

# **Codename OCT: The Organic Chemistry Tutorial**

**Jeff Hauck**  
**Division of Science and Mathematics**  
**University of Minnesota, Morris**  
**hauckjj@mrs.umn.edu**

**Scott Lewandowski**  
**Division of Science and Mathematics**  
**University of Minnesota, Morris**  
**swl@mrs.umn.edu**

## **Abstract:**

The long-term goal of this project is the development of an interactive, computer-based tutorial for beginning organic chemistry students to help develop their problem solving skills in the area of retrosynthetic analysis. Retrosynthetic analysis refers to breaking target molecules into sensible pieces so that their synthesis can be planned. Since no such application is known to exist, this project fills an unmet need for organic chemistry tutorials.

The main goal of our current work is to design and build an extensible, and web deployable application that requires no installation for clients. To achieve the latter goal, the application uses a four-tier architecture consisting of a MySQL database, Enterprise JavaBeans (EJB's), JavaServer Pages (JSP pages), and the client.

Our design thus allows anyone with Internet access and a user account to practice retrosynthetic analysis problems. Faculty can monitor students' progress from a web browser, and thereby adjust their teaching accordingly.

## Project History

### Origins

The Organic Chemistry Tutorial (OCT) project was originally conceived of in 1998. It grew out of a faculty enrichment mentorship pairing between Scott Lewandowski from Computer Science and Nancy Carpenter from Chemistry. The mentorship pairing focused on issues related to developing and sustaining a successful research program involving undergraduates. The OCT project was envisioned as an ongoing collaborative and interdisciplinary research effort. The long-term goal of the project is the design and development of an interactive, computer-based tutorial for beginning organic chemistry students to help develop their problem solving skills, particularly in the area of retrosynthetic analysis.

### About Retrosynthetic Analysis

Retrosynthetic analysis is the process of breaking down complex target molecules into sensible pieces so that their synthesis can be planned. In other words, it's a kind of molecular reverse engineering performed to determine what starting molecules (or precursors) and reagents could be used to create (or synthesize) the desired target molecule (see Figure 1).



**Figure 1:** simplified example of a retrosynthetic reaction

The key to retrosynthetic analysis is to work the problem backwards. Unfortunately, this seems to be a skill students have difficulty mastering, and while there are software products available to assist professional chemists in performing retrosynthetic analyses, there appear to be no tutorials designed to teach organic chemistry students the basic concepts behind this process.

### Development Efforts

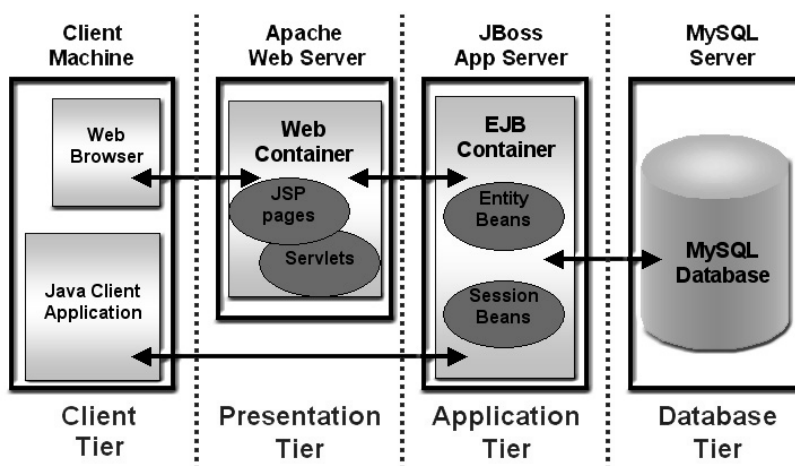
Development work began in the fall of 1999. Matt Hardy, a Computer Science major who was also pursuing a minor in Chemistry received funding through the Morris Academic Partnership (MAP) program to work on the design and development of a Java-based prototype tutorial system during the 1999-2000 academic year. The resulting system served as a proof of concept and also a framework upon which further development efforts could build.

Matt expanded on the work of his MAP project during the 2000-2001 academic year. These efforts, funded by the Undergraduate Research Opportunities Program (UROP) of the University of Minnesota, focused on the development of an intelligent tutoring system capable of adapting to different students' educational requirements. While this work did not result in a working system, it did lay the groundwork for much of our current approach. Matt made significant contributions with respect to back-end system design (including the use of a database for storing problem information) and also to the design of the user interface.

