Common Data Model

CommonDataModel Introduction

The Library Simplified circulation manager uses a complex data model to represent the current state of:

- All the libraries which use the circulation manager.
- All the ebooks available to the patrons of those libraries.
- All the patrons who use the circulation manager, with their active loans and holds.

Looking at the whole data model at once can be overwhelming, so we'll consider it as a few smaller simpler systems:

- Bibliographic metadata
- Licensing
- Works
- Custom lists
- Libraries
- Patrons
- Site configuration
- Background processes

These systems overlap around a few key classes, mainly `DataSource`, `Identifier`, `LicensePool`, and `Work`.

The code for the data model is in the `model` package of the `server_core` project.

This data model is common between the circulation manager and the metadata wrangler, although some pieces are exclusively used by one component or the other. For example, only the circulation manager has lanes or patrons, and only the metadata wrangler has integration clients. The library registry component has a separate data model which is similar but much simpler.

For the sake of simplicity, this document will talk about "books", but the rules are the same for audiobooks and other forms of content.

Bibliographic metadata

Bibliographic information is information *about* books as opposed to the books themselves. A book's title, its cover image, and its ISBN are all bibliographic information--the text of the book is not. Bibliographic information flows into the circulation manager and metadata wrangler from a variety of sources, mainly OPDS feeds and proprietary APIs. We keep track of all this information and where it came from, and when necessary we weigh it, sort it, and boil it down into a small amount of information that can be used by other parts of the system.

`DataSource`

A `DataSource` is some external entity that puts data into the system. This data generally falls into two categories:

- Bibliographic information *about* a book, such as its title or cover image. This goes into the bibliographic metadata subsystem.
- Licensing information which can be used to serve actual copies of the book to library patrons. This goes into the licensing subsystem.

Some examples of `DataSources`:

- Overdrive, Bibliotheca, and Axis 360 license commercially published ebooks to libraries for delivery to patrons. They also provide bibliographic information about the books they license.
- Standard Ebooks provides bibliographic information about books, as well as free copies of the books themselves.
- OCLC and Content Cafe provide bibliographic information about books, but have no way of giving access to the actual books.
- VIAF provides information about the people who write books, but very little about the books themselves.
- The New York Times knows the ISBNs of the books on its best-seller lists, but not much more.

A `DataSource` may also:

- Provide many `Editions`
- Provide many `Equivalencys`
- Provide many `Hyperlinks`
- Provide many `Resources`
- Provide many `Classifications`
- Provide many `CustomLists`
- Grant access to many `LicensePools`
- Provide many `LicensePoolDeliveryMechanisms`
- Generate many `CoverageRecords`
- Have many associated `Credentials`
- Have one `IntegrationClient`

`Identifier`

An `Identifier` provides a way to uniquely refer to a particular book. Common types of `Identifier` include ISBNs and proprietary IDs such as Overdrive or Bibliotheca IDs.

An `Identifier` may:

- Have many `Classifications` representing how the book would be shelved in a bookstore or library. (See the classification subsystem.)
- Have many `Measurements` of quantities like quality and popularity. (See the measurement subsystem.)
- Have many `HyperLinks` to associated files such as cover images or descriptions. (See the linked resources subsystem.)
- Participate in many `Equivalencys`.
- Serve as the `primary_identifier` for multiple `Editions`.
- Serve as the `identifier` for many `LicensePools`, through `Collection`.
- Be associated with one Work, through Edition

`Equivalency`

An `Equivalency` is an assertion made by a `DataSource` that two different `Identifiers` refer to the same book.

- The `strength` of the `Equivalency` is a number from -1 to 1 indicating how much we trust the assertion. When Overdrive says that an Overdrive ID is equivalent to an ISBN, we give that `Equivalency` a `strength` of 1, because Overdrive got the ISBN from the publisher and assigned the Overdrive ID itself. When OCLC says that two ISBNs represent the same book, we give it a lower `strength`, because OCLC is frequently wrong about this. A negative `strength` means that the `DataSource` is pretty sure two `Identifiers` represent *different* books.

