cPath Architectural Overview

Last Updated: March 1, 2004

I. Introduction

This document provides an architectural overview of the cPath database and web application. Information about the database schema, data access objects, servlet architecture, configuration, and unit testing is provided.

II. Tools and Libraries

cPath is currently built with the following open source tools and libraries:

- **Apache Tomcat**: open source servlet/jsp engine. Information available at: http://jakarta.apache.org/tomcat/.
- **Ant**: used for the cPath build and deployment process. Information available at: http://ant.apache.org/.
- **Castor**: Java/XML Binding Framework. Information available at: http://www.castor.org/.
- **Checkstyle**: open source toolkit for checking coding conventions. cPath adheres to the Sun Java coding conventions. Information available at: http://checkstyle.sourceforge.net/.
- **CVS**: used for source control and revision tracking. Information available at: http://www.cvshome.org/.
- **Jakarta Database Connection Pool (DBCP):** provides a database connection pool for connecting to relational databases. Information available at: http://jakarta.apache.org/commons/dbcp/index.html
- **JDOM**: Java XML API. Information available at: http://www.jdom.org.
- **JUnit**: used for all cPath unit tests. Information available at: http://www.junit.org.
- **Log4j**: open source logging framework. Information available at: http://logging.apache.org/log4j/docs/.
- **Lucene**: open source text indexer; provides cPath full text search functionality. Information available at: http://jakarta.apache.org/lucene/docs/index.html.
- **Maven:** Maven is a "Java project management and project comprehension tool". We currently use Maven to automatically generate the cPath Developer site every 24 hours.
- **MySQL**: Backend database is built in MySQL. Information available at: http://www.mysql.com.

- **Struts**: open source framework for building web applications. Information available at: http://jakarta.apache.org/struts/.
- **Xerces**: fast, validating XML Parser. Information available at: http://xml.apache.org/xerces-j/.

III. Architectural Overview

cPath is a database driven web site that uses a traditional three-tier architecture (see Figure 1). The first tier consists of client applications, including web browsers and other software applications, such as Cytoscape. Web browser clients communicate with cPath by issuing http requests, and receiving HTML in response. Other client applications, such as Cytoscape communicate with cPath by issuing http requests, and receiving XML in response. The second tier consists of Java code running in the open source Tomcat servlet engine. This layer consists of core Java classes, Java Servlets, and Java Server Pages. The third tier consists of a backend database running mySQL. The mySQL database stores individual pieces of data, as well as XML document fragments, and we communicate with mySQL via JDBC.