Jeff Hauck joined the project in the fall of 2001. Since that time we have completely redesigned the tutorial. Drawing on our previous experiences (both with the prototype tutorial and also from Jeff's summer internship) and taking advantage of recent technological developments such as Enterprise JavaBeans (EJB's) we moved to what we refer to as a *zero footprint* design. *Zero footprint* meaning that a user of the system need not install any software (beyond a web browser) on their computer in order to use our tutorial. The remainder of this paper describes in further detail our *zero footprint* design and the technologies that make it possible.

## System Overview

The system is comprised of four tiers, the database tier, the application tier, the presentation tier, and the client tier (see Figure 2). The database tier consists of a MySQL database server. This server is used to store information about users, reactions, and molecules. This tier is queried for information by the application running on the application tier and then the information is presented to the user via the presentation tier.



**Figure 2:** description of the four-tier architecture of the system

A Java 2 Enterprise Edition (J2EE) server is designed to provide a container and a container management system for applications that use the Java 2 Enterprise Edition API's. Our design makes use of these API's with implementations of Enterprise JavaBeans (EJB's or beans) and JavaServer Pages (JSP pages). In our case, two different

J2EE servers define the presentation and application tiers: JBoss, the application server, provides an EJB container, and Apache, the web server, provides a web container for JSP pages.

JBoss is a free, open source application server that provides a J2EE container, or Java runtime environment, for the Enterprise JavaBeans. The runtime environment consists of a Java Virtual Machine (JVM), the J2EE classes, and supporting files. The JBoss runtime environment is responsible for creating bean instances ahead of time and placing them in a bean object pool for the client to use[3]. By using an object pool, the container does not continue to create new instances of objects and then garbage collect them later[1]. JBoss handles the management and security of transactions occurring in the system. These transactions include remote method invocations (Java RMI) from the presentation tier, and database tier queries using Java Database Connectivity (JDBC). The packaged beans are placed or deployed in the container, where JBoss will unpack them, and run them in the JBoss runtime environment.

Apache is a free, open source web server that provides a J2EE container to manage servlets and JSP pages. The web container also provides transaction management. For this project, we are running an installation of the Apache web server included with JBoss. The JSP pages deployed in the presentation tier are responsible for the display of static and dynamic web page content to the user in the form of problems.

The final tier consists of a client machine anywhere in the world with an Internet or Intranet connection and a web browser. As Figure 2 illustrates, a java application could also remotely invoke bean methods directly from the client machine. The remote invocations would be identical to those in contained in the JSP pages in the presentation tier. However, this approach defeats the goal of having a *zero footprint* system by requiring installation to each remote client machine.

The statistics that are collected when organic chemistry students submit solutions would be skewed if non-students were allowed to use the system. An example of such a statistic would be the number of times a problem was answered incorrectly, or the average skill level of the class. User names and passwords ensure the integrity of the data stored in the database tier.

All database transactions will take place using a single MySQL account only intended for the OCT tutorial, so faculty will not need individual accounts on the database server. Instead, special privileges will be assigned to organic chemistry faculty. These privileges will include the ability to access information such as student progress, or the ability to add and remove molecules or reactions from the system. If a user does not have the right privileges, the link to the JSP page containing the administrative tools will not be displayed.

The system, as designed, can be scaled in a number of ways to accommodate more users. In this particular case, we have MySQL, JBoss, and Apache running on the same machine. Each server could be installed on different physical machines to compensate

