

CATE - Creating a Taxonomic e-Science

Reference Documentation

Ben Clark

CATE - Creating a Taxonomic e-Science: Reference Documentation

Ben Clark

1.0

Table of Contents

- I. Architectural Overview 1
 - 1. Database, Persistence Layer, Free-Text Search & Versioning 4
 - 2. Controller & View Layer 5
 - 3. CATE Web Application 6

List of Figures

1. CATE is a web application which consists of four separate layers or modules. Model data objects are exchanged between layers. Cross-cutting aspects, such as security, transactions, and caching are shared across layers. 2

Part I. Architectural Overview

Layers

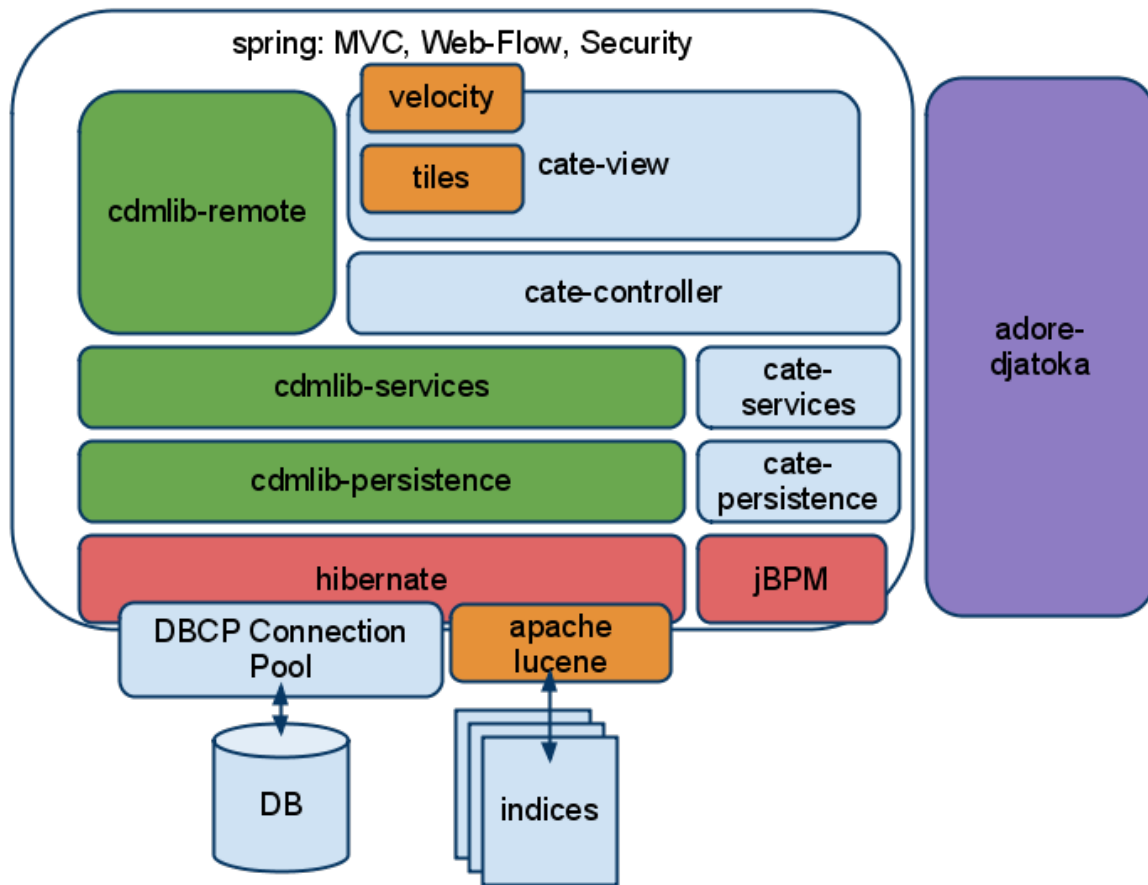
CATE is a java web-application. It relies heavily upon the following technologies:

- The Spring Framework (<http://www.springframework.org>), including Spring MVC
- The Hibernate Object-Relational Mapping Tool (<http://www.hibernate.org>), including hibernate-annotations, hibernate-search, and hibernate-tools
- The JBoss Business Process Management Library (<http://www.jbpm.org>)
- Apache Tiles 2 templating framework (<http://tiles.apache.org>)
- Apache Velocity templating engine (<http://velocity.apache.org>)
- JAXB2, the Java API for XML Binding (<http://jaxb.dev.java.net/>)
- The dojo javascript library (<http://www.dojotoolkit.org>)
- The EDIT Common Data Model Java Library (<http://wp5.e-taxonomy.eu>)

CATE is built using Apache maven 2, a build manager. The project has also produced some perl code designed to extract data from databases produced by the content teams and write the data into a set of xml document that validate against the cate xml schema packaged in a module called CATE.

