Using Vitro and VIVO to extend the Fedora/based Sufia repository platform: the Sufitro/SufIVO use case

Element	Description
Title (Goal)	Explore and document the potential for VIVO to provide unique new functionality to Sufia and other Fedora-based Hydra tools by leveraging largely existent work and extending VIVO to communicate directly with Fedora 4, Sufia, and Blacklight
Primary Actor	As a repository system architect, I want to know what functionality I will be able to offer my stakeholders and end users by taking advantage of largely existing tools and interfaces.
Scope	This is a major feature having broad implications for Fedora, Hydra, and VIVO
Level	This is a high-level use case aimed at a technical audience
Story	As a repository system architect, I need to provide more flexible collection management and customizable collection experiences than the Fedora and Sufia tools provide out of the box, and am looking for a solution that does not require extensive code modification with every new repository

Supplementary Content

Note this use case is based in part on a complementary Fedora use case entitled Reasonable curator's admin interface. Here is the story from that use case:

Story
The curator of Fedora data wants to be able to discover, modify, and ingest all data in the Fedora repository before the curator's organization has had the resources to put into building a personalized, user-friendly special administrative interface. That is, without any bells and whistles, but with more usability than the existing developer-focused Fedora administrative interface, the (non-programming, non-XML expert) curator wants to be able to add one or many items to the Fedora repository, discover one or many items via search, purge one or many items, and modify one or a batch of items. They want to be able to handle new datastreams without necessarily writing special code. They want to be able to at least see a simple field-based editor for in-line XML datastreams, and an upload/view tool for external/managed datastreams. They want to be able to crosswalk metadata fields in those XML datastreams to each other, so, for example, when they are editing the title of an object they do not need to make the same edit in oai_dc:title, local_dc:title, local_vra:title, etc, as you currently have to do in the existing developer-focused administrative interface. Explanation: The number one roadblock that we see when talking to members of our community asking if they want to be partners in our Fedora venture is the lack of an out-of-the-box curator-focused administrative interface for Fedora data. While the response from the Fedora community has persistently been that Fedora is a framework, and user facing interfaces are to be designed by the outside community, the reality is that every single existing administrative tool is designed for specific institutional use cases, usually specific metadata formats (cf. sufia), and don't provide an out-of-the-box view of the data. Right now we are working with one potential Fedora user who is currently spending an enormous amount of money and time to create a minimal user interface before they find out if Fedora is even something that will work for them, because that minimal view of the data and ability to have non-technical people work with it is necessary for them to make an informed decision.

Platforms involved

- W3C Linked Data Platform (LDP)
  - LDP Primer
  - LDP Specification
  - Fedora 4.x documentation: Linked Data Platform
  - Apache Marmotta
- Apache Camel
  - What exactly is Apache Camel? from Stack Overflow
    - (somewhat cryptically) "Apache Camel is messaging technology glue with routing. It joins together messaging start and end points allowing the transference of messages from different sources to different destinations. For example: JMS -> JSON, HTTP -> JMS or funneling FTP -> JMS, HTTP -> JMS, JSON -> JMS"
    - Enterprise Integration Patterns (EIP) are *blueprints* for how we could *best* design large component-based systems, where components can be running on the same process or in a different machine. They basically propose that we structure our system to be *message* oriented -- where components communicate with each others using messages as inputs and outputs and absolutely nothing else. They show us a complete set of patterns that we may choose from and implement in our different components that will together form the whole system.
- Ruby
  - Ruby-RDF (GitHub)
  - Ruby repository
- Blacklight