Chapter VI

cancer Biomedical Informatics Grid (caBIG™)

cancer Biomedical Informatics Grid

Whole genome sequencing projects that led to the sequencing and assembly of the human genome and scores of other vertebrate and invertebrate genomes have changed the face of biology and medicine forever. The convergence of molecular-scale biological science, high-throughput technologies and large-scale computing has led to an explosive growth in the volume of information that is available to the modern day biomedical scientist. The success of biomedical research in designing effective therapies for the treatment of complex diseases such as cancer is fundamentally dependent on our ability to integrate and assimilate this raw and largely unstructured data from a variety of experimental platforms encompassing the genomics, proteomics, transcriptomics and the pharmacological and clinical domains. It is also increasingly becoming evident that cooperation among research organizations across geographical boundaries and an open sharing of datasets and analytic tools as well as individual expertise and knowledge is critical to the continued advancement of biomedical research towards its goals.

The cancer Biomedical Informatics Grid project or caBIG™ (pronounced see-ay-big) is built on this very premise. We had provided an introduction to the caBIG™ program in Chapter 1. To recap, the caBIG project was launched in July 2003 and is initiated and funded by the United States National Cancer Institute (NCI) under the aegis of the United States National Institutes of Health (NIH). CaBIG™ is a critical

component of NCI's challenge goal of eliminating suffering and death due to cancer by the year 2015. Indeed, CaBIG™ is an effort designed to achieve a level of cross-disciplinary integration that is unprecedented in the history of cancer research. According to NCI Director, Dr. Andrew von Eschenbach, "*...caBIG will become the 'World Wide Web' of cancer research informatics and will accelerate the development of exciting discoveries in all areas of cancer research*". According to the official website (http://cabig.nci.nih.gov/), caBIG is a voluntary, open source, open access initiative that is being designed and built in partnership with the cancer research community across the United States. Since the caBIG pilot program was launched, more than 50 interested NCI-designated cancer centers and more than 800 individuals have participated in the development of the vision, approach and structure of caBIG..

Structure and Organization of caBIG™

caBIG™ participating institutions are organized into Workspaces that are devoted to specific domains of interest relevant to cancer research. Currently, there are four Domain Workspaces, two Cross Cutting Workspaces and three Strategic Level Workspaces. **Table 6.1** provides names and descriptions of the various Workspaces under caBIG™.

Table 6.1. Structure of caBIG™

Workspace name	Purpose
Domain Workspaces	
Clinical Trial Management Systems Workspace	Modular development of tools for the management of clinical trials. These include development of a structured model for protocol representation as well as tools for managing and reporting adverse events that occur during the course of a clinical trial, a laboratory interface module to facilitate automated submission of data to clinical trials systems, a reporting module to submit data electronically to NCI's CDUS (Clinical Data Update System) and the NCI's Clinical Trial Monitoring Service (CTMS) and a financial/billing module to monitor budgets and expenditure in clinical trials. The Workspace is divided into special interest groups for each of these difference activities.
Integrative Cancer Research	Development of modular and interoperable tools and

Workspace	interfaces that provide for integration of clinical and basic research data derived from genomics and proteomics platforms. The Workspace is organized into special interest groups devoted to topics such as Genome Annotation, Microarray Repositories, Pathways Tools, Data Analysis & Statistics, Population Sciences and Translational Tools. Tools being developed under the Workspace include Rproteomics (MALDI-TOF proteomics analysis tool), Gene Ontology Miner (tool for aggregate analysis of gene sets), HapMap (map of haplotypes in human genome), caArray (cancer microarray data management system), Distance Weighted Discrimination (microarray data analysis integrator), Visual and Statistical Data Analyzer (multivariate statistical visualization tool for the analysis of complex data), FunctionExpress (integrated analysis and visualization of microarray data), Quantitative Pathway Analysis in Cancer (pathway modeling and analysis tool), TrAPSS (disease gene mutation discovery and analysis tool), etc.
In Vivo Imaging Workspace	Development of tools to share and integrate the wealth of information provided by in vivo imaging with other types of data. The in vivo imaging technologies and modalities will include systems for research and clinical imaging of live patients and animals (including single-cell organisms) used as model systems for human disease.
Tissue Banks and Pathology Tools Workspace	Development and integration of tissue bank and pathology tools and infrastructure components to enable researchers to locate and analyze tissue specimens for use in cancer research based on tissue, clinical, and genomic characteristics. Tools created under this Workspace include a standard Biospecimen Object Model and suite of tools to facilitate specimen management, annotation and sharing. Specific applications being developed are a specimen inventory and tracking system (caTISSUE Core), a mapping module to get data from tumor registries and clinical anatomy laboratory information systems (caTISSUE Clinical Annotation Engine) and a cancer Text Information Extraction System to automate the process of coding, storing and retrieving data from free-text Pathology Reports (caTIES).

Cross Cutting Workspaces	
Architecture Workspace	Development of tools to ensure consistent application of caBIG™ principles by the large caBIG™ developer community and to meet the caBIG™ program goals of data sharing and interoperability on the grid. Activities include formulating guidelines and definitions for caBIG™ participants to evaluate the maturity level of potential caBIG™ systems and applications (caBIG™ Compatibility Guidelines), development of the grid infrastructure to support the caBIG™ community (caGrid), development of a comprehensive grid security infrastructure for managing federated authentication and authorization in caBIG™, etc.
Vocabularies and Common Data Elements Workspace	Development of policies and guidelines to evaluate and integrate systems based on vocabulary and ontology content as well as software systems for content delivery. Among the major deliverables of this Workspace are the Common Data Elements (CDE) Governance Model to manage the development and administration of CDEs in the Domain Workspaces, data standards approval guidelines for defining the procedures for reviewing and approving data standards, procedures for review and approval of new VCDE content to provide for overall standardization of CDEs within caBIG™, a vocabularies deployment document which lists vocabulary standards consistent with caBIG™ compatibility requirements and LexGrid, a vocabulary server that can be accessed through a well-structured application programming interface (API) capable of accessing and distributing vocabularies as commodity resources.
Strategic Level Working Groups	
Strategic Planning Working Group	Development of strategic planning and vision guidelines in support of the caBIG™ Oversight Board. Activities include creating white papers and planning documents that help define the strategic goals for each individual Workspace as well as for the overall caBIG™ project, along with metrics to measure the success of defined objectives.
Data Sharing and Intellectual Capital	Development of policies and white papers to clarify caBIG's stand on issues surrounding data sharing and intellectual property. Some of the major activities of

	the Working Group include development of guidelines and a model agreement for use by caBIG™ participant institutions to distribute caBIG™ software and related documentation, a caBIG™ publications policy, guidelines on best practices and model agreements for the sharing of data and of biospecimens, reagents and other materials, and a white paper on the de-identification of patient data.
Training Working Group	Development of a caCORE curriculum designed to prepare caBIG™ participants to operate and use the NCI resources such as Enterprise Vocabulary Services (EVS), Cancer Data Standards Repository (caDSR), and Cancer Bioinformatics Infrastructure Objects (caBIO) as well as creating templates and guidelines for caBIG™ documentation and training and organizing boot camps to impart training on caBIG™ technologies.

Further details on caBIG™, its constituent Workspaces and Working Groups and their objectives are available on the WWW at http://cabig.nci.nih.gov/. The ultimate aim of caBIG™ is to enable researchers to collect comprehensive data about cancer in a standardized manner, to enable the study of cancer data as a whole, thereby accelerating the pace of cancer research.

The purpose of this chapter is to not only inform the readers of current efforts in the area of cancer research but also provide knowledge about the technologies that are being developed as an integral part of the effort so that the biomedical and the computer scientists among us can begin using them and in so doing, contribute to their continued development that will ultimately lead to better healthcare solutions and better care and treatments for patients. We will begin by reviewing a few tools and technologies that are relevant to our understanding of how information technologies can assist biomedical research.

Data Integration and ETL

Biomedical researchers routinely need to access and cross-reference sequence and related annotation data from a wide variety of sources such as *PubMed*, *Entrez Gene* (previously called *LocusLink*), *Gene Ontology*

(GO), *UniGene*, *Swiss-Prot*, *Ensembl*, *HomoloGene*, *UniSTS*, etc. Because difference data sources use different formats, it is not easy to compare and combine data from these sources unless they are converted into a common format. Data in UniGene, for example, is presented in text format; data in the GO database is described in an XML format and data in Entrez Gene is available in binary Abstract Syntax Notation number One (ASN.1) format.

CaBIG™ also handles a wide array of data sources, types and formats, from a number of different public domain sources since one of its major goals is to enable access to and sharing of translational research data between cancer researchers. In order to facilitate integration of diverse data types, tools that perform what is known as *Extract*, *Transform* and *Load* (ETL) functions are used. These tools convert data in different formats into a common, standard, usable format. The first step - Extraction - is the part that establishes access to the external database or source that contains the data of interest. The next step - Transformation – analyzes the original data format and converts it to fit with the format of the target repository. For example, information on a gene id can be coded as an XML tag in the form:

<gene id="<my id>"/>

or, as an SQL varchar(64) which means a string of variable length with a maximum size of 64 characters, and other formats. When we design the ETL strategy, we will first create business rules and define the format of the gene identifier that will be used to store that information in the target repository. If this is the SQL varchar(64) type, we will transform data from sources that use a different format into this pre-selected target gene ID format.

The Transformation step can also involve a data-cleansing step to eliminate bad or duplicate entries from input data sources. This process can be done after transforming the data or just before adding data to the target repository. The last step - Load – gets the transformed data loaded into a data repository or a data warehouse, which is optimized to enable faster access to the stored data. An additional step after Extract-Transform-Load is Transportation, which facilitates transport of the formatted data from its current location to the defined location, before it is processed or used further.