`Edition`

An `Edition` is a collection of information about a book from a particular data source. Like most items in the "bibliographic metadata" section, it represents an *opinion*. If different data sources give conflicting information about a book, that's fine -- everyone has their opinion. When this happens, we create multiple `Editions` and we sort it out later, when it's time to make the *presentation edition*.

An `Edition`:

- Has one `DataSource`. This is the data source whose opinions are recorded in the `Edition`.
- Has one `Identifier`, the `primary_identifier`. This identifies the book the data source is talking about.
- Contains basic metadata -- title, series, language, publisher, medium -- for that book.
- May have one or more `Contributors`, through `Contribution`.
- May be the *presentation edition* for a specific `Work`. The presentation edition is a synthetic `Edition` created by the system. We look over a bunch of `Editions` which are all (supposedly) talking about the same book, and consolidate it into a new `Edition` containing the best or most trusted metadata.

The contributor subsystem

This system basically tracks who wrote which book. There are two classes in this subsystem: `Contributor` and `Contribution`.

`Contributor`

A `Contributor` is a human being or a corporate entity who is credited with work on some `Edition`. The credit itself is kept in a `Contribution`, which ties a `Contributor` to an `Edition`.

A `Contributor`:

- Contains basic biographical information about a person or corporation. Most notably, it has both a `display_name` such as "Octavia Butler", the name that would go on the front of a book, and a `sort_name` such as "Butler, Octavia", the name that would go in a card catalog.

`Contribution`

A `Contribution`:

- Links a `Contributor` to an `Edition`.
- Contains a `role` describing the work the `Contributor` did on the `Edition`. Common roles include author, editor, translator, illustrator, and narrator.

The classification subsystem

This system tracks how a book might be classified in a card catalog or shelved in a bookstore. There are two classes in this subsystem: `Subject` and `Classification`.

`Subject`

A `Subject` represents a classification that someone might give a book. `Subject` handles a variety of classification schemes: Dewey Decimal, LLC, LCSH, BISAC, proprietary systems like Overdrive's, and free-form tags, among others. Four pieces of information might be derived from the `Subject`, and will be stored with the `Subject` if possible:

- Genre ("Billionare Romance" is a type of romance)
- Fiction/nonfiction status ("Science Fiction" is always fiction)
- Target audience ("Young Adult Fantasy" is always YA)

- Target age ("Picture books" are generally for very young children, not 12-year-olds.)

`Classification`

A `Classification` is someone's opinion that a book should be filed under a certain `Subject`.

A `Classification`:

- Links a `Subject` to an `Identifier`.
- Has an associated `DataSource` -- this tracks whose opinion it is.
- Has an associated `weight` representing how certain we are that the book should be filed under this subject. The higher the number, the more certain we are. If OCLC says that a single library has filed a certain book under "Whales", we'll record that information but give it a low `weight`. If OCLC says that ten thousand libraries have filed this book under "Whales", then it's probably about whales.

`Genre`

There are many different data sources which use many different classification schemes for the same books. Rather than expose this chaos to patrons, we have defined about 150 `Genres`, corresponding to the sections of a large bookstore or branch library: "Romance", "Biography", and so on.

Each `Subject` may be associated with a `Genre`. When `LicensePools` are turned into `Works`, all the related `Classifications` are gathered together. We then assign the Work to the `Genre` that showed up the most.

A `Genre` may also be associated with one or more `Lanes` -- this is the primary technique we use when choosing how to show books to patrons.

`Measurement`

A `Measurement` is a numeric value associated with an `Identifier`. It represents some quality that distinguishes one book from others. The most useful measurements are *popularity* (a popular book is read/accessed/purchased/accessioned more often) and *rating* (a highly rated book is considered to be of high quality).

