CollectionSpace Configuration and Extension

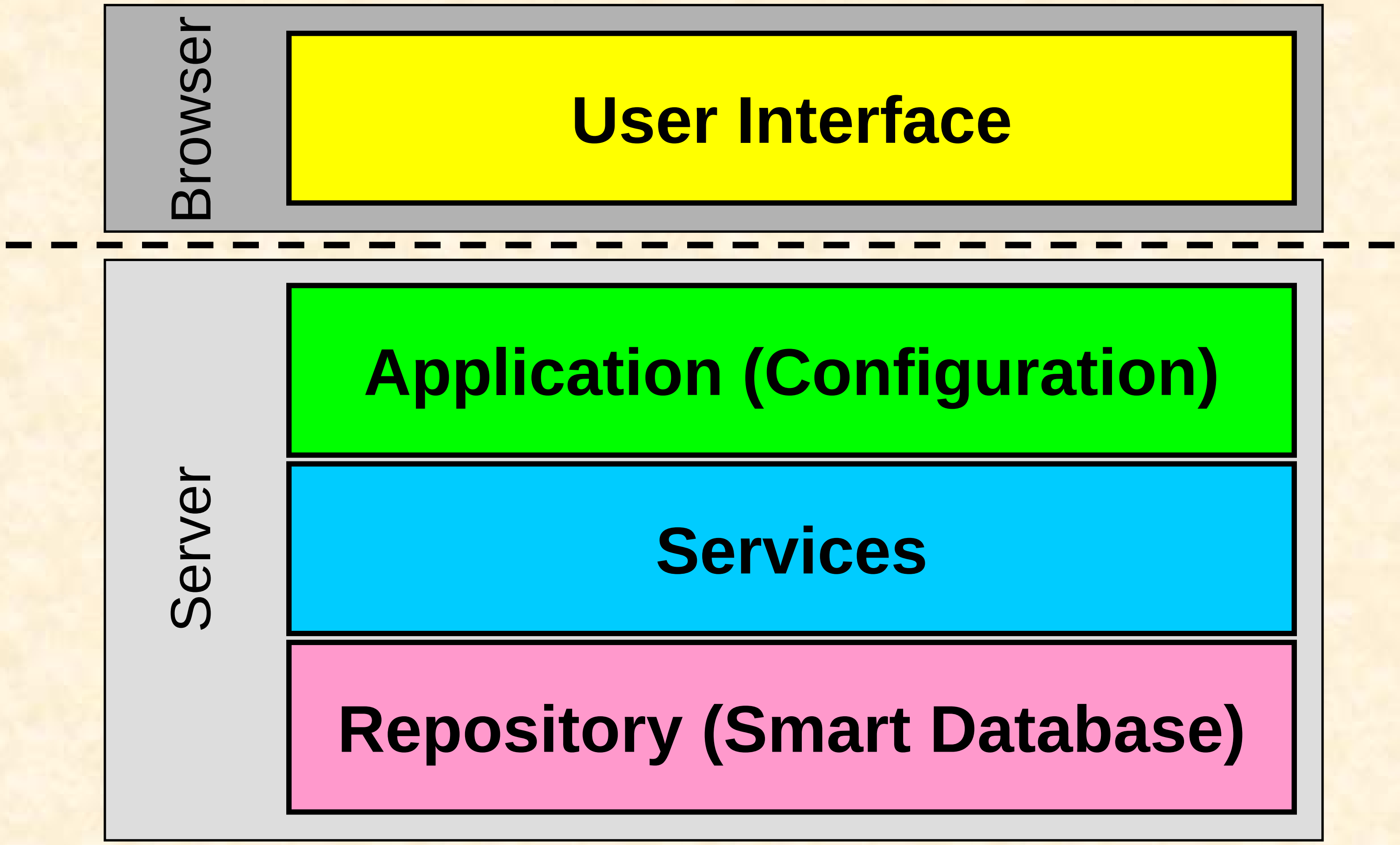
i290-rmm

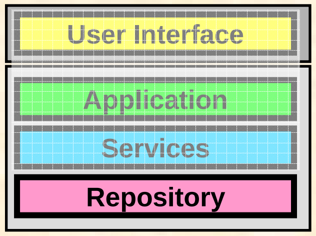
Patrick Schmitz

Overview

- Architecture and function
- Shared semantics, domain and local extensions
- IT Architecture and community dynamics
- Extensions, overlays, and replacements
- Multi-tenancy and its implications
- Communications and project workflows (using the wiki, IRC, email lists)