A number of open source ETL tools are available, for example, Kettle (available from http://www.kettle.be/), Octopus (available from http://www.enhydra.org/tech/octopus/index.html), and others. Examples of ETL tools being developed under the caBIG™ program include *cancer Function Express tool* (caFE), which annotates individual *probe* sequences (short DNA sequences that represent individual genes or transcripts from a particular genome) on *microarray chips* (arrays of thousands of individual probe sequences embedded on a substrate to detect the presence of specific genes or transcripts in a given genome by hybridizing probes with nucleic acids from the test sample) using data from a number of NCBI and other public databases.

cancer Common Ontologic Representation Environment (caCORE)

A critical component of the partnership between NCI and the Cancer Centers in building the biomedical informatics grid is NCI's Center for Bioinformatics (NCICB). NCICB's mission is to create a close knit and cooperative cancer research community and an interoperable federation of informatics resources covering all aspects of cancer research. NCICB is providing critical support for caBIG through the development of *caCORE*, an open source semantic enterprise architecture for NCI-supported research information systems for genomic and clinical research. A large number of NCI applications such as the *Cancer Molecular Analysis Project* (CMAP), the *Cancer Models Database* (caMOD), and *Gene Expression Data Portal* (GEDP) are directly supported by caCORE. A list of publicly available data sources in the caCORE database is provided in **Table 6.2**. More information on caCORE is available on the NCI caCORE.

Table 6.2. caCORE data sources

Name	Purpose
CGAP Cancer Genome Anatomy Project (CGAP)	Determine the gene expression profiles of normal, precancer, and cancer cells, leading eventually to improved detection, diagnosis, and treatment for the patient.
CGAP Genetic Annotation Initiative (GAI)	Develop a systematic and comprehensive notation of variations in the DNA sequences of each cancer-related gene.
Mouse Models of Human Cancers Consortium	Derive and characterize mouse models, and to generate resources, information, and innovative

(MMHCC)	approaches to the application of mouse models in cancer research.
Cancer Molecular Analysis Project (CMAP)	Facilitate the identification and evaluation of molecular targets in cancer by integrating comprehensive molecular characterizations of cancer.
Gene Expression Data Portal (GEDP)	Provide access to microarray data as well as online data annotation and analysis tools.
Integrated Molecular Analysis of Genomes and their Expression (IMAGE) Consortium	Establish a common resource of publicly available cDNA libraries for access to sequence, map, and expression data.

caCORE is built on the principle of *Model Driven Architecture*, which is a way to organize and manage enterprise architectures supported by automated tools and services for both defining the models and facilitating transformations between different types of models. CaCORE is built on an *n-tier architecture* model and provides *open source Application Programming Interfaces (APIs)* which allow for easy access to data by applications.

The main components of caCORE are:

- *Enterprise Vocabulary Services* (EVS): Controlled vocabulary resources (such as the *NCI thesaurus* and *metathesaurus*) for the life sciences domain that provide a context driven semantic basis for the construction of data elements, classes, and objects.

- *Cancer Data Standards Repository* (caDSR): A metadata registry based upon the ISO/IEC11179 standard that renders research data on cancer reusable and interoperable. The 11179 standard created by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) specifies the criteria for metadata that are necessary to describe data, as well as the management and administration of that metadata in a metadata registry.

- Cancer Bioinformatics Infrastructure Objects (caBIO): A suite of software, vocabulary, and metadata models for cancer research.

We will explore caBIO objects in further detail in this chapter.

Cancer Bioinformatics Infrastructure Objects (caBIO)

caBIO objects constitute the primary programming interface to caCORE. caBIO objects are implemented using Java and *Java Bean* technology, and model the behavior of hierarchies of biological entities such as *genes*, *sequences* and *chromosomes*, their constituent molecular forms such as *Single Nucleotide Polymorphisms* (SNPs, a single nucleotide difference at a defined location within an individual's DNA sequence), and other entities such as *clones*, *libraries*, *agents*, *pathways*, *tissues* and diseases. A representative list of objects and their descriptions are shown in **Table 6.3**.

Table 6.3. caBIO domain objects

Object name	Description
Gene	The basic physical and functional unit of heredity. Gene objects are the effective portal to most of the genomic information provided by the caBIO data services such as organs, diseases, chromosomes, pathways, sequence data, and expression experiments.
GeneAlias	An alternative name for a gene; provides descriptive information about the gene (as it is known by this alias), as well as access to the Gene object it refers to.
GoOntology	An object providing entry to a Gene object's position in the Gene Ontology Consortium's controlled vocabularies. GoOntology provides access to gene objects corresponding to the ontological term, as well as to ancestor and descendant terms in the ontology tree.
Target	A gene thought to be at the root of a disease etiology and targeted for therapeutic intervention. Defined and used by the CMAP project.
Protein	An object representation of a protein; Protein objects provide access to the encoding gene via its GenBank ID, the taxon in which this instance of the protein occurs, and references to homologous proteins in other species.
Disease	Specifies a disease name and ID; also provides access to ontological relations to other diseases; clinical trial protocolstreating the disease; and specific histologies associated with instances of the disease.
Pathway	An object representation of a molecular/cellular pathway compiled by BioCarta. Pathways are associated with specific Taxon objects, and contain multiple Gene objects, which may be targets for treatment.

Therapeutic agent	A therapeutic agent (drug, intervention therapy) used in a clinical trial protocol.
ClinicalTrialProtocol	The protocol associated with a clinical trial; organizes administrative information about the trial such as Organization ID, participants, phase, etc. provides access to the administered Agents.
Histopathology	An object representing anatomical changes in a diseased tissue sample associated with an expression experiment; captures the relationship between organ and disease.

caBIO provides programmatic access to a variety of open source genomic, biological, and clinical data sources available from the NIH such listed previously (such as Unigene, EntrezGene, etc.) as well as others such as *Biocarta* and clinical trials protocols, etc. caBIO is built upon open source technologies such as Java, *Simple Object Access Protocol* (SOAP, an XML based platform and language independent protocol for exchanging information between applications over the web), *Apache*, *Jakarta Tomcat*, *XML* and *UML*. There are a number of ways that users can access caBIO. Java-based clients communicate with caBIO via the Java API, which contains the domain objects provided by the caBIO.jar file. Non-Java based applications can communicate via SOAP, or by using the caBIO HTTP API and receive objects as XML. caBIO provides access to curated data from multiple sources as described in **Table 6.4**.

Table 6.4. caBIO data sources

NCBI UniGene	Unigene provides a nonredundant partitioning of the genetic sequences contained in GenBank into gene clusters. Each such cluster has a unique UniGene ID and a list of the mRNA and EST sequences that are subsumed by that cluster.
NCBI Entrez Gene (previously called LocusLink)	Entrez Gene contains curated sequence and descriptive information associated with a gene such as gene name, aliases, sequence accession numbers, phenotypes, UniGene cluster IDs, OMIM IDs, gene homologies, associated diseases, map locations, etc.
Gene Ontology (GO) terms	The Gene Ontology Consortium provides a controlled vocabulary for the description of molecular functions, biological processes, and cellular components of gene products.
NCBI HomoloGene	HomoloGene is a resource for curated and calculated gene homologs.
BioCarta pathways	BioCarta provides detailed graphical renderings of

	pathway information concerning apoptosis, cell signalling, cell cycle regulation, immunology, metabolism, and neuroscience, etc.
NCI Cancer Therapy Evaluation Program (CTEP)	CTEP funds an extensive national program of basic and clinical research to evaluate new anti-cancer agents, with a particular emphasis on translational research to elucidate molecular targets and drug mechanisms.
NCI Cancer Models Database (caMOD)	caMOD provides information on animal models of human cancer.

Downloading and Configuring caBIO

caBIO can be downloaded from the NCICB website at:

```
http://ncicb.nci.nih.gov/download/index.jsp
```

Download caCORE3-1.zip file (or the latest available version), unzip to extract the required libraries and save them in an appropriate location making sure that the absolute path to client.jar is declared in the Java classpath. **Fig. 6.1** below shows the caCORE download page.

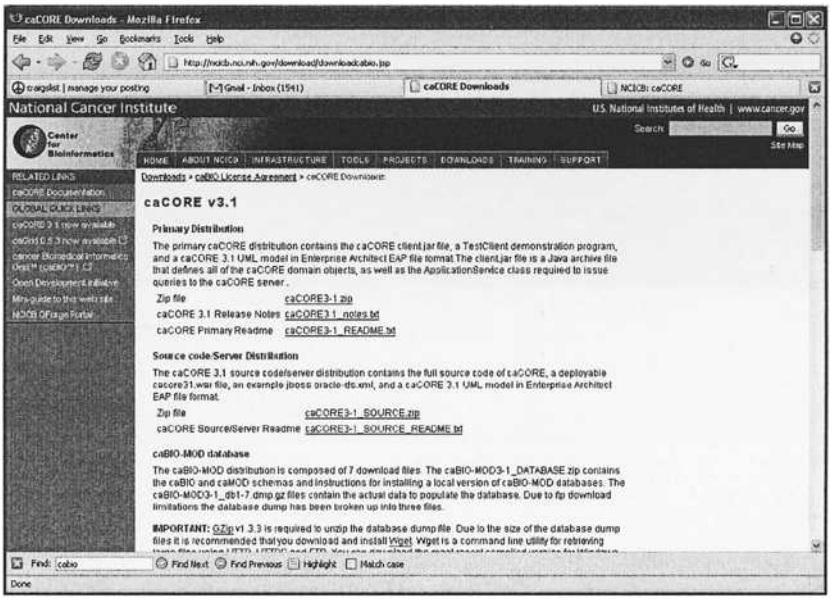

Fig. 6.1. caBIO download page

Now that we have reviewed some of the concepts, technologies and resources available to us from NIH, NCI and other sources, we will create a simple practical application to demonstrate how to integrate the individual isolated bits of data together into a richer, more usable dataset.