The linked resources subsystem

This system keeps track of external resources associated with a book. An "external resource" can be pretty much anything, but these are the most common types of resources we track:

- A cover image
- A thumbnailed version of a preexisting cover image
- A textual description
- An EPUB copy of a free book
- A review

`Hyperlink`

A `Hyperlink` represents a connection between an `Identifier` and a `Resource`. It contains two extra pieces of information about the link:

- A `DataSource` -- who provided this link?
- `rel` -- what is the relationship between the `Identifier` and the `Resource`? "There's a link" is very vague; this is more specific. Different `rel` values are defined for a cover image, a thumbnail image, review, a description, a copy of the actual book, and so on.

`Resource`

A `Resource` represents a document found somewhere on the Internet -- probably either a cover image or a free book. It has a `url`, and that's basically it -- everything about the document itself is kept in `Representation`.

- A `Resource` that's an image may be chosen by an `Edition` as the best available cover image for a given book.
- A `Resource` that's a textual description may be chosen by a `Work` as the best description for a given book.
- A `LicensePoolDeliveryMechanism` for an open-access book will point to a `Resource` that represents the book itself.

`Representation`

A `Representation` is a local cache of a `Resource`. It represents our attempt to actually download a `Resource` and records what happened when we tried. If everything went well, the `Representation` will contain a file--binary, text, HTML, or image. Otherwise, the `Representation` will contain information about what went wrong -- maybe the server was down or something.

Circulation managers don't usually create `Representations` -- they rely on the metadata wrangler to do that.

An image `Representation` that's a thumbnail of another image `Representation` is connected to its original through `.thumbnail_of`.

Putting it all together

Here's how the whole subsystem works together. Let's say one of our data sources that claims the URL http://example.org/covers/my-book.png is a cover image for the ISBN "97812345678". We want to represent this fact in our system.

1. We'd create an `Identifier` for the ISBN "97812345678".
2. We'd create a `Resource` for `http://example.org/covers/my-book.png`
3. We'd create a `Hyperlink` with the `rel` "http://opds-spec.org/image", for "cover image". The `.data_source` of this `Hyperlink` would be set to the `DataSource` that made the original claim.
4. We don't have to actually download http://example.org/covers/my-book.png, but if we do decide to download it, the binary image will be stored as a `Representation`. If there's a problem and we can't complete the download, that fact will be stored in the `Representation`instead.
5. If we download the image and everything goes well, we may also decide to create a thumbnail out of it. This would be stored as a second `Representation`, and its `.thumbnail_of` would point to the original, full-size `Representation`.

ResourceTransformation

A `ResourceTransformation` represents a change that was made to one `Resource` to generate another `Resource`. Currently it's used in the circulation manager's "cover image upload" feature. You can upload a background image (the original `Resource`) and paste the title and author onto it (a `ResourceTransformation` which results in a second `Resource`).

Theoretically, thumbnailing could also be handled as a `ResourceTransformation`, but it's probably not worth making this change.

Licensing

Collection

A `Collection` represents a set of books that are made available through one set of credentials.

One library may have multiple collections from different vendors, and multiple libraries on the same circulation manager may share a collection.

A `Collection` may have children which are also `Collections`. We model an Overdrive Advantage account as a child `Collection` of the main Overdrive `Collection`.

The books themselves are stored as `LicensePools`, and the credentials are stored in an `ExternalIntegration`.

LicensePool

A LicensePool represents an agreement on the part of a book vendor to actually deliver a book to a patron.

a group of licenses granting access to one particular Work.

If a Work is not associated with a LicensePool, patrons will not be able to check it out.

In some cases, usually involving open-access LicensePools, there may be more than one LicensePool associated with the same Work; if this happens, the LicensePool which provides the highest-quality version of the book will take precedence.