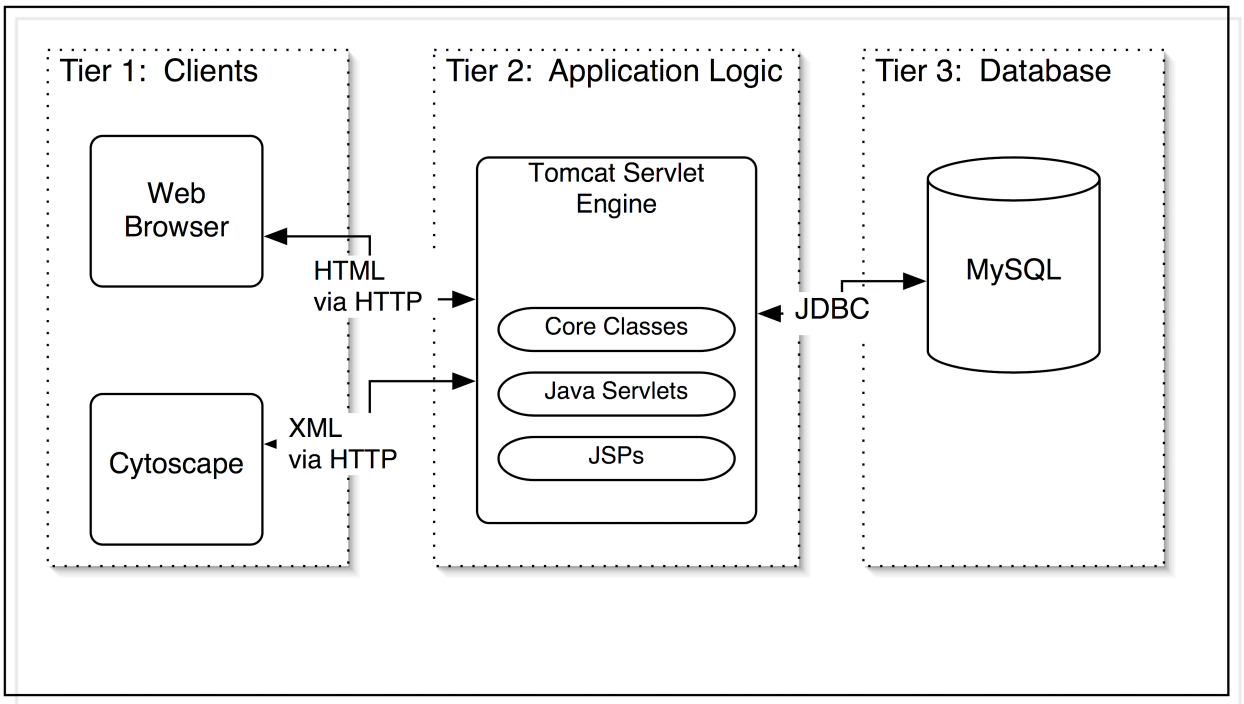

Figure 1: cPath Three-Tier Architecture.

In the discussion that follows, we work backwards by first describing the database schema, and then moving our way to the application logic.

IV. cPath Database Schema

All cPath data is stored in mySQL. The cPath database schema consists of eight relational tables. These tables are grouped into four sets:

- **import table**: used for importing new data into cPath.
- **core entity tables**: used to store core entities, such as interactors and interactions, and internal/external links.
- **external database tables**: used to store information about external databases, such as SWISS-PROT, NCBI, etc.
- **administrative tables**: used for logging and caching purposes.

Details on each set of tables is provided in the sections below.

A. Import Table

The import table contains information about XML/text records, which are scheduled for import into cPath. For example, if an administrator wants to import a new PSI-MI file, the XML data is first loaded into the import table, where it is logged, and recorded. From here, the XML document is parsed and chopped into its constituent parts and loaded into the core cPath tables. The import table contains the following structure:

import	
IMPORT_ID	*Primary ID*
DESC	*Record Description*
DOC_BLOB	*Document Text/XML*
DOC_MD5	*MD5 Fingerprint*
STATUS	*Record Status*
CREATE_TIME	*Timestamp Created*
UPDATE_TIME	*Timestamp Updated*
EX_DB_ID	*Reference to External DB_ID*
LO_ID	*Reference to LinkedOut ID*

B. Core cPath Tables

The cPath core consists of three tables: cpath, internal_link, and external_link. The cpath table contains core entities, such as interactors and interactions. Each entity record contains a short name, description and type. Type must be specified as either: PHYSICAL_ENTITY or INTERACTION. Each record also contains an XML document fragment, written in PSI-MI format.[1] To obtain the full information for an entity, the cPath code must extract and parse the specified XML document fragment.

[1] Eventually, we will transition from PSI-MI to BioPax.

The internal_link table stores links between cPath records. For example, a cPath interaction record will specify bidirectional links between the interaction record and all its interactors. The external_link table records links to external databases, such as SWISS-PROT, NCBI, etc. Information about these external databases is provided in the next section.

The core cPath tables contain the following structure:

cpath **Contains core cPath Entities**	
CPATH_ID	*Primary ID*
NAME	*Entity Name*
DESC	*Entity Description*
TYPE	*Entity Type*
SPEC_TYPE	*Entity SubType*
NCBI_TAX_ID	*NCBI Taxonomy ID*
XML_CONTENT	*XML Document*
CREATE_TIME	*Timestamp Created*
UPDATE_TIME	*Timestamp Updated*

external_link	
EXTERNAL_LINK_ID	*Primary ID*
CPATH_ID	*Reference to CPATH_ID*
EXTERNAL_DB_ID	*Reference to External_DB_ID*
LINKED_TO_ID	*Linked to Identifier*
CREATE_TIME	*Timestamp Created*
UPDATE_TIME	*Timestamp Updated*

internal_link	
INTERNAL_LINK_ID	*Primary ID*
CPATH_ID_A	*Reference to First Entity*
CPATH_ID_B	*Reference to Second Entity*