Creating the JcaBIO Application

We will create an application based on the caBIO API that we will call JcaBIO to demonstrates how data pertaining to the *Gene* and the *Agent* object can be retrieved using caBIO API. We will create three search functions as outlined below that will define the business logic of the application:

Gene search function: The gene search function will create a report that provides information such as gene name and symbol, Unigene Cluster ID, associated GO terms, gene product name and aliases.

Pathway search function: The pathway search function creates a report that provides information on the pathways that a gene participates in along with a description and a link to the pathway map on BioCarta.

Agent search function: The agent search function creates a report that contains the names of the target(s) that a therapeutic agent binds, the clinical trials that an agent is involved in along with the status, Phase and the name of the institution conducting the trial.

According to this scheme, we will need four command buttons – one each for creating the Gene, Pathway and Agent report and one to clear the report. We will label the command buttons, "Run a Gene Search", "Run a Gene/Pathway Search" and "Run an Agent Search" respectively. We will need a text area to display the reports. We will place this below the command buttons. We will need one text box each to enter the gene name, the agent name and specify the number of reports we want to retrieve for each search. We will place a default value of 10 in the last text box to begin with. We also need a message area to provide the users information on the current state of the application. When the application is launched and when a search is complete, the status will display the "`Ready!`" message. We will place the status bar below the text area.

When the application is initially launched, all the command buttons will be disabled; the command buttons will become available after a valid gene or an agent is entered into the appropriate fields. Only the appropriate command buttons corresponding to the entries will be activated. An entry in the Gene field, for example, will activate the "Run a Gene Search" and "Run a Gene/Pathway Search" buttons while an entry in the Agent field will activate the "Run an Agent Search" button. The Clear button will be activated only after a search has been run and there are results to display.

JcaBIO Classes and Application Structure

The structure of the `JcaBIO` application is shown in **Fig. 6.2**.

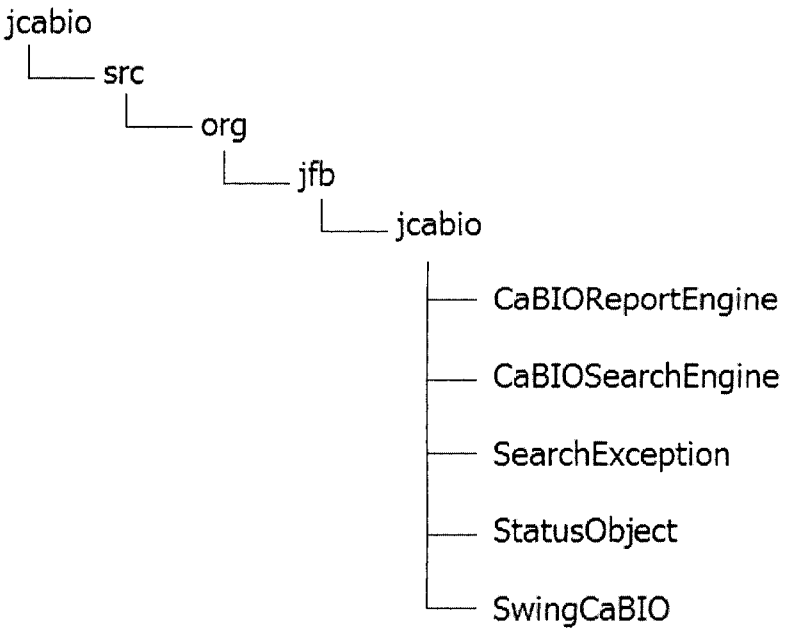

Fig. 6.2. Structure of JcaBIO

A description of the various classes and the corresponding code is as follows:

`SwingCaBIO`: This is the main *Swing* application interface that enables users to send queries and display reports about genes or agents using the caCORE API.

`SearchException`: This class handles exceptions when a search fails.

`StatusObject`: This class stores information on the state of the `CaBIOSearchEngine` or the `CaBIOReportEngine`, which respectively handle the search and the report processes. Using a `StatusObject` instead of a *String* object affords a more generic way of passing the required information. As a result, we have the freedom to modify the `StatusObject` class without having to change the signature of the method that uses this object. We would need to modify only the content of the code as appropriate.

`CaBIOReportEngine`: This is the class that generates the Gene or Agent reports.

`CaBIOSearchEngine`: This class provides the functionality that enable users to perform gene or agent searches.

The application at start up showing the various Swing components is shown in **Fig. 6.3**.

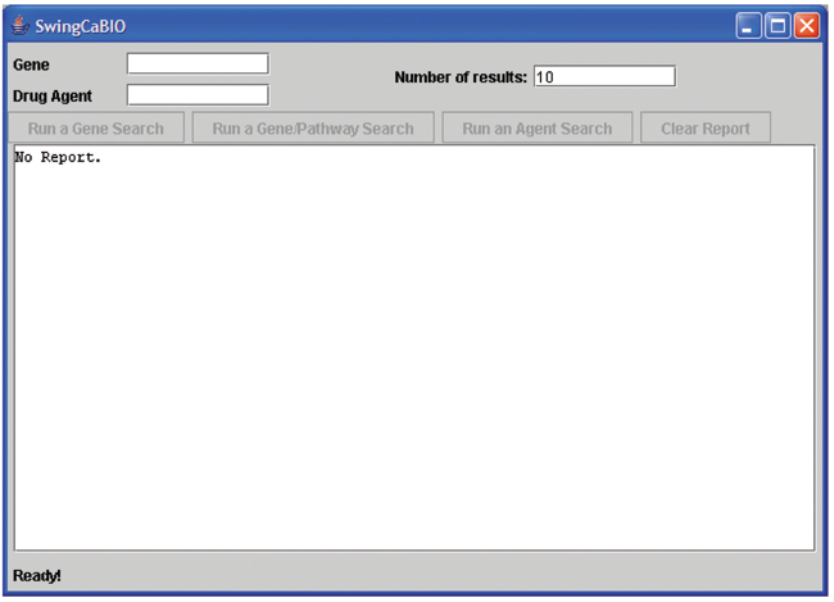

Fig. 6.3. The JcaBIO application at start up

Coding the SwingCaBIO Application

`SwingCaBIO` defines the application interface that the user interacts with to send and retrieve queries using the caCORE API. `SwingCaBIO` is based on the same *Swing* elements and concepts that were described in Chapter 1. `SwingCaBIO` extends *JFrame* in order to generate a basic container for the application. `SwingCaBIO` also extends *DocumentListener* to listen to the *JTextField* objects in order to enable or disable the corresponding buttons that run the report. The constructor `SwingCaBIO()` calls the *super* constructor and adds the *observer* to the Agent search and the report engine. At that point, the application consists of a frame and nothing else built inside. We then call the `init()` function explained below that will build our form for the Gene and Agent searches.

As described earlier, we need three command (search) buttons for running the three custom reports and a text area to display the search results. We use what are called *factory* methods to create the different pieces that are assembled in the `init()` method. *Factory* methods refer to the *Factory Design Pattern*, which specifies a way to create objects without having to know how they are created or assembled. This design allows the developer to change the way the buttons and other Swing components are displayed without interfering with the rest of the Swing components that make the application. The `SwingCaBIO()` method is described below:

```
public SwingCaBIO() throws HeadlessException {
  super();
  AGENT_SEARCH.addObserver(observer);
  REPORT_ENGINE.addObserver(observer);
}

private void init() {
  setTitle("SwingCaBIO");
  final Container contentPane = getContentPane();
  contentPane.setLayout(new BorderLayout());
  JPanel formPanel = createForm();
  JPanel reportPanel = createReportPane();
  statusBar = new JLabel(STATUS_READY);
  statusBar.setBorder(BorderFactory.createEmptyBorder(5, 5,
  5, 5));
  contentPane.add(formPanel, BorderLayout.NORTH);
  contentPane.add(reportPanel, BorderLayout.CENTER);
  contentPane.add(statusBar, BorderLayout.SOUTH);
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  pack();
  setSize(DIMENSION);
```

```
    final Dimension screenSize = Toolkit.getDefaultToolkit().
    getScreenSize();
    setLocation(new Point((screenSize.width - SW_WIDTH) / 2,
    (screenSize.height - SW_HEIGHT) / 2));
    show();
  }
```

The `getNumberOfobjectsForResult()` method handles the total number of results to return for a search. As indicated earlier, we will place a default value of 10 for this.

```
  private static final StatusObject STATUS_REPORT_GENERATED =
    new StatusObject("Report generated!", 10);
    final int len =
    getNumberOfObjectsForResult(genes.length);
    sb.append("Search Results for  '" + genePattern + "' (" +
len + " gene(s) found):\n\n");
    for (int i = 0; i < len; i++) {
      Gene gene = genes[i];
      REPORT_ENGINE.printFullGeneReport(gene, sb, i + 1);
    }
```

We use the `showReport()` method to set the text area with the current *StringBuffer* object containing the report generated. Anytime a Swing object is modified, we need to make sure that the method runs in the *event-dispatching thread* (also called the *AWT thread*) to avoid painting problems. The method checks if we are already in the *event-dispatching thread* before calling the `runnable` object.

```
    private void showReport(final StringBuffer sb) {
        Runnable runnable = new Runnable() {
            public void run() {
                jTextArea.setText(sb.toString());
            }
        };
        if (SwingUtilities.isEventDispatchThread())
            runnable.run();
        else
            SwingUtilities.invokeLater(runnable);
    }
```