Each `LicensePool`:

- is associated with an Identifier, representing how the vendor identifies the book.
- is associated with a DataSource, representing the vendor who provides the book.
- belongs to one Collection.
- has one presentation edition, containing the most complete set of metadata available for the book.
- can have many Loans, Holds, Annotations, and Complaints
- can have many `CirculationEvents`.
- should have at least one `DeliveryMechanism`, through `LicensePoolDeliveryMechanism`.
- has a RightsStatus, through LicensePoolDeliveryMechanism.

`DeliveryMechanism` and `LicensePoolDeliveryMechanism`

A `DeliveryMechanism` describes what format a book is actually available in. There are two parts to a `DeliveryMechanism`: 1) the DRM scheme implemented by the distributor, if any, and 2) the format of the book (EPUB, PDF, audiobook manifest, Kindle, and so on).

`LicencePoolDeliveryMechanism` is a three-way join table: a record of a promise by a vendor (identified by a `DataSource`) to deliver copies of a book (identified by an `Identifier`) in a specific format (identified by a `DeliveryMechanism`).

RightsStatus

A `RightsStatus` represents the terms under which a book is being made available to patrons. The most common varieties of RightsStatus are 1) in copyright, 2) public domain, and 3) a Creative Commons license. "In copyright" implies that a book is being made available to patrons by virtue of a licensing agreement between the library and the vendor. The other `RightsStatus` values imply that a book is being made available to library patrons on the same terms as it would be to the general public.

Complaint

Patrons may lodge one or more Complaints against a specific LicensePool. Complaints indicate problems with specific books. For example, a Patron can lodge a Complaint stating that a book is incorrectly categorized or described, or that there is a problem with checking it out, reading, or returning it.

`CirculationEvent`

A `CirculationEvent` is a record of something happening to a LicensePool. A `CirculationEvent` happens when an event takes place within the circulation manager (e.g. a work is checked out or placed on hold), or when we notice that an event happened on the distributor's side (such as licenses for a book being added or removed), or when a client app (i.e. a book having been opened).

`CirculationEvents` are aggregated and used to create library analytics.

Works

A Work represents a book in general, as opposed to one specific edition of a book, or a specific licensing agreement to deliver copies of a book.

A Work:

- May have copies scattered across many LicensePools
- May have many Editions, but derives its presentation metadata from one particular Edition, which is known as its “presentation edition.” This special `Edition` represents the best available bibliographic metadata for the book.
- Stores information that has been aggregated from multiple sources and summarized:
    - Subject matter classification (aggregated from `Classifications`)
    - Intended audience (aggregated from `Classifications`)
    - Fiction/nonfiction status (aggregated from `Classifications`)
    - Popularity (aggregated from `Measurements`)
    - The best available summary (aggregated from `Resources`)
- May be referenced by multiple `CustomListEntries` and/or `CachedFeeds`.
- May participate in many `WorkGenre` assignments. `WorkGenre` is a simple join table that tracks the assignment of `Works` to `Genres`.

Custom lists

A CustomList is a list of books, typically grouped by a criterion such as genre, subject, bestseller status, etc., which a librarian has compiled in the admin interface. Each CustomList is associated with, and presented to patrons in the front-end as, one Lane. A CustomList has at least one CustomListEntry, each of which refers to a particular Work.

Libraries

`Library`

A library represents some organization that serves a distinct set of patrons.

Each Library can have:

```
* one or more `Collections`.
* one or more `CustomLists`.
* one or more Lanes, each of which is associated with one CachedFeed.
* one or more Admins.
```

`Admin`

Admins are people such as librarians who have access to the admin interface (via accounts in the circulation manager). An `Admin` is associated with a particular `Library` through `AdminRole`. An Admin may have more than one `AdminRole`. The `AdminRoles` are:

- Librarian
- SitewideLibrarian
- LibraryManager
- SitewideLibraryManager
- SystemAdmin

`Lane`

A library groups its books together using `Lanes`. A `Lane` may group books by any combination of these criteria:

- Genre ("Science Fiction")
- Fiction status ("Nonfiction")
- Audience ("Young Adult")
- Target age ("Young Readers")
- Language ("Spanish")
- Media ("Audiobooks")
- Membership on a specific `CustomList` ("Best Sellers")

A lane may have many `CachedFeeds`.