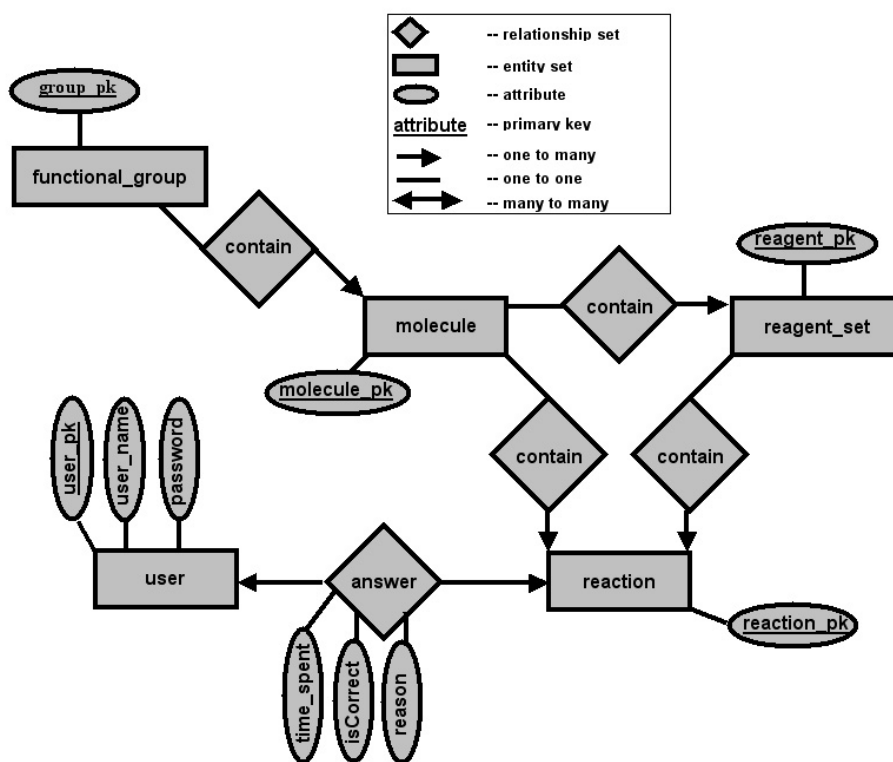
for speed degradation caused by too many users. Expensive remote method invocations between the web server and the application server are used regardless of the number of physical machines because they each use a different JVM. The efficiency of the JDBC connections between the application server and the database will also remain about the same. The additional cost of making transactions from a larger distance will depend upon the networking bandwidth between the separate machines. The organic chemistry tutorial system could also be "mirrored" or installed on a second set of machines, in which case students could be routed to the alternate system to balance the load.

## MySQL

MySQL is an open source relational database server. The MySQL database used in our tutorial contains six tables or relations represented in the schema and Entity-Relationship (E-R) diagram shown in Figure 3.

```

user ( user_pk, user_name, passwd, privileges, skill_level );
answer ( user_pk, reaction_pk, time_spent, isCorrect, reason );
molecule ( molecule_pk, iupac_name, common_name, formula, functional_groups );
reagent_set ( set_pk, reagent_1, reagent_2 );
functional_groups ( group_pk, group_name );
reaction ( reaction_pk, reactant, reagent_set, target, difficulty );
  
```



**Figure 3:** schema and Entity-Relationship (E-R) diagram for the database

The relationship sets in Figure 3 labeled `contain` do not actually exist in the database, but instead represent primary key – foreign key mappings between the different entity sets in the table. Leaving them out creates no repetition in the remaining tables and allows more efficient retrieval of information. For example, only two tables need to be joined instead of three in order to retrieve information on molecules *and* the functional groups they contain.

The `user` table contains the name and password of each user. The information provided by the user at login is checked against this table. Each user has a corresponding privilege level, which is used for the suppression of certain information for non-authorized users. Rows containing the user names of students have an integer value in the `skill_level` column. As the student correctly answers problems of increasing difficulty, their skill level increases. This column contains a null value for faculty.

The `answer` table records the interaction between the student and the tutorial. A row in this table will contain a reference to the reaction the problem was modeled after. In addition, the row stores a reference to the user, and the time spent on the problem. The `isCorrect` column is used to record whether or not the solution proposed by the student was correct, and a reason the student may have answered incorrectly is also stored. Queries to the `answer` table will be the main source of information for the intelligent tutoring agent (see Future Work).

The `reaction` table relates molecules together to form recipes for specific retrosynthetic reactions from which problems can be made. The `reactant` and `target` columns are foreign keys to rows in the `molecule` table. The `reagent_set` is a foreign key for a row in the `reagent_set` table. Each row in the `reagent_set` table contains a pair of foreign keys for rows in the `molecule` table. It must be noted that molecules can generally be synthesized in more than one way. There may be many reactions in the table with the same target molecule, so proposed solutions must be carefully checked against the entire `reaction` table.

The `molecule` table holds data for the specific molecules, such as common and IUPAC names, chemical formula, and a list of functional groups. Functional group interactions are the defining feature in retrosynthetic reactions, so information on each functional group for each molecule is entered into the database. If a molecule is removed from the database, the system will cascade through the `reaction` and `reagent_set` tables and remove any reactions and reagent sets that contain that molecule's primary key. Similarly, if a functional group is removed from the database, all of the molecules containing that functional group will be removed.