The methods `insertUpdate()`, `changeUpdate()` and `removeUpdate()` are methods from the `DocumentListener` interface. We will use these methods to update the buttons according to the values found in the gene and the agent fields.

```
  public void insertUpdate(DocumentEvent event) {
```

```
    updateButtons();
  }

  private void updateButtons() {
    boolean enabled = gene.getText().trim().length() > 0;
    runGenePathwayReport.setEnabled(enabled);
    runFullGeneReport.setEnabled(enabled);
    runTargetAgentReport.setEnabled(agent.getText().trim().
    length() > 0);
  }
```

The three update methods are delegating the treatment of the event to the `updateButtons()` method. The `updateButtons()` method enables or disables buttons according to the status of the search and the corresponding report that is generated. We add a utility method called `errorDump()` to create an error message and update the status bar to alert user about any problems encountered:

```
  private void errorDump(StringBuffer sb, SearchException e)
 {
    sb.delete(0, sb.length());
    sb.append("An error occured!\n\n" +
      e.getEmbedded().getMessage());
    updateStatus(new StatusObject("An error occured!", 5));
  }
```

We include a method to update the status of the search and reporting using the `updateStatus()` method. `updateStatus()` sets the text in the status bar with information on the state of the search. We need to invoke and display the update right away to ensure that the user is alerted as soon as an issue arises. For this reason we have implemented the `invokeAndWait()` instead of the `invokeLater()` method.

These methods force the JVM to invoke the `run()` method of the *Runnable* object passed as an argument, inside the *event-dispatching thread.*

```
      private void updateStatus(final StatusObject
statusObject) {
          Runnable runnable = new Runnable() {
              public void run() {

statusBar.setText(statusObject.getStatusText());
                  if (statusObject.hasTimer()) {
                      new Thread(new Runnable() {
                          public void run() {
                              try {
                                  synchronized (this) {
```

```
this.wait(statusObject.getTimer() * 1000);
                                    }
                                    updateStatusToReady();
                                } catch (InterruptedException
e) {
                                    e.printStackTrace();
                                }
                            }
                        }).start();
                    }
                }
            };
            if (SwingUtilities.isEventDispatchThread()) {
                runnable.run();
            } else {
                try {
                    SwingUtilities.invokeAndWait(runnable);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } catch (InvocationTargetException e) {
                    e.printStackTrace();
                }
            }
        }
```

The `main()` method starts the creation of the `SwingCaBIO` in the *event-dispatching thread.* This avoids having paint methods that freeze the application i.e., the application does not respond to any mouse clicks or keyboard interactions or the application is just gray with no components created in the main frame.

```
public static void main(String[] args) {
  SwingUtilities.invokeLater(new Runnable() {
    public void run() {
      final SwingCaBIO swingCaBIO = new SwingCaBIO();
      swingCaBIO.init();
    }
  });
}
```