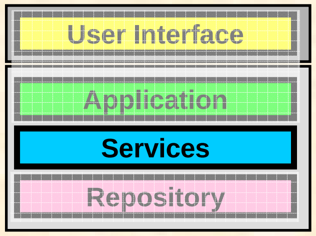
CollectionSpace Architecture





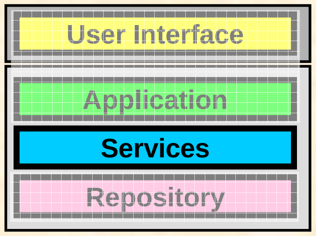
Repository

- Smart Database (a.k.a. *Object Store*)
- Provided by ECM platform
- Manages objects, relations in the database
- XML Schema driven
- Handles versioning, media, etc.
- Supports SQL-like query language
- Used directly by Reporting engine



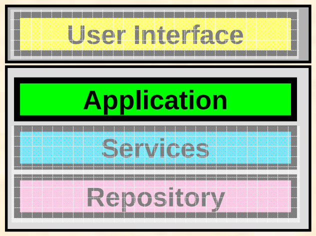
Services

- Provide Web-Services abstraction/access
- Support “CRUDL”, search
- Manage the repository
  - Coordinate common and extension schemas
  - Handle Security (authentication and authorization)
  - Provide “multi-tenancy” support
- Also model some state, workflow
- Largely independent of one another
- Fairly low-level (entity-, not page- based)
- Mostly XML payloads (currently)



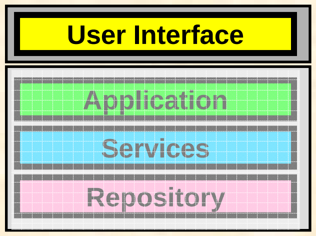
Services examples

- **`/cspace-services/collectionobjects`**
- **`…/intakes`**
- **`…/loansin?kw=damaged`**
- **`…/personauthorities`**
  - **`…/personauthorities/{id}/items/{item-id}`**
  - **`…/personauthorities/{id}/items?pt=joe`**
- **`…/media`**
  - **`…/media/{id}/blob/content`**
  - **`…/media/{id}/blob/derivatives/thumbnail`**
- **`…/reports?doctype=Accession`**
- **`…/users`**
- **`…/relations`**



Application Layer

- Provides a UI-specific abstraction
  - Also supports web services, but page-based
  - Maps/aggregates UI requests to service requests
  - Mostly json payloads
- Aggregates service payloads into application data model
  - For editors and admin (cataloging, loans, intakes, etc.)
  - For widgets/tools (sidebar lists, term-completion, etc.)
- Manages configuration of termlists, authorities
- Manages UI-model aspects of customization and extension (overlays), and multi-tenancy