C. External Database Tables

The external database tables contain information about external databases and URL construction rules for connecting to those databases. The external_db_cv table contains controlled vocabulary terms, which match a specific database. For example, cPath recognizes: SWISS-PROT, SWP, and SWISSPROT, and all these CV terms point to a single SWISS-PROT record in the external_database table. The external_database table contains URLs for connecting to the external database. The token %ID% is replaced dynamically with the linked to ID. For example, here is a URL for connecting to SWISS-PROT:

http://us.expasy.org/cgi-bin/niceprot.pl?%ID%

The external database tables contain the following structure:

external_db	
EXTERNAL_DB_ID	*Primary Key*
NAME	*Database Name*
URL	*URL Construction*
DESC	*Database Description*
FIXED_CV_TERM	*Foreign Key to CV_TERM*
DBDB_ID	*Link to Database of DBs*
DBDB_URL	*Link to Database of DBs*
CREATE_TIME	*Timestamp Created*
UPDATE_TIME	*Timestamp Updated*

external_db_cv	
CV_ID	*Primary Key*
EXTERNAL_DB_ID	*Foreign Key to external_db*
CV_TERM	*Term*

V. cPath Application Logic

A. Package Structure

Application logic for cPath is written in Java, and is specified in the package: **`org.mskcc.pathdb`**. It contains the following package structure:

Package	Description
`action`	Struts Action Classes. Each action class corresponds to a user-initiated action.
`controller`	Classes for processing and validating web service API requests.
`form`	All Struts Action Forms. Each form corresponds to an HTML form.
`lucene`	Classes for connecting to the Lucene Full Text Search engine.
`model`	JavaBean objects which encapsulate cPath records, such as ImportRecord, CPathRecord, etc.
`servlet`	All cPath Servlets.
`sql`	All Database Access code.
`taglib`	All Custom JSP Tags.
`task`	Long-term Tasks, which require mulit-threading

	execution.
test	All JUnit Unit Tests.
tool	All Command Line Utilities and Programs.
util	Misc. Utility classes, such as an XML validator, Cross-Site scripting filter, etc.
xdebug	Live Debugging/Diagnostics Facility.
xmlrpc	XML-RPC Services for uploading new data into cPath.

B. From Browser Request to System Response

Each time a web browser makes a request to cPath, the application logic layer handles the processing of the request, and the generation of dynamic HTML/XML pages. To aid in the processing of web requests, cPath utilizes the open source Struts Framework. The Struts framework provides a number of built-in advantages, including: clean separation of logic and presentation, form validation, centralization of request handling, and centralization of exception handling.

Flow through the application layer proceeds as follows (see Figure 2):

- All user requests go through the central **`cPathServlet`** class. This class provides a central spot for all cPath initialization and configuration.
- Based on the URL requested, Struts will check the Struts.xml configuration file, and determine which action class is invoked. Action classes represent user request actions, such as “run a full text search”, or “get all interactions for yeast”.
- All cPath action classes inherit from the **`BaseAction`** class, and all password protected administrator classes inherit from the **`AdminBaseAction`** class (see Figure 3). The base action classes provide a common framework for capturing exceptions, gathering real-time web diagnostics, and protecting specific resources via HTTP authentication.
- Some action classes will require access to the mySQL database. In that case, the action class will make calls to a DAO Object. A DAO, or Data Access Object encapsulates all database access code, and also takes care of handling connections to the database connection pool. There is one DAO object for each cPath table. For example, the import table has an ImportDao class.
- All final HTML construction is completed in Java Server Pages, and custom JSP tags.