The complete code for the `SwingCaBIO` class is provided in **Listing 6.1.**

Listing 6.1. Class SwingCaBIO

```
package org.jfb.jcabio;

import gov.nih.nci.cabio.domain.Agent;
import gov.nih.nci.cabio.domain.Gene;
```

```
import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.lang.reflect.InvocationTargetException;
import java.util.Observable;
import java.util.Observer;

public class SwingCaBIO extends JFrame implements
DocumentListener {
    private static final int SW_WIDTH = 700;
    private static final int SW_HEIGHT = 600;
    private static final Dimension DIMENSION = new
Dimension(SW_WIDTH, SW_HEIGHT);
    private static final Dimension DIM_FIELD = new
Dimension(85, 18);
    private final static String CABIO_HTTP_SERVER_URL =

     "http://cabio.nci.nih.gov/cacore30/server/HTTPServer";
    private final static ApplicationService APP_SERVICE =
     ApplicationService.getRemoteInstance(CABIO_HTTP_SERVER
_URL);
    private static final CaBIOSearchEngine AGENT_SEARCH =
new CaBIOSearchEngine(APP_SERVICE);
    private static final CaBIOReportEngine REPORT_ENGINE =
new CaBIOReportEngine(APP_SERVICE);
    private static final String STATUS_READY = "Ready!";
    private static final StatusObject
STATUS_REPORT_GENERATED = new StatusObject("Report
generated!", 10);

    private JTextArea jTextArea;
    private JTextField gene;
    private JButton runFullGeneReport;
    private JButton runTargetAgentReport;
    private JButton runGenePathwayReport;
    private JButton clear;

    private JLabel statusBar;
    private JTextField result;
    private JTextField agent;

    public SwingCaBIO() throws HeadlessException {
        super();
        AGENT_SEARCH.addObserver(observer);
        REPORT_ENGINE.addObserver(observer);
    }

    private void init() {
        setTitle("SwingCaBIO");
```

```
            final Container contentPane = getContentPane();
            contentPane.setLayout(new BorderLayout());
            JPanel formPanel = createForm();
            JPanel reportPanel = createReportPane();
            statusBar = new JLabel(STATUS_READY);

statusBar.setBorder(BorderFactory.createEmptyBorder(5, 5, 5,
5));
            contentPane.add(formPanel, BorderLayout.NORTH);
            contentPane.add(reportPanel, BorderLayout.CENTER);
            contentPane.add(statusBar, BorderLayout.SOUTH);
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            pack();
            setSize(DIMENSION);
            final Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
            setLocation(new Point((screenSize.width - SW_WIDTH)
/ 2, (screenSize.height - SW_HEIGHT) / 2));
            show();
        }

        private JPanel createReportPane() {
            jTextArea = new JTextArea();
            jTextArea.setText("No Report.");
            jTextArea.setEditable(false);
            jTextArea.setWrapStyleWord(true);
            jTextArea.setLineWrap(true);
            final Font sf = jTextArea.getFont();
            Font f = new Font("Monospaced", sf.getStyle(),
sf.getSize());
            jTextArea.setFont(f);
            jTextArea.getDocument().addDocumentListener(new
DocumentListener() {
                public void insertUpdate(DocumentEvent event) {

clear.setEnabled(jTextArea.getText().trim().length() != 0);
                }

                public void removeUpdate(DocumentEvent event) {

clear.setEnabled(jTextArea.getText().trim().length() != 0);
                }

                public void changedUpdate(DocumentEvent event)
{

clear.setEnabled(jTextArea.getText().trim().length() != 0);
                }
            });
            JPanel jPanel = new JPanel();
            jPanel.setLayout(new BorderLayout());
            final JScrollPane jScrollPane = new
JScrollPane(jTextArea);
```

```
            jPanel.add(jScrollPane, BorderLayout.CENTER);
            jPanel.setBorder(BorderFactory.createEmptyBorder(0,
5, 5, 5));
            return jPanel;
        }

        private JPanel createForm() {
            final JPanel genePanel = new JPanel();
            genePanel.setLayout(new        BoxLayout(genePanel,
BoxLayout.LINE_AXIS));
            gene = new JTextField(10);
            this.gene.setMaximumSize(DIM_FIELD);
            gene.getDocument().addDocumentListener(this);
            final JLabel geneLabel = new JLabel("Gene");
            geneLabel.setPreferredSize(DIM_FIELD);
            genePanel.add(Box.createRigidArea(new  Dimension(5,
0)));
            genePanel.add(geneLabel);
            genePanel.add(Box.createRigidArea(new  Dimension(5,
0)));
            genePanel.add(gene);
            genePanel.add(Box.createHorizontalGlue());
            final JPanel agentPanel = new JPanel();
            agentPanel.setLayout(new      BoxLayout(agentPanel,
BoxLayout.LINE_AXIS));
            agent = new JTextField(10);
            agent.setMaximumSize(DIM_FIELD);
            agent.getDocument().addDocumentListener(this);
            final JLabel agentLabel = new JLabel("Drug Agent");
            agentLabel.setPreferredSize(DIM_FIELD);
            agentPanel.add(Box.createRigidArea(new Dimension(5,
0)));
            agentPanel.add(agentLabel);
            agentPanel.add(Box.createRigidArea(new Dimension(5,
0)));
            agentPanel.add(this.agent);
            agentPanel.add(Box.createHorizontalGlue());

            JPanel jPanel = new JPanel();
            jPanel.setLayout(new              BoxLayout(jPanel,
BoxLayout.PAGE_AXIS));
            jPanel.add(Box.createRigidArea(new     Dimension(0,
5)));
            jPanel.add(genePanel);
            jPanel.add(Box.createRigidArea(new     Dimension(0,
5)));
            jPanel.add(agentPanel);
            jPanel.add(Box.createRigidArea(new     Dimension(0,
5)));

            final JPanel resultPanel = new JPanel();
            resultPanel.setLayout(new    BoxLayout(resultPanel,
BoxLayout.LINE_AXIS));
```

```
            result = new JTextField("10", 10);
            result.setMaximumSize(DIM_FIELD);
            final JLabel resultLabel = new JLabel("Number of
results:");
            resultPanel.add(Box.createRigidArea(new
Dimension(5, 0)));
            resultPanel.add(resultLabel);
            resultPanel.add(Box.createRigidArea(new
Dimension(5, 0)));
            resultPanel.add(result);

            JPanel jPanelResult = new JPanel();
            jPanelResult.setLayout(new  BoxLayout(jPanelResult,
BoxLayout.LINE_AXIS));
            jPanelResult.add(jPanel);
            resultPanel.add(Box.createRigidArea(new
Dimension(20, 0)));
            jPanelResult.add(resultPanel);
            resultPanel.add(Box.createHorizontalGlue());


            runFullGeneReport   =   new   JButton("Run   a   Gene
Search");
            runFullGeneReport.setToolTipText("Please provide a
gene to search for.");
            runFullGeneReport.setEnabled(false);
            runFullGeneReport.addActionListener(new
ActionListener() {
                public void actionPerformed(ActionEvent event)
{
                    final StringBuffer sb = new StringBuffer();
                    Runnable runnable = new Runnable() {
                        public void run() {
                            final    String    genePattern    =
gene.getText();
                            showReport(new
StringBuffer("Searching   with   gene   '"   +   genePattern   +
"'..."));
                            try {
                                final     Gene[]     genes     =
AGENT_SEARCH.searchGenesWithGenePattern(genePattern);
                                final       int       len       =
getNumberOfObjectsForResult(genes.length);
                                sb.append("Search  Results  for
'" + genePattern + "' (" + len + " gene(s) found):\n\n");
                                for (int i = 0; i < len; i++) {
                                    Gene gene = genes[i];

REPORT_ENGINE.printFullGeneReport(gene, sb, i + 1);
                                    showReport(new
StringBuffer(jTextArea.getText()).append("\n")
                                            .append("Generated
report for gene '" + gene.getFullName() + "'"));
```

```
                                    if (i + 1 < len)
                                        sb.append("\n\n");
                                }

updateStatus(STATUS_REPORT_GENERATED);
                            } catch (SearchException se) {
                                errorDump(sb, se);
                            }
                            showReport(sb);
                        }
                    };
                    new Thread(runnable).start();
                }
            });
            runGenePathwayReport    =    new    JButton("Run    a
Gene/Pathway Search");
            runGenePathwayReport.setToolTipText("Please provide
a Gene to search for.");
            runGenePathwayReport.setEnabled(false);
            runGenePathwayReport.addActionListener(new
ActionListener() {
                public void actionPerformed(ActionEvent event)
{
                    final StringBuffer sb = new StringBuffer();
                    Runnable runnable = new Runnable() {
                        public void run() {
                            final    String    genePattern    =
gene.getText();
                            showReport(new
StringBuffer("Searching   with   gene   '"   +   genePattern   +
"'..."));
                            try {
                                final    Gene[]    genes    =
AGENT_SEARCH.searchGenesWithGenePattern(genePattern);
                                int             len             =
getNumberOfObjectsForResult(genes.length);
                                sb.append("Search  Results  for
'" + genePattern + "' (" + len + " gene(s) found):\n\n");
                                Gene gene;
                                for (int i = 0; i < len; i++) {
                                    gene = genes[i];

REPORT_ENGINE.printGenePathwayReport(gene, sb, i + 1);
                                    showReport(new
StringBuffer(jTextArea.getText()).append("\n")
                                            .append("Generated
report for gene '" + gene.getFullName() + "'"));
                                    if (i + 1 < len)
                                        sb.append("\n\n");
                                }

updateStatus(STATUS_REPORT_GENERATED);
                            } catch (SearchException se) {
```

```
                                errorDump(sb, se);
                            }
                            showReport(sb);
                        }
                    };
                    new Thread(runnable).start();
                }
            });
            runTargetAgentReport = new JButton("Run an Agent
Search");
            runTargetAgentReport.setToolTipText("Please provide
an agent to search for.");
            runTargetAgentReport.setEnabled(false);
            runTargetAgentReport.addActionListener(new
ActionListener() {
                public void actionPerformed(ActionEvent event)
{
                    final StringBuffer sb = new StringBuffer();
                    Runnable runnable = new Runnable() {
                        public void run() {
                            final    String    agentPattern    =
agent.getText();
                            showReport(new
StringBuffer("Searching with agent '" + agentPattern
                                    + "'..."));
                            try {
                                final    Agent[]    agents    =
AGENT_SEARCH.searchAgentsWithAgentPattern(agentPattern);
                                final      int      len      =
getNumberOfObjectsForResult(agents.length);
                                sb.append("Search Results for
'" + agentPattern +
                                        "'  ("  +  len  +  "
agent(s) found):\n\n");
                                Agent agent;
                                for (int i = 0; i < len; i++) {
                                    agent = agents[i];

REPORT_ENGINE.printGeneAgentCliTriReport(agent, sb, i + 1);
                                    showReport(new
StringBuffer(jTextArea.getText()).append("\n")
                                            .append("Generated
report for agent '" + agent.getName() + "'"));
                                    if (i + 1 < len)
                                        sb.append("\n\n");
                                }

updateStatus(STATUS_REPORT_GENERATED);
                            } catch (SearchException se) {
                                errorDump(sb, se);
                            }
                            showReport(sb);
                        }
```

```
                };
                new Thread(runnable).start();
            }
        });
        clear = new JButton("Clear Report");
        clear.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event)
{
                jTextArea.setText("");
            }
        });
      clear.setEnabled(false);
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new    BoxLayout(buttonPanel,
BoxLayout.LINE_AXIS));
        buttonPanel.add(runFullGeneReport);
        buttonPanel.add(Box.createRigidArea(new
Dimension(5, 0)));
        buttonPanel.add(runGenePathwayReport);
        buttonPanel.add(Box.createRigidArea(new
Dimension(5, 0)));
        buttonPanel.add(runTargetAgentReport);
        buttonPanel.add(Box.createRigidArea(new
Dimension(5, 0)));
        buttonPanel.add(clear);

        JPanel formPanel = new JPanel();
        formPanel.setLayout(new BorderLayout());
        formPanel.add(jPanelResult, BorderLayout.NORTH);
        formPanel.add(buttonPanel, BorderLayout.CENTER);
        return formPanel;
    }

    private int getNumberOfObjectsForResult(int len) {
        String numOfRes = result.getText();
        if (numOfRes != null && numOfRes.trim().length() >
0) {
            return     Math.min(Integer.parseInt(numOfRes),
len);
        }
        return len;
    }

    private void showReport(final StringBuffer sb) {
        Runnable runnable = new Runnable() {
            public void run() {
                jTextArea.setText(sb.toString());
            }
        };
        if (SwingUtilities.isEventDispatchThread())
            runnable.run();
        else
            SwingUtilities.invokeLater(runnable);
```

```
        }

      public void insertUpdate(DocumentEvent event) {
          updateButtons();
        }

        private void updateButtons() {
            boolean enabled = gene.getText().trim().length() >
0;
            runGenePathwayReport.setEnabled(enabled);
            runFullGeneReport.setEnabled(enabled);

runTargetAgentReport.setEnabled(agent.getText().trim().length
() > 0);
        }

        private void errorDump(StringBuffer sb, SearchException
e) {
            sb.delete(0, sb.length());
            sb.append("An       error       occured!\n\n"       +
e.getEmbedded().getMessage());
            updateStatus(new StatusObject("An error occured!",
5));
        }

        private Observer observer = new Observer() {
            public void update(Observable observable, Object o)
{
                updateStatus((StatusObject) o);
            }
        };

        private void updateStatusToReady() {
            updateStatus(StatusObject.STATUS_READY);
        }

        private     void     updateStatus(final     StatusObject
statusObject) {
            Runnable runnable = new Runnable() {
                public void run() {

statusBar.setText(statusObject.getStatusText());
                    if (statusObject.hasTimer()) {
                        new Thread(new Runnable() {
                            public void run() {
                                try {
                                    synchronized (this) {

this.wait(statusObject.getTimer() * 1000);
                                    }
                                    updateStatusToReady();
                                } catch (InterruptedException
e) {
```

```
                                    e.printStackTrace();
                                }
                            }
                        }).start();
                    }
                }
            };
            if (SwingUtilities.isEventDispatchThread()) {
                runnable.run();
            } else {
                try {
                    SwingUtilities.invokeAndWait(runnable);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } catch (InvocationTargetException e) {
                    e.printStackTrace();
                }
            }
        }

        public void removeUpdate(DocumentEvent event) {
            updateButtons();
        }

        public void changedUpdate(DocumentEvent event) {
        }

        public static void main(String[] args) {
            SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    final    SwingCaBIO    swingCaBIO    =    new
SwingCaBIO();
                    swingCaBIO.init();
                }
            });
        }
    }
```

Coding JcaBIO: The CaBIOReportEngine Class

In order to provide information on what the report engine is doing while it is generating the report, the CaBIOReportEngine class extends `java.util.Observable` to send notification to all *observers* about the status of the generation of the report.

```
public class CaBIOReportEngine extends Observable { }
```

The constructor for the class takes a `gov.nih.nci.system.applicationservice.ApplicationService`

object that retrieves further information during report generation as needed.

`CaBIOReportEngine` contains a number of print methods in order to generate the reports. These methods print specific information about the gene or agent (that the user supplied) into the `StringBuffer` object. The `printGene()` method takes two parameters to generate the report – the gene object and the `StringBuffer` object which will contain the information to be included in the report:

```
public void printGene(Gene gene, StringBuffer sb) { }
```

Within the `printGene()` method, we implement methods provided by the caBIO API such as `getFullName()`, `getSymbol()` and `getClusterId()` to access the relevant information about the input gene.

The `printPathways()` method takes the same two parameters to generate the pathways report:

```
public void printPathways(Gene gene, StringBuffer sb) {
}
```

Information on pathways is obtained as a collection of pathway objects using the method `search()` from the application service object `appService`. The `search()` method requires two parameters - the type of the object we want in the collection result and the gene we need the pathways for as shown below:

```
final Collection tmp =
appService.search(Pathway.class, gene);
```

Similarly, we use print methods to retrieve information on gene aliases (`printGeneAliases()`), clinical trials (`printClinicalTrials()`), Agent (`printAgent()`) etc. The complete code for `CaBIOReportEngine` is provided in **Listing 6.2**.