In our original database planning, we proposed the idea of storing the images of the molecules and functional groups themselves. The overhead involved with retrieving the long bit sequence representing an image from the database tier would easily match the overhead of retrieving the image as a file on the presentation tier. We consequently chose to store the image files in an ordinary directory.

## Enterprise JavaBeans

The Enterprise JavaBeans (EJB's or beans) encode the main application logic. Beans are server-side components that encapsulate the business logic of an application[2]. In this case, the business logic includes retrieving data from the database and presenting it as a `Problem` object for a student to solve. The business logic then has the responsibility of checking and storing the solution submitted by the student.

Beans generally involve three parts, the home interface, the component interface and the enterprise bean implementation. The home interface is the piece that is responsible for creating instances of the component interface for an enterprise bean[1]. The component interface of an enterprise bean defines the methods available for clients to invoke[1]. The enterprise bean contains the implementations of the methods specified in the component interface[1].

The home interface is found by doing a Java Naming and Directory Interface (JNDI) *lookup*. If the stub is successfully located in the JNDI namespace maintained by the application server, the `create()` method can then be called on the home interface in which an instance of the enterprise bean is chosen from the JBoss instance pool and the component interface stub is returned. Methods are invoked on the component interface stub which then calls the same methods implemented in the enterprise bean as shown in Figure 4.

```
// find the home interface from the JNDI Namespace
BookKeeperHome home = (BookKeeperHome) naming.lookup("BookKeeperHome");

// retrieve the component stub by calling create() on the home interface
BookKeeper keeper = home.create();

// invoke methods on the component stub
keeper.checkSolutionAndStore(userName, answered_problem);
```

**Figure 4:** example code that locates the home interface and retrieves the component interface to invoke methods on the bean

Too much logic existed to house in one enterprise bean, so we initially designed six EJB's to contain the main application logic of the system. The four entity beans, `ListGenerator`, `ProblemGenerator`, `UserManager`, and `SolutionCheck` have specific tasks to perform for each individual user, and are shown in Figure 5.

The `UserManager` bean is responsible for checking appropriate login information for security purposes. The `UserManager` is also used for storing correct and incorrect solutions submitted for problems and the time spent for each problem, in addition to maintaining the student's skill level.

The `ProblemGenerator` bean is only responsible for retrieval of problem data. This bean is not designed to hold any intelligence logic, but instead simply takes a problem number as an argument. The value corresponds to the primary key of a reaction stored in

the reaction table. The problem generator retrieves this reaction and the molecules contained in it, and returns a `Problem` object to be displayed by the JSP pages.

A `Problem` object has two states: answered and unanswered. Once a solution to a problem has been submitted by a user, the problem is marked as answered, checked by the `SolutionCheck` bean and stored by the `UserManager` bean as shown in Figure 5. The `Problem` object is an application of the *Value Object Pattern*[4]. If, for example, reactions were comprised of ten total pieces of data, the presentation tier could use ten different accessor methods to retrieve all of it. This would require ten remote method invocations, and would be costly in terms of performance. Instead, all of the necessary data can be placed in a data structure or *value object*, so the same data can be retrieved with only one remote method invocation.

The `ListGenerator` bean is responsible for generating lists of reagent sets and possible starting materials for the problem. These will be displayed for the user to pick from when solving the problem. Like the `ProblemGenerator` bean, the `ListGenerator` bean contains no intelligence logic, and simply takes parameters that determine which molecules and reagent sets are chosen and returned.

The `SolutionCheck` bean is responsible for checking the student's solution to the problem. This bean must query the database to find all possible reactions with the correct target molecule in the rare case that more than one answer is possible for the given starting materials and reagent sets. When this is completed, the appropriate response is given to the presentation tier, and the `UserManager` records the student's solution.