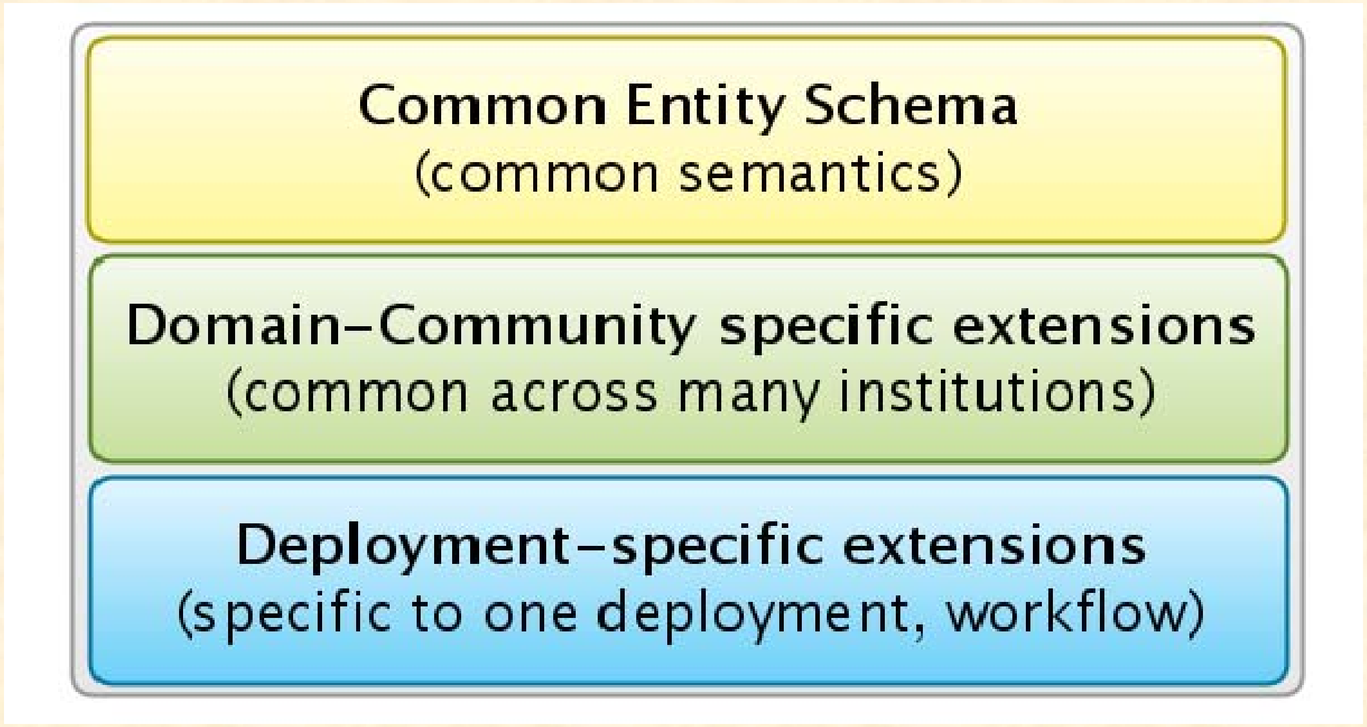
User Interface

- Client (browser) software (Javascript-based)
- Kind of like a template engine
- Maps a “UI-schema” from application layer, to the page templates (HTML)
- Synthesizes HTML for lists, repeating blocks, etc.
- Also maps HTML back to a data-model, for create and update (Save) operations
- Includes various widgets
  - Term completion, Structured date editor, pagination
  - Can integrate “foreign” widgets, like GTK Calendar

Shared semantics, extensions

- Want to have common information that covers many cases, many domains
- Must support additional information, and alternate models
- Traditional services model has fixed “contract” (XML schema for information)
- CollectionSpace supports multiple *parts*
  - Services only “understand” common part
  - But manage (save+get, search over) additional parts
  - UI does not really care whether common or custom

Schema Extension Model


Common Entity Schema
(common semantics)
Domain–Community specific extensions
(common across many institutions)
Deployment–specific extensions
(specific to one deployment, workflow)

Schema model for a customized service deployment

IT Architecture and community dynamics

- Community supported and sustained
- Need sub-communities to form, and share ownership for domain extensions
  - This is largely new to museums, for software
  - There are *some* already, e.g., herbaria
- How to get them thinking as *community*?
  - Step one: push their extensions into this model, using domain *plus* local schemas
  - Step two: foster shared discussion/governance

Extensions, overlays, replacements

- Application+UI ship with base (default) config
- Framework allows for key pieces to be replaced (in whole or in part), or extended:
  - HTML templates and widgets
  - Message bundles (labels, static text)
  - CSS (for general themes, or specific layout)
  - Javascript files (for UI logic)
- *Default* resources are replaced/overlaid by *tenant* resources

HTML templates

- Main and Admin record editors
  - Cataloging, Intake, Loans UI
  - Users, Roles, Reports, Controlled-vocabs, etc.
- Advanced Search editors
  - Subset of fields used for search
  - UI varies by field type (strings, dates, numbers)
- Widgets and components, e.g.,
  - Term completion widget
  - Structured Date editor
  - Media uploader
- Overlays *replace* a template

Message Bundles

- All strings are *named*, and have default values
- UI Framework binds values to HTML templates
- Override to change form labels, titles, etc.
- Can be used to localize the UI
- Overlays bundles are additive (selectively replace values)

CSS, javascript, json

- These define style, page logic, data models
- Modularized for pages, components, etc.
- Can override individually (replace)
- Can also add extension script, stylesheets, etc.

Multi-tenancy implications

- Each tenant has separate model in repo
- Base resources shared across all tenants
  - Services schemas and configuration
  - Application configuration of data models, etc.
  - UI templates, CSS, bundles, etc.
- Default/demo tenant has no extensions
- Additional tenants specify extensions, customizations, overlays, etc.
- User Accounts must be tied to one or more tenants
- Media, reports, etc. *could* be shared, should not be

UI config layout

- Files (also) shipped as part of Web-app
  - In /defaults, includes “base” resources
    - /defaults/bundle, /defaults/css, /defaults/js , /defaults/config, etc.
    - /defaults/html: high level page layouts
    - /defaults/html/pages: record editor templates
    - /defaults/html/components: widget templates
  - In /tenants, folders for each tenant by name, within which same structure is used to add overlays
- Only need add the ones you want to customize

Application config layout

- Files shipped as part of Web-app
  - In resources/defaults, includes “base” configuration for each procedure
  - In resources/tenants, folders for each tenant by name, within which domain and local overrides for configuration can be added.
- Only need add the ones you want to customize

Services config layout

- Base schemas shipped as part of Web-app
  - Expanded to /nuxeo-server/schemas
  - Development framework handles this, but requires Java development tools (ant, maven).
- Extension schemas added for a service, then declared in configuration
- Plan is to generate this from the Application configuration (automatically)

Project+Community process

- New procedures, objects, features etc.,
  - Proposals presented to community for review
  - Schemas sketched on wiki, discussed on *talk* list
  - UI Wireframes attached to wiki
  - Integration issues discussed on *work*, *tech* lists
  - May be developed by core team, or some museum or group of museums
  - Formal review process for contributing to core
- Bugs, refinements, etc.
  - Filed as issues in JIRA
  - Fixed with patches
- Mapping, customizations often public, some on local wikis
- Started with SVN, moving to Git