Listing 6.2. Class CaBIOReportEngine

```
package org.jfb.jcabio;

import gov.nih.nci.cabio.domain.Agent;
import gov.nih.nci.cabio.domain.ClinicalTrialProtocol;
import gov.nih.nci.cabio.domain.Gene;
import gov.nih.nci.cabio.domain.GeneAlias;
```

```
import gov.nih.nci.cabio.domain.GeneOntology;
import gov.nih.nci.cabio.domain.HomologousAssociation;
import gov.nih.nci.cabio.domain.Pathway;
import gov.nih.nci.cabio.domain.Protein;
import gov.nih.nci.cabio.domain.Target;

import java.text.SimpleDateFormat;
import java.util.Collection;
import java.util.Iterator;
import java.util.Observable;

public class CaBIOReportEngine extends Observable {
    private static final SimpleDateFormat DATE_FORMATTER =
new SimpleDateFormat("yyyy.MM.dd G 'at' HH:mm:ss z");
    private static final StatusObject STATUS_REPORT_DONE =
new StatusObject("Report done!");

    private ApplicationService appService;

    public CaBIOReportEngine(ApplicationService appService)
{
        this.appService = appService;
    }

    public void printGene(Gene gene, StringBuffer sb) {
        sb.append("Name: " + gene.getFullName());
        sb.append("\n-Symbol: " + gene.getSymbol());
        sb.append("\n-Unigene     Cluster     Id:     "     +
gene.getClusterId());
    }

    public void printPathways(Gene gene, StringBuffer sb) {
        notifyObservers(new    StatusObject("Printing    the
pathway report for gene '" + gene.getFullName() + "'..."));
        try {
                final        Collection        tmp        =
appService.search(Pathway.class, gene);
            final int size = tmp.size();
            if (size == 0) {
                sb.append("Gene    not    found    in    any
pathways.");
                notifyObservers(STATUS_REPORT_DONE);
                return;
            }
            sb.append(size + " pathway(s) found: \n");
            Pathway[] pathways = new Pathway[size];
            tmp.toArray(pathways);
            for (int i = 0; i < pathways.length; i++) {
                Pathway pathway = pathways[i];
                sb.append("\t-Pathway     name:      "      +
pathway.getName());
                sb.append("\n\t-Description:       "       +
pathway.getDisplayValue());
```

```
                    sb.append("\n\t-Pathway                    Map:
http://www.biocarta.com/pathfiles/"   +   pathway.getName()   +
".asp");
                    if (i + 1 < pathways.length) {
                        sb.append("\n");
                    }
                }
            } finally {
                notifyObservers(new        StatusObject("Pathway
report done for gene '" + gene.getFullName() + "'!"));
            }
        }

        public void printGeneAliases(Gene gene, StringBuffer
sb) {
            notifyObservers(new StatusObject("Printing the gene
alias report for gene '" + gene.getFullName() + "'..."));
            try {
                final          Collection          tmp          =
appService.search(GeneAlias.class, gene);
                final int size = tmp.size();
                if (size == 0) {
                    sb.append("No gene aliases found.");
                    notifyObservers(STATUS_REPORT_DONE);
                    return;
                }
                sb.append(size + " gene aliases found: ");
                GeneAlias[] geneAliases = new GeneAlias[size];
                tmp.toArray(geneAliases);
                for (int i = 0; i < geneAliases.length; i++) {
                    GeneAlias geneAlias = geneAliases[i];
                    sb.append(geneAlias.getName());
                    if (i + 1 < geneAliases.length) {
                        sb.append(", ");
                    }
                }
            } finally {
                notifyObservers(new   StatusObject("Gene   alias
report done for gene '" + gene.getFullName() + "'!"));
            }
        }

        private void printAgent(Agent agt, StringBuffer sb) {
            sb.append("Drug Agent Name: " + agt.getName());
            final String source = agt.getSource();
            sb.append("\n-Agent Source: " + (source != null ?
source : "Unknown"));
        }

        public void printGenes(Target target, StringBuffer sb)
{
            notifyObservers(new StatusObject("Printing the gene
report for target '" + target.getName() + "'..."));
```

```
        try {
            final Collection tmp =
appService.search(Gene.class, target);
            final int size = tmp.size();
            if (size == 0) {
                sb.append("No genes found.");
                notifyObservers(STATUS_REPORT_DONE);
                return;
            }
            sb.append(size + " gene(s) found: ");

            Gene[] genes = new Gene[size];
            tmp.toArray(genes);
            for (int i = 0; i < genes.length; i++) {
                Gene agt = genes[i];
                printGene(agt, sb);
                if (i + 1 < genes.length) {
                    sb.append("\n");
                }
            }
        } finally {
            notifyObservers(new StatusObject("Gene report
done for gene '" + target.getName() + "'!"));
        }
    }

    public void printClinicalTrials(Agent agt, StringBuffer
sb) {
        notifyObservers(new StatusObject("Printing the
clinical trial report for agent '" + agt.getName() +
"'..."));
        try {
            final Collection tmp =
appService.search(ClinicalTrialProtocol.class, agt);
            final int size = tmp.size();
            if (size == 0) {
                sb.append("No clinical trials found for
agent.");
                notifyObservers(STATUS_REPORT_DONE);
                return;
            }
            sb.append(size + " clinical trial(s) found: ");
            ClinicalTrialProtocol[] clinicalTrials = new
ClinicalTrialProtocol[size];
            tmp.toArray(clinicalTrials);
            for (int i = 0; i < clinicalTrials.length; i++)
{
                ClinicalTrialProtocol clinicalTrial =
clinicalTrials[i];
                sb.append("\n\nTitle: " +
clinicalTrial.getTitle());
                sb.append("\n-Status: " +
clinicalTrial.getCurrentStatus());
```

```
                    sb.append("\n-Date:                "            +
DATE_FORMATTER.format(clinicalTrial.getCurrentStatusDate()));
                    sb.append("\n-Lead Organization Name: " +
clinicalTrial.getLeadOrganizationName());
                    sb.append("\n-Phase:               "            +
clinicalTrial.getPhase());
                    sb.append("\n-Participation   Type:    "    +
clinicalTrial.getParticipationType());
                    if (i + 1 < clinicalTrials.length) {
                        sb.append("\n");
                    }
                }
            } finally {
                notifyObservers(new      StatusObject("Clinical
trial report done for agent '" + agt.getName() + "'!"));
            }
        }

        public void printGeneOntology(Gene gene, StringBuffer
sb) {
            notifyObservers(new StatusObject("Printing the gene
ontology report for gene '" + gene.getFullName() + "'..."));
            try {
                final          Collection          tmp          =
appService.search(GeneOntology.class, gene);
                final int size = tmp.size();
                if (size == 0) {
                    sb.append("No associated GO terms found.");
                    notifyObservers(STATUS_REPORT_DONE);
                    return;
                }
                sb.append(size + " GO Term(s) found: ");
                GeneOntology[]     geneOntologies     =     new
GeneOntology[size];
                tmp.toArray(geneOntologies);
                for (int i = 0; i < geneOntologies.length; i++)
{
                    GeneOntology          geneOntology          =
geneOntologies[i];
                    sb.append(geneOntology.getName());
                    if (i + 1 < geneOntologies.length) {
                        sb.append(", ");
                    }
                }
            } finally {
                notifyObservers(new StatusObject("Gene ontology
report done for gene '" + gene.getFullName() + "'!"));
            }
        }

        public void printProteins(Gene gene, StringBuffer sb) {
            notifyObservers(new    StatusObject("Printing    the
protein report for gene '" + gene.getFullName() + "'..."));
```

```
        try {
            final Collection tmp =
appService.search(Protein.class, gene);
            final int size = tmp.size();
            if (size == 0) {
                sb.append("No proteins found for " +
gene.getFullName() + ".");
                notifyObservers(STATUS_REPORT_DONE);
                return;
            }
            sb.append("Protein name: ");
            for (Iterator iterator = tmp.iterator();
iterator.hasNext();) {
                Protein protein = (Protein)
iterator.next();
                sb.append(protein.getName());
                if (iterator.hasNext()) {
                    sb.append(", ");
                }
            }
        } finally {
            notifyObservers(new StatusObject("Protein
report done for gene '" + gene.getFullName() + "'!"));
        }
    }

    public void printGenes(Agent agent, StringBuffer sb) {
        notifyObservers(new StatusObject("Printing the gene
report for agent '" + agent.getName() + "'..."));
        try {
            final Collection tmp =
appService.search(Target.class, agent);
            final int size = tmp.size();
            if (size == 0) {
                sb.append("No targets found for " +
agent.getName() + ".");
                notifyObservers(STATUS_REPORT_DONE);
                return;
            }
            sb.append(size + " targets found: ");
            Target[] targets = new Target[size];
            tmp.toArray(targets);
            for (int i = 0; i < targets.length; i++) {
                Target target = targets[i];
                printGenes(target, sb);
                if (i + 1 < targets.length) {
                    sb.append("\n");
                }
            }
        } finally {
            notifyObservers(new StatusObject("Gene report
done for agent '" + agent.getName() + "'!"));
        }
```

```
    }

    public void notifyObservers(Object o) {
        setChanged();
        super.notifyObservers(o);
    }

    public void printFullGeneReport(Gene geneFound, final
StringBuffer sb, int paragraphNumb) {
        sb.append(paragraphNumb + ". ");
        printGene(geneFound, sb);
        sb.append("\n-");
        printGeneOntology(geneFound, sb);
        sb.append("\n-");
        printProteins(geneFound, sb);
        sb.append("\n-");
        printGeneAliases(geneFound, sb);
    }

    public void printGenePathwayReport(Gene geneFound,
final StringBuffer sb, int paragraphNumb) {
        sb.append(paragraphNumb + ". ");
        printGene(geneFound, sb);
        sb.append("\n-");
        printPathways(geneFound, sb);
    }

    public void printGeneAgentCliTriReport(Agent agent,
final StringBuffer sb, int paragraphNumb) {
        sb.append(paragraphNumb + ". ");
        printGenes(agent, sb);
        sb.append("\n-");
        printAgent(agent, sb);
        sb.append("\n-");
        printClinicalTrials(agent, sb);
    }
}
```

Coding JcaBIO: The CaBIOSearchEngine Class

`CaBIOSearchEngine` extends *Observable* to notify *observers* about what the search engine is doing so we can keep the users of the application informed about the status of the search. As described earlier, we provide two search capabilities in the `SwingCaBIO` application: one to create a Gene report and one to create an Agent report. We will call the Gene search method `searchGenesWithGenePattern()` and the agent search method `searchAgentsWithAgentPattern()` respectively.