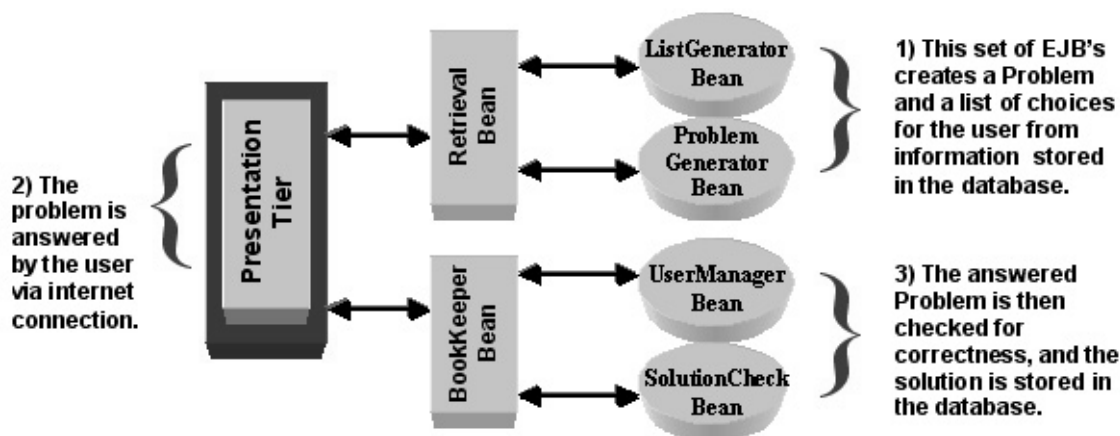


Figure 5: diagram of EJB interaction in the system

There are two session beans for the application, the `BookKeeper` and `Retrieval` beans. These are shown in Figure 5. Both of the session beans are stateless, meaning that there is not a dedicated bean instance for each user for the duration of the session. Instead the beans will be re-pooled when each transaction between the presentation and application tiers is complete.



The *Core J2EE Patterns* book warns that exposing the entity bean to clients from a different tier “results in network overhead and performance degradation”[4]. A proposed refactoring for this is called *Wrap Entities With Session*, where a *Session Façade* is created to “manage the business objects, and provide a uniform coarse-grained service access to clients”[4]. The BookKeeper and Retrieval session beans were introduced as examples of a *Session Façades*. These beans reduce the number of remote method invocations between the application and presentation tiers. The session beans use container managed invocations on entity bean methods, which are less costly than remote invocations from the presentation tier[1]. The two session beans provide a service access layer for the presentation tier. By communicating only with the session beans, the presentation tier will be buffered further from the inner workings of the business logic contained in the entity beans. This increases the modularity of the system.

Once the beans were implemented, they were packaged in a `jar` (Java Archive) file along with the appropriate deployment descriptors, and deployed. Deployment descriptors are XML documents used by the JBoss application server to run the deployed beans. The home, component, and enterprise bean interface implementations of each bean must be specified inside the `ejb-jar.xml` file. The names of the beans are put into the JNDI namespace, and instances of the specified beans are pooled for use. The beans are automatically deployed by placing the `jar` file in the JBoss `deploy` folder.

The Enterprise JavaBeans were tested using the JUnit framework. The test classes remotely invoke methods in the application tier, and compare the returned values with values queried directly from the database using JDBC. Testing uncovers bugs in the code, and increases our confidence in the system.

## Java Server Pages

The Java Server Pages (JSP pages) for the organic chemistry tutorial combine static html with dynamic content from the database[2]. The JSP pages are deployed in the presentation tier and are solely responsible for making remote method calls on the session beans through the bean’s remote component interface, and presenting the data given to them.

The first JSP page is a standard login page to allow the user into the system. As previously mentioned, the system needs to be secure from non-users, so the user’s name and password will be checked by the EJB’s before the second page is displayed.

The second JSP page displays individual user data such as the student’s skill level, and the number of correct and incorrect solutions previously submitted by the student. The page also contains a list of skill level ranges a student can choose from so they can work at a comfortable level. The chosen skill level is used by the EJB’s to pick an appropriate problem for the student. The third JSP page then displays all the necessary information about the problem to the student. The reaction is displayed in standard reaction notation much like Figure 1, but only the target molecule is displayed. Possible starting materials

and reagents sets are listed for the student to choose from, and when selected, are displayed in the reaction. Once the solution is complete the problem is returned to the BookKeeper bean, the solution is checked and stored, and the appropriate response is posted.

Organic chemistry faculty will have access to additional JSP pages that display the available administrative tools. These pages will not be visible or accessible to anyone without the necessary privileges. The administrative tools are further described in Future Work.

JSP pages are deployed in the Apache web server. Like EJB's, the JSP pages must be packaged along with the appropriate deployment descriptor. Instead of using an ordinary .jar file extension however, they must be packaged in a web-archive (war) file. The war file is made by using the jar tool to package the class files and deployment descriptors, and then renaming the file with a .war extension.

## **Future Work**

A number of additional enhancements are planned for the system. The first is a set of