(services)
  end

  # @deprecated favor Hyrax.config.derivative_services
  def self.services
    Deprecation.warn(
      "Hyrax::DerivativeService.services is deprecated; " \
"favor Hyrax.config.derivative_servies"
    )
    Hyrax.config.derivative_services
  end
end
```

Continued on next page...

2. Hyrax::DerivativeService Revisited: Part 2

...Continued from previous page

```
class Hyrax::DerivativeService
  # @api public
  #
  # Get the first valid registered service for the given
  # file_set.
  #
  # @param file_set [#uri, #file_set]
  # @return [#cleanup_derivatives, #create_derivatives]
  def self.for(file_set, services: Hyrax.config.derivative_services)
    services.map do |service|
      service.new(file_set)
    end.find(&:valid?) || new(file_set)
  end
end
```

The above code allows you to configure the Hyrax::DerivativeService via class attribute overrides.

You can use either `Hyrax.config.derivatives` or the deprecated `Hyrax::DerivativeService.services=`.

3. Presently Released Hyrax::DerivativeService Code

   I chose the above because up until recently Hyrax::DerivativeService looked looked like this:

```
class Hyrax::DerivativeService
  class_attribute :services
  self.services = [Hyrax::FileSetDerivativesService]
  def self.for(file_set)
    services.map do |service|
      service.new(file_set)
    end.find(&:valid?) || new(file_set)
  end
end
```

   In the released code there is no configuration option. Instead the "config" option was tucked away.

   The samvera-labs/newspaper_works gem makes use of this point of configuration.

4. How to Override the Class Attribute

   To configure the released version, I made the following changes in my application's config.

```
config.to_prepare do
  # See https://gitlab.com/notch8/adventist-dl/-/issues/147
  #
  # By default plain text files are not processed for text
  # extraction.  In adding
  # Adventist::TextFileTextExtractionService to the
  # beginning of the services array we are enabling text
  # extraction from plain text files.
  Hyrax::DerivativeService.services
    .unshift(Adventist::TextFileTextExtractionService)
end
```

5. Other Examples of Class Attributes

   If you've worked in Samvera you've probably seen other instances:

   - Hyrax's myriad of `*_presenter`, `*_builder_class`, etc.
   - Blacklight::SearchBuilder.default_processor_chain
   - Blacklight::Rendering::Pipeline.operations

   Consider making class adjustments in your `config/application.rb` or in the gem specific initializer.

Attribution: jimmyweee, CC BY 2.0, via Wikimedia Commons

5.1.3 Other Sundry Items

As one might expect there are many ways to make changes. Hyrax has several places I'll touch on but warrant their own review:

Hyrax::CurationConcern want to alter the Actor Stack? This is your file for guidance on how to do that. Altering can mean removing, adding, or shuffling the order of the actors.

Hyrax::Transactions::Container the successor to the Actor Stack; this defines what all "happens" when we perform a transaction in Hyrax. And it's configurable.

Hyrax::Publisher here is where you can find documentation on the "application-wide publisher for Hyrax's Pub/Sub interface."

5.2 Containment

Now that we've equipped ourselves to perform an assessment; let's talk about containment.

1. No Crease to Be Found

   Let's assume you don't find a suitable crease in the code and need to do something more drastic.

   Let's break this into two categories:

   - Views
   - Not Views

2. Quick Explanation of Ruby's Load Path

   In Ruby and Rails, we have a `$LOAD_PATH` array. It contains a list of directories. When we call `require`, we test for the given parameter's existence in each of the directories.

5.2.1 Views

If you need to **make changes to a view**:

Copy those views and paste them into the same relative directory structure in your application.

Warning: Any changes in the upstream file will not show up in your application; this can create some notable breaks as you maintain your application and others maintain that upstream dependency.

5.2.2 Not Views

You have two primary options:

Copy the file Similar to the view pathway, and one that I don't recommend. Because there are more durable/robust mechanisms.

Prepend a module I go through those patterns in github.com/jeremyf/responsible_overrides. The tl;dr is to leverage `Module.prepend` or `Module.class_eval`.

1. Minimize the Code You Change

   > Measure twice, cut once.

   When you use the `Module.prepend`, work to change the least amount of code as possible; all code you copy or adjust is a fork in the road and you're taking a different pathway than others.

   Attribution: Georgia National Guard from United States, CC BY 2.0, via Wikimedia Commons

2. Think of Others

Consider what you are changing; it is likely others elsewhere may also want this.

Write and refactor accordingly.

3. Reduce Surprises in Your Code

   Follow a consistent pattern for indicating those changes in your codebase. Ask yourself, "How will others know about this change?"

4. Consider Logical Groupings

   When you need to override several files for a singular concept, consider placing those modifications in a single file.

   Let folks know how these relate; both in documentation and in file organization; itself a documentation strategy.

Attribution: Michal Klajban, CC BY-SA 4.0, via Wikimedia Commons

5.3 Document

Which leads to documentation. You have several things to consider:

- Your **local application** and it's maintainers.
- **Other adopters** that may be interested in your approach.
- Tracking **drift** between your local application and it's upstream(s).

Attribution: Nachoom Assis, CC BY-SA 3.0, via Wikimedia Commons

5.3.1 Override Procedure and Policy

I encourage you (and your team) to **document how you document** these kinds of overrides; write a policy or procedure if you will.

At Software Services by Scientist.com our overrides go in files with the suffix of `_decorator.rb`.

We document that in How To: Override a method from a dependency without copying the entire file over.

1. Consider Your's and Our's Future Selves

   In your overrides consider how you might use:

   - file names
   - method names
   - documentation
   - inline comments
   - commit messages
   - logging

   All **in service of helping** future code spelunkers know both the how and why of the change.

5.3.2 Sharing is Caring

Share the **how** and **why** of your changes:

- **File issues** in the upstream repository.
- **Link to pull requests** to provide hints of approach.
- **Create a fork of your changes**
- **Submit a pull request** to the upstream.
- **Hop on the Samvera Tech call** and talk about the change.
- Present on it, email the Samvera Tech list, jump on Slack.

5.3.3 The Goal is to Go Together

Share this information because you will likely find common cause amongst folks; or learn of an alternate approach that doesn't necessitate as much code drift.

The goal is to **make it easy to stumble upon** the fact that you've made a local change and to understand the implications of those changes.

6 Conclusion

To reiterate:

- **Assess** where to make the change.
- Work to **contain** the impact of the change.
- And **document why and how** you made the change.

This is all in service of others:

- Our patrons
- Our future selves
- Our current colleagues
- Our future colleagues

Let's help each other **cope with the antics** of today and yesterday.

7 About Me

Jeremy Friesen (he/him/his)

Senior Lead Software Engineer at Software Services by Scientist.com
Email: jeremy@jeremyfriesen.com Website: `https://takeonrules.com`

From November 2021 to February 2022, we fostered 4 puppies and their mom, through Clancy's Dream, a Border Collie Rescue program. We adopted Queen Anne's Lace "Lacey" Lulu Bunny Belle. The other larger dog is our 8 year old Owlbear "Ollie" Camus.

Through Clancy's Dream, Orlando, Mookie, and Gambit are all adopted and living wonderful lives in Illinois and Indiana.

8 Licensing