The constructor for the class takes a `gov.nih.nci.system.applicationservice.ApplicationService` object that helps to run the initial search. Based on the caCORE API, we create an object called `GeneCriteria` and set the gene name with the pattern we're looking for. We run the `ApplicationService` and supply it with the object we need to retrieve. We then collect all the genes that match the input criteria and return the result in an array of `Gene` objects.

As with the `searchGenesWithGenePattern()` method, `searchAgentsWithAgentPattern()` returns an array of `Agent` objects found. The complete code for the `CaBIOSearchEngine class` is provided in **Listing 6.3**.

Listing 6.3. Class CaBIOSearchEngine

```
  package org.jfb.jcabio;

  import gov.nih.nci.cabio.domain.Agent;
  import gov.nih.nci.cabio.domain.Gene;
  import gov.nih.nci.cabio.domain.GeneAlias;
  import gov.nih.nci.cabio.domain.impl.AgentImpl;
  import gov.nih.nci.cabio.domain.impl.GeneAliasImpl;
  import
gov.nih.nci.system.applicationservice.ApplicationService;

  import java.util.ArrayList;
  import java.util.Collection;
  import java.util.List;
  import java.util.Observable;
  public class CaBIOSearchEngine extends Observable {

      private static final StatusObject STATUS_SEARCH_DONE =
new StatusObject("Search done!");

      private ApplicationService appService;

      public CaBIOSearchEngine(ApplicationService appService)
{
          this.appService = appService;
      }

      public      Gene[]      searchGenesWithGenePattern(String
geneNamePattern) throws SearchException {
          try {
              notifyObservers(new       StatusObject("Starting
search with gene pattern '" + geneNamePattern + "'..."));
       List resultList = appService.search(Gene.class, gene);
              Gene[] genes = new Gene [resultList.size()];
              resultList.toArray(genes);
```

```
                notifyObservers(STATUS_SEARCH_DONE);
                return genes;
            } catch (Throwable e) {
                notifyObservers(new    StatusObject("An    error
occured while searching for genes using gene pattern '"
                        + geneNamePattern + "'!", 10));
                throw new SearchException("", e);
            }
        }

        public    Agent[]    searchAgentsWithAgentPattern(String
agentPattern)
                throws SearchException {
            try {
                notifyObservers(new      StatusObject("Starting
search with gene pattern '" + agentPattern + "'..."));
         Agent agentCriteria = new AgentImpl();
         agentCriteria.setName(agentPattern);
         List    resultList    =    appService.search(Agent.class,
agentCriteria);
              Agent[] agents = new Agent[resultList.size()];
               resultList.toArray(agents);
               notifyObservers(STATUS_SEARCH_DONE);
               return agents;
            } catch (Throwable e) {
               notifyObservers(new    StatusObject("An    error
occured while searching for agents with gene pattern '"
                        + agentPattern + "'..."));
               throw new SearchException("", e);
            }
        }

        public void notifyObservers(Object o) {
            setChanged();
            super.notifyObservers(o);
        }
    }
```

`SearchException` and `StatusObject` respectively provide mechanisms to handle errors that occur during the search process and provide the user with messages on the status of the search. The code for these two classes is provided in **Listing 6.4** and **Listing 6.5** below.

Listing 6.4. Class SearchException

```
package org.jfb.jcabio;
public class SearchException extends Exception {
    private Throwable embedded;

    public SearchException(String s, Throwable throwable) {
```

```
        super(s, throwable);
        this.embedded = throwable;
    }

    public Throwable getEmbedded() {
        return embedded;
    }
}
```

Listing 6.5. Class StatusObject

```
package org.jfb.jcabio;

public class StatusObject {
    private static final int NO_TIMER = 0;
    private static final String STATUS_TEXT = "Ready!";

    public static StatusObject STATUS_READY = new
StatusObject(STATUS_TEXT, NO_TIMER);

    private String statusText;
    private int timer;

    public StatusObject(String statusText, int timer) {
        this.statusText = statusText;
        this.timer = timer;
    }

    public StatusObject(String statusText) {
        this.statusText = statusText;
        timer=NO_TIMER;
    }

    public String getStatusText() {
        return statusText;
    }

    public int getTimer() {
        return timer;
    }

    public boolean hasTimer() {
        return timer != NO_TIMER;
    }
}
```

Running the JcaBIO Application

As described in **Table 6.4**, among the caBIO domain objects, the gene object serves as central hub of the basic research objects and provides access to object such as organs, diseases, chromosomes, pathways, sequence data, etc. To begin with, therefore, we will create a Gene report using the `JcaBIO` application. **Fig. 6.4** shows the results of a gene report conducted to search for genes named "*erb*". Note that *wild-cards* (*) can be used for retrieving information on genes. In this case, for example, we have performed a search with *erb** which as the report indicates has identified genes called "*v-erb-b2 erythroblastic leukemia viral oncogene homolog 3 (avian)*", with the approved Human Gene Nomenclature Committe (HGNC) gene symbol *ERBB3* and "*v-erb-b2 erythroblastic leukemia viral oncogene homolog 2, neuro/glioblastoma derived oncogene homolog (avian)*" with the approved HGNC gene symbol *ERBB2*, both of which are members of a family of *growth factor receptor* genes called *epidermal growth factor receptors* (EGFR).

Fig. 6.5 displays the results of a pathway search for the keyword *erb**. The search identifies three genes that match the input keyword *erb**: *ERBB2*, *ERBB3* and *ERBB4*, the corresponding pathways the three genes are involved in and a link to the graphical representation of the pathways on the BioCarta website for each (as shown in **Fig. 6.6** for *ERBB2*).