`CachedFeed`

A `CachedFeed` is a pregenerated OPDS document that's stored in the database to serve future client requests. If a `CachedFeed` can be used, it greatly improves patron-visible response time.

Any OPDS feed served by the circulation manager can be cached. This includes the various feeds served as a patron browses a `Lane`, but it also includes the feeds of books by a given author, books in a given series, and recommendations from sources like NoveList.

Patrons

`Patron`

A `Patron` object represents a human being who is a patron of some `Library`. More precisely, a `Patron` object represents that person's library card. A human being with cards for multiple libraries will have multiple `Patron` objects.

Most of the information in the `Patron` record comes from the library's ILS. The most important pieces of information are the three fields used to uniquely identify a patron:

- `external_identifier` - A permanent unique identifier for this patron's ILS record. The patron probably does not know their own `external_identifier`. We keep track of this so that you don't lose your loans and holds when you get a new library card.
- `authorization_identifier` - A nonpermanent unique identifier for the patron, usually numeric. This identifier is printed on their physical library card and used by the patron to identify themselves to the library. Libraries may call this by different terms: a "card number", a "barcode", etc.
- `username` - A nonpermanent unique identifier chosen by the patron themselves as a shorthand method of identification. Most libraries don't give their patrons usernames as distinct from their authorization identifiers, but some do. If present, this is generally alphanumeric.

`Loan` and `Hold`

These classes are nearly identical. They represent a patron's relationship with a `LicensePool` -- either they have the right to read the book right now (`Loa`) or they're waiting in line for the chance to read it (`Hold`).

`Credential`

The `Patron` object keeps track of unique identifiers that identify a patron to their ILS. Other identifiers are stored in `Credential` objects.

One major type of credential is the “Identifier for Adobe Account ID purposes”. This is an alias provided to Adobe (through the Short Client Token system) whenever the patron needs to activate a mobile device with their Adobe ID.

Another major type of credential is the “OAuth Token”. This is a temporary token granted by an ebook vendor such as Overdrive. It gives the circulation manager the ability to take action on the patron's behalf, e.g. by borrowing books or placing holds.

A Credential may have associated `DRMDeviceIdentifiers`. This is used to keep track of the device IDs associated with a patron's Adobe ID. This makes the ACS Device Management Protocol possible.

`Annotation`

A patron in the process of reading a book has a "last reading position" -- the place where they left off. If a patron closes and reopens a book, they expect the book to open their last reading position, not to the beginning of the book.

An `Annotation` stores a `Patron`'s last reading position for a `LicensePool`. If the patron creates bookmarks or highlights text, those are also stored as `Annotations`.

Annotations are synced between client and server using the Web Annotation Protocol. A patron must opt in before their annotations are synchronized with the circulation manager. A patron's decision to opt-in or not is tracked in the field `Patron.synchronize_annotations`.

Site configuration

`ExternalIntegration`

A ConfigurationSetting holds information about an extra piece of site configuration. A ConfigurationSetting may be associated with an ExternalIntegration, a Library, both, or neither.

`ConfigurationSetting`

An ExternalIntegration contains the configuration for connecting to a third-party API. Commonly used third-party APIs include the metadata wrangler, DataSources that require protocols, authentication services, storage services, and search providers.

Background processes

- A Timestamp provides a record of when a Monitor was run.
- A CoverageRecord provides a record of any processes that have been performed on a book (referred to via its Identifier)
- A WorkCoverageRecord provides a record of any processes that have been performed on a Work (similar to what CoverageRecord does for Identifiers).