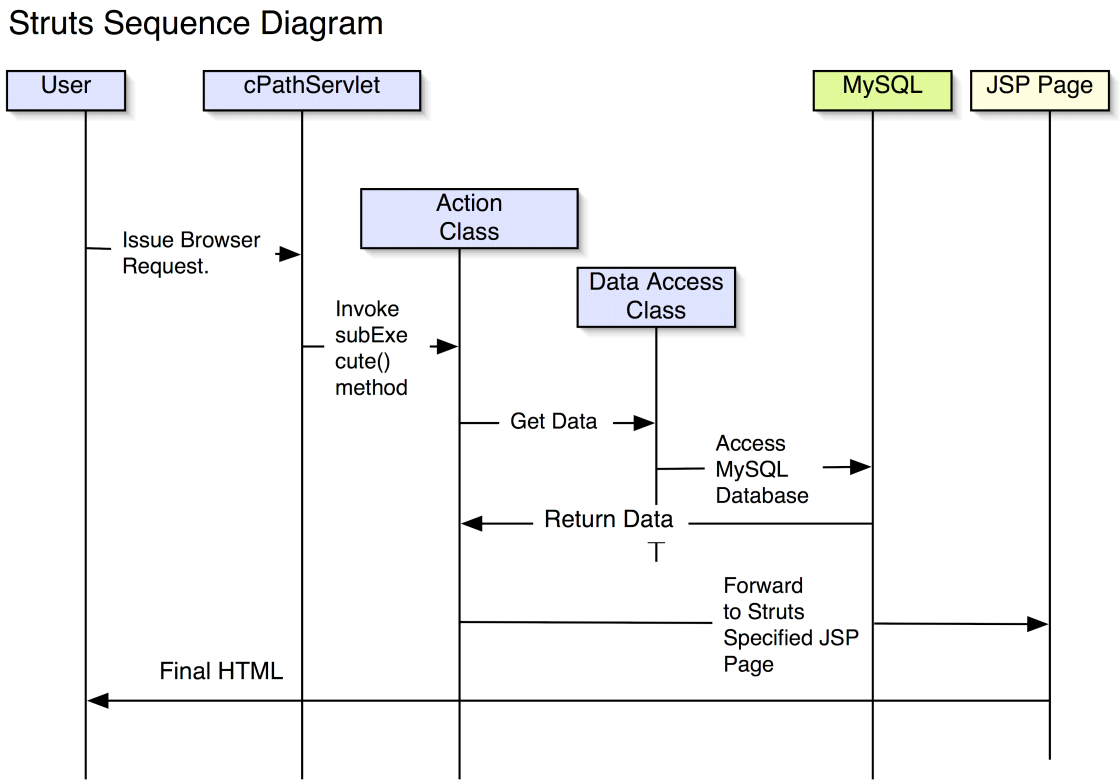

Figure 2: Data flow within the Application Logic Layer.

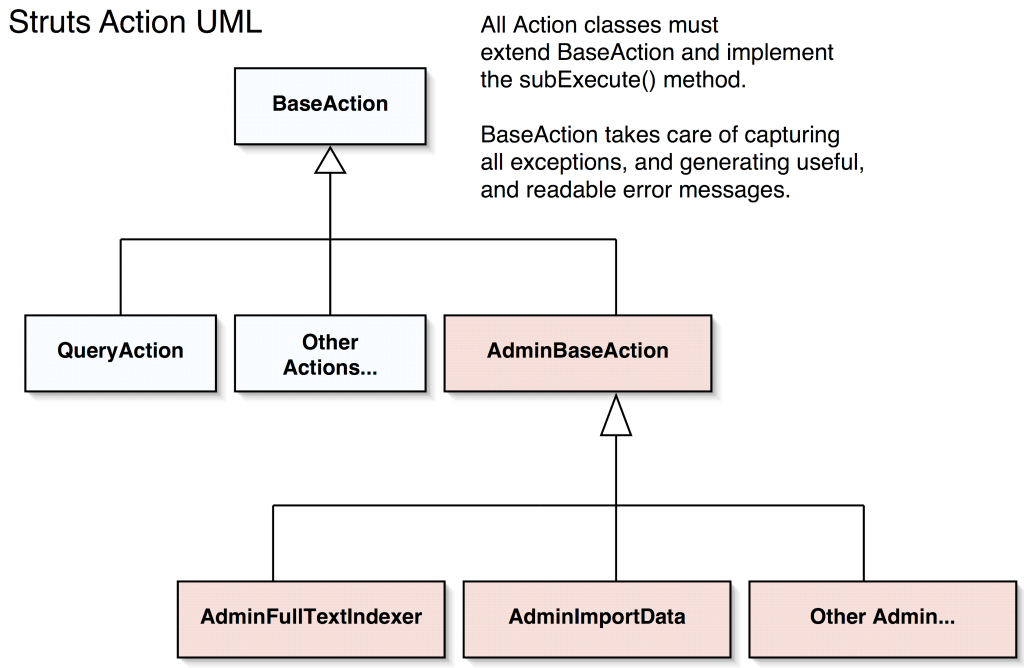

Figure 3: UML Diagram of the Struts Action Classes. Each Action class corresponds to a user-requested action.