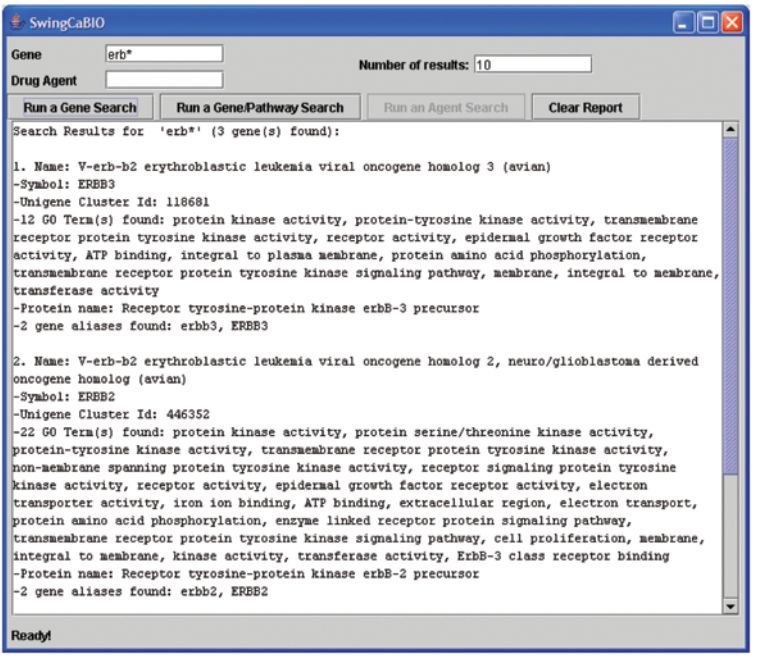

Fig. 6.4. Gene report for erb*

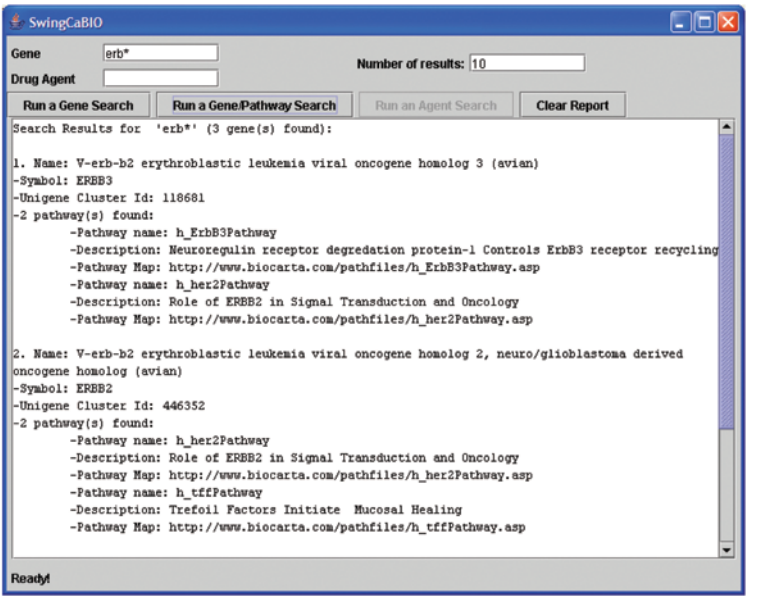

Fig. 6.5. Pathway report for erb*

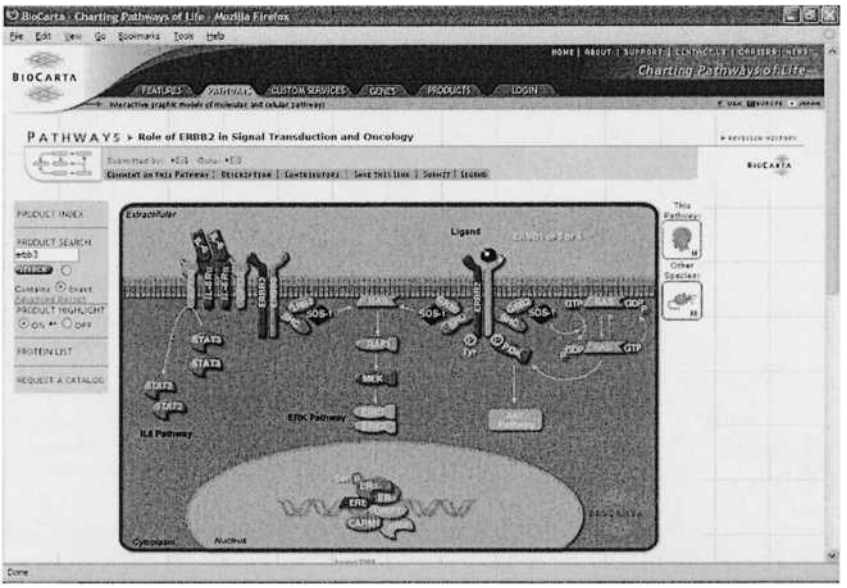

Fig. 6.6. Graphical representation of an *ERBB2* pathway in BioCarta

Next we will perform a therapeutic agent search for a well-known anti-cancer agent called *Taxol*. **Fig. 6.7** displays the results of a wild-card search performed with the term *TAX**. As expected, the search resulted in reports on Taxol, a compound present in the bark of the Pacific yew tree (*Taxus brevifolia*), which was later found to possess anti-cancer properties and approved for the treatment of ovarian, breast and non-small cell lung cancer. The report also presents detailed information on the many clinical trials that are being conducted using Taxol providing such details as the name of the study, its status, the organization conducting the study, Phase of the clinical trials and so on.

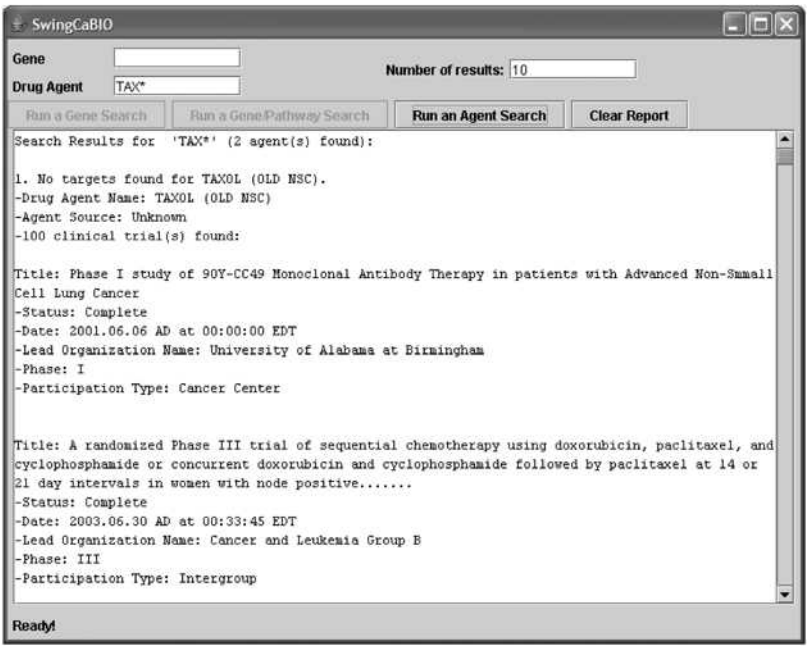

Fig. 6.7. Therapeutic agent report for Taxol

Summary

The NCI caBIG™ initiative is ushering a new era in cancer research by providing scientists with standardized tools to access and share information with one another overcoming cultural, geographical and technological barriers in ways not conceivable just a few years earlier.

In this chapter, we learnt about the rationale behind the creation of caBIG™ and the technologies that are being created or developed under the initiative to enhance the pace of cancer research. We created a very basic application to demonstrate a few of the many ways in which NCI's caCORE and caBIO domain objects can be used to retrieve information on biomedical objects in a way that bridges basic and clinical research. Needless to say, caCORE offers many more capabilities than what we have attempted to demonstrate and we encourage readers to take these small examples as a springboard to gain a better understanding of the power of the technology and build more complex queries as dictated by their individual research needs.

The power of the caBIG™ concept is uniting cancer researchers across the world. A similar initiative was launched by the UK National Health Service (NHS) for the development of cancer research informatics in that country through a strategic partnership with the NCICB on the caBIG™ effort. Both the initiatives will work together to build a truly global infrastructure for cancer research. These are indeed very exciting times for biomedical and clinical research and it is hoped that the joint efforts of people across the world will eventually lead to the demise of the scourge that we are battling.

As a living testimony of the work being done in this area, the NCI was recently awarded the 2006 Computerworld Honors 21st Century Achievement Award for Science for their accomplishment under caBIG™ Program. The Computerworld Honors Program was established to honor people or institutions who apply Information Technology for the benefit of society. Further information on the award is available at http://www.cwhonors.org/archives/2006/index.htm and https://cabig.nci.nih.gov/News_Folder/NCI_award.

Questions and Exercises

1. The NCICB has launched the Open Development Initiative (ODI, http://ncicb.nci.nih.gov/NCICB/infrastructure/open_dev_initiative) as an opportunity for biomedical researchers and bioinformaticians to contribute to on-going development efforts in the cancer domain. Explore the caBIO, caCORE and other ODI's of interest to you and think of ways you can participate in this effort.

2. The observation that, "Gene and/or protein X is significantly overexpressed in a specific cell population, tissue and/or in a laboratory model of disease Y" is that fundamental first indication of evidence that feeds hypothesis driven research into the biology and treatment of disease.

    a. What caBIO objects would you need to establish a causative link between biomolecules expressed in specific tissues (for example, cerebral cortex) and disease (for example, Alzheimer's disease)?

b. How would you extend the query to identify pathways that the biomolecules participate in and discover known chemical agents that selectively inhibit or modify events along the pathways?
c. Which caCORE data stores would you mine for such information?
d. Given that the ultimate aim of caBIG™ is to make biomedical and clinical data accessible via the grid, how would you design an application to take the information obtained above to locate appropriate tissue samples, patient cohorts and on-going clinical trials for further analysis and validation studies? What technical and non-technical issues would you need to address to build such an application?
e. Create an application expanding available caBIG™ technologies and data stores that will allow users to run such queries.

Additional Resources

Select NIH/NCI resources

- caBIO - http://ncicb.nci.nih.gov/core/caBIO
- caCORE - http://ncicb.nci.nih.gov/NCICB/infrastructure
- CaDSR - http://ncicb.nci.nih.gov/core/caDSR
- CaMOD - http://cancermodels.nci.nih.gov
- CMAP - http://cmap.nci.nih.gov
- CTEP - http://ctep.cancer.gov/
- CGAP - http://cgap.nci.nih.gov/
- CGAP GAI - http://gai.nci.nih.gov/
- EVS - http://ncicb.nci.nih.gov/core/EVS
- GEDP - http://gedp.nci.nih.gov

- MMHCC - http://mouse.ncifcrf.gov/
- NCI metathesaurus - http://ncimeta.nci.nih.gov/
- NCI thesaurus - http://nciterms.nci.nih.gov/NCIBrowser/Dictionary.do
- UniSTS - http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=unists

Other biomedical repositories and resources

- BioCarta pathways - http://www.biocarta.com/
- Gene Ontology Project - http://www.geneontology.org/
- IMAGE Consortium - http://image.llnl.gov/

Standards and protocols

- ISO/IEC - http://www.standardsinfo.net/isoiec/index.html
- ISO/IEC 11179 standard - http://metadata-standards.org/11179/
- SOAP - http://www.w3.org/TR/soap/

ETL tools

- Kettle - http://www.kettle.be/
- Octopus - http://www.enhydra.org/tech/octopus/index.html

Selected Reading

The caCORE Software Development Kit: streamlining construction of interoperable biomedical information services. Phillips J, Chilukuri R, Fragoso G, Warzel D, Covitz PA. BMC Med Inform Decis Mak. 2006 Jan 6;6:2.

Covitz PA, Hartel F, Schaefer C, De Coronado S, Fragoso G, Sahni H, Gustafson S, Buetow KH. caCORE: a common infrastructure for cancer informatics. Bioinformatics. 2003;19:2404–2412.

Database resources of the National Center for Biotechnology Information. Wheeler DL, Barrett T, Benson DA, Bryant SH, Canese K, Chetvernin V, Church DM, DiCuccio M, Edgar R, Federhen S, Geer LY, Helmberg W, Kapustin Y, Kenton DL, Khovayko O, Lipman DJ, Madden TL, Maglott DR, Ostell J, Pruitt KD, Schuler GD, Schriml LM, Sequeira E, Sherry ST, Sirotkin K, Souvorov A, Starchenko G, Suzek TO, Tatusov R, Tatusova TA, Wagner L, Yaschenko E. Nucleic Acids Res. 2006 Jan 1;34(Database issue):D173-80.