This section gives an overview of the web-application, the installed instances of CATE, cate-araceae.org and cate-sphingidae.org, and the artifacts produced by the project.

Figure 1. CATE is a web application which consists of four separate layers or modules. Model data objects are exchanged between layers. Cross-cutting aspects, such as security, transactions, and caching are shared across layers.



CATE consists of four major components:

- The persistence layer, which provides access to a local database, and also provides components which proxy remote resources: cate-persistence
- The service layer, which provides a facade to the persistence layer, some business logic, and workflow: cate-service
- The controller layer, which handles http requests, calls methods in the service layer, and routs data to be rendered in a view: cate-controller
- The view layer, which renders java objects as xml, json, and html: cate-view

In addition, there are several other packages, which encompass

- The domain model, object-relational mapping, xml bindings: cate-model
- Javascript Widgets used in the view layer: cate-js
- Command line tools for data import, sitemap generation, and distribution map (image) generation: cate-tools
- The web-application: cate-web

-
- A customized maven site skin, branded for the cate project: cate-site

CATE is currently hosted on the gForge site at the National e-Science Centre. Documentation (including this document) can be found at <http://www.cate-project.org>.

Building CATE

CATE relies upon the maven build system to compile and package the web application. Each layer is packaged as a separate jar file by maven. These are then combined into a single web application archive (war) artifact by the cate-web project. The CATE araceae and CATE sphingidae sites require a few extra resources (mainly content and branding), and some specific configuration (database details, web application root keys), and represent a customized instance of cate-web, replacing a few specific properties and files where necessary.

The build environment required for CATE is as follows:

- Java Development Kit version 5 or more (CATE makes extensive use of the features of J2SE 5.0, notably annotations, enumerations, and generics).
- Apache Maven 2 (<http://maven.apache.org>)
- A CVS Client (such as Tortoise CVS)

Details about the source repository can be found here . Detailed instructions for installing and running Apache Maven can be found on the Maven project site, but in short, issuing the command **mvn install** will build and install the maven binaries to a local repository.

```
C:\Documents and Settings\ben\My Documents\cate> mvn install  
[INFO] Scanning for projects . . .
```

The cate-araceae and cate-sphingidae web-applications are customized versions of the of the cate web module. To generate cate-araceae or cate-sphingidae, set the desired properties (db connection url, username, password, webapp theme name, lsid authority name etc) in maven. The best way to do this is to set all the properties in a profile in your settings.xml file, and then enable the desired profile from the commandline thus **mvn install -Pprofilename** to build cate-araceae or cate-sphingidae without exposing these application specific parameters.

Chapter 1. Database, Persistence Layer, Free-Text Search & Versioning

CATE is based on a MySQL database, although being hibernate based, it could use a different database server if such a thing was necessary (in theory). The database schema is generated by hibernate initially, although hbm2ddl does not map columns properly, and these have had to be tweaked by hand. The ddl files for generating a blank database can be found in `cate-server/root/data/* .ddl`, where `001_cate.ddl` generates the main tables, and `002_jbpm.ddl` generates the tables for jBPM. `003_acl.ddl` is the schema for Spring Security's Access Control List implementation (not currently used).

The persistence layer uses a patched version of hibernate 3.4.0 (there is an outstanding bug in the trunk of hibernate relating to subselects that renders hibernate-envers unusable). Hibernate configuration files live in `cate-model/src/main/resources/org/cateproject/model/hibernate.cfg.xml` with the properties file being found in `WEB-INF/classes` (i.e. in the root of the classpath). The properties in the `hibernate.properties` file are also picked up by the spring application context (using `PropertyPlaceholderConfigurer`) and are added to other beans, like the `datasource`. CATE currently uses the Apache commons Database Connection Pool component.

Almost all of the persistence and service layer is based on the EDIT Common Data Model Java Library (currently about to be released as version 2.1, although not in their maven repository just yet). In addition, CATE has a couple of DAO's that act as a facade over SOAP services, using apache axis (consequently, the CATE web application needs to be allowed to make outgoing network connections to the Biodiversity Heritage Library, and to Genbank). It is likely that these components will be turned into generic components and committed to the EDIT repository in the medium term. There are also a couple of DAO's specific to CATE, to handle the persistence of jBPM tasks and processes, which the CDM doesn't cover.