C. Style Sheets, JSP Templates and JSP Custom Tags

cPath is built using open source CSS style sheets created at http://style.tigris.org. Tigris provides a cross-browser compatible set of style sheets and open source icons. This enables cPath to have a very professional look, with minimal effort. It also enables us to easily provide a "printer-friendly" version of all cPath pages.

All presentation is done via Java Server Pages (JSPs) and custom JSP tags. The WEB-INF/jsp/global directory contains a number of reusable JSP components which are used throughout the site. These include the following:

JSP Component	Description
header.jsp	Global header component, used on all cPath pages.
footer.jsp	Global footer component, used on all cPath pages.
tabs.jsp	Global tabs component, seen at the top of all cPath pages.
navbar.jsp	Global navigation bar component, seen in left column of all cPath pages.
xdebug.jsp	Global component used to output real-time web diagnostics.

The custom JSP tags provide reusable tags, which can be inserted into any cPath page. All cPath custom tags are defined in: WEB-INF/tag-lib/cbio-taglib. A summary of all custom tags is provided below:

JSP Custom Tag	Description	Example Usage
DiagnosticsTable	Outputs internal tests used in the Admin cPath self-test page.	`<cbio:diagnosticsTable />`
ErrorMessage	Centralizes all display of error messages to the user.	`<cbio:errorMessage throwable= "<%= exception %>"/>`
HtmlTable	Base Class for all Custom JSP Tags which create HTML tables.	Not used directly.
InteractionTable	Displays a list of matching interactions.	`<cbio:interactionTable interactions="<%= interactions %>" protocolRequest="<%= protocolRequest %>"/>`
InteratorTable	Displays all information about a specific interactor.	`<cbio:interactorTable />`
SearchResultsTable	Displays results from a	`<cbio:searchResultsTable`

	full-text search.	`protocolRequest= "<%= protocolRequest %>"/>`
`TaskTable`	Displays all admin tasks (in progress and completed tasks are shown.)	`<cbio:taskTable/>`

D. Web Application Configuration

cPath configuration is centralized in three files.

File	Description
`WEB-INF/web.xml`	The main configuration file for the cPath web application. Here, you can modify the Struts debugging level, the user/name password for connecting to cPath, and the admin user/name password for accessing administrative functionality.
`WEB-INF/struts-config.xml`	The main configuration file for the Struts Framework. This file specifies all the action URLs, HTML form objects, and JSP pages.
`config/cpathResources.properties`	Defines a number of String constants used throughout the cPath site. All user error messages are defined here.

VI. Development Practices

A. Automated Build Process

The cPath build process is automated with Ant. A number of Ant targets are defined:

Target	Description
Main Development Targets	
boot	Performs a complete and clean build.
check	Run CheckStyle on all source code.
clean	Deletes all build files and starts fresh.
compile	Compile all Java sources.

Test Targets	
boot	Loads Bootstrap data into MySQL. Required for running JUnit Tests.
test	Runs JUnit Test Suite.
Local Tomcat Development Targets	
install	Installs cPath to local development servlet container.
list	Lists installed applications on servlet container.
reload	Reloads cPath application on servlet container.
remove	Removes cpath application on servlet container.
Deployment / WAR Targets	
deploy_war	Deploy WAR to cBIO Production server.
war	Creates binary WAR distribution.

B. Coding Standards

cPath adheres to the Sun Java coding standards. These are enforced via the ant checkstyle target. To see if your code adheres to the standards, run “ant check”.

C. Unit Testing

cPath uses the JUnit testing framework. All unit tests are organized into suites, and all suites are organized into **`org.mskcc.pathdb.test.AllTest`**. To run all cPath tests, run “ant test”. To create a new test, create the new test, and add is to the correct package suite.

D. Documentation

cPath uses the open source Maven toolkit for automatically generating the cPath developer site every 24 hours. All documentation is stored in cpath/xdocs. Add new documents to this directory.