The CDM uses the hibernate-search library to provide free-text searching functionality. This library integrates the apache Lucene search API with hibernate, and creates lucene indices (files) that are synchronized with the database. These files live in `WEB-INF/indices`. In addition, the CDM uses the hibernate-envers auditing library to version the objects in CATE. Both hibernate-search, and hibernate-envers work using hibernate listeners that react to lifecycle events. These are configured within the hibernate configuration file, and work transparently from the perspective of the application.

The CATE service layer itself is relatively shallow, and consists of a couple of custom formatters, and a few custom services. As with the DAOs, some of the remaining services may be committed to the EDIT repository in the medium term. Transactions are managed by hibernate and spring using `@Transactional` annotations on services for the main part. Once objects are returned from the service layer the transaction is closed and the object is detached (in the hibernate sense, meaning that any calls to uninitialized properties will give rise to a `LazyInitializationException`).

The persistence layer is configured using the files in `cate-persistence/src/main/resources/org/cateproject/persistence/`, with the files called `applicationContext*.xml` being spring application context files, and the files called `jbpm.*` and the files in the `workflows` subdirectory being related to jBPM. The service layer does not require any configuration, as the services are annotated with the spring `@Service` stereotype.

Chapter 2. Controller & View Layer

The controller layer is based around a controller-per-resource pattern, so there are controllers for taxa, names, references, specimens, descriptions, agents, terms etc. In addition there is a smaller number of supporting controllers. These controllers are stateless and use Spring MVC, in particular, the annotation-based controller configuration. At the moment the controllers map to uris that end with `*.do` but in the short/medium term the controller layer will be refactored to conform to a more RESTful approach, and will aim to conform with the CDM REST Service 2.0 API.

In addition to using stateless controllers for serving requests for data, CATE uses the Spring Web Flow API to allow users to edit data by following non-linear paths between related data items. As with many things in CATE, the flows are modularized into a flow-per-resource, with flows launching subflows when a user wants to edit a related item without committing first. Currently there is no pessimistic locking of data. This may have to be implemented in the future.

The controller layer does not, in general use separate data transfer objects (DTOs), instead using property editors to bind data in a request to CDM model objects. This is not always very easy, but leads to a much simpler situation overall as there are no DTO's to keep in sync with the domain model. CATE currently uses a couple of custom interceptors to apply global behaviour to the web requests. One is a `ContentNegotiationHandlerInterceptor`, which is likely to be replaced with the standard spring `ContentNegotiationViewResolver`. The second is the `AuditEventContextHandlerInterceptor` that sets the audit event context based on a parameter in the request (`view`) if present, or retrieves it from the session, if it exists, or sets it to the current view, if all else fails.

Being the primary entry point for data into CATE there is also some data validation logic in this layer. This is likely to be ported to the CDM Java Library as part of an upcoming contract.

The view layer in CATE is also based on Spring MVC, and uses Apache Tiles and Apache Velocity to render content as HTML. In general, CATE tries to return a single object, or a couple of objects per request. For the HTML pages, several objects are usually needed to populate the navigation bar etc, and these are retrieved using a Tiles `ViewPreparer`. There is some caching of this content for efficiency. Also there is a customized `VelocityToolboxView` that inserts objects from the spring applicationcontext into the velocity context as tools.

The controller layer is mainly configured using annotations, but a few beans are declared using XML in `cate-controller/src/main/resources/org/cateproject/controller/applicationContext.xml`. The view layer is configured using `cate-view/src/main/resources/org/cateproject/view/applicationContext.xml` plus `views.xml` from the same directory. There is a standard cate-project theme, plus cate-araceae and cate-sphingidae specific themes in `themes`, Apache Tiles definitions in `tiles`, and velocity templates, layouts and macros in `velocity`. The static files (images and css) are in `cate-view/src/static`.

Chapter 3. CATE Web Application

The CATE Web application is a standard Spring DispatcherServlet called `cate`. It follows the standard conventions for Spring, and is configured using a `WEB-INF/cate-servlet.xml`. Any specific configuration is in `WEB-INF/classes/org/cateproject/*` and subdirectories. In particular, the application context is configured in `WEB-INF/classes/org/cateproject/web/applicationContext.xml`. CATE uses the spring security library for authentication and authorization. This is configured in `WEB-INF/classes/org/cateproject/web/applicationContext-security.xml`.

In addition to the CATE web application, CATE uses an instance of the `adore djatoka` JPEG 2000 image server to serve images (configured within the same `Host` element, mounted under `/images`). The CATE exemplar sites use the `apache httpd` web server to serve static content, and pass all requests for dynamic data onto `tomcat` using `mod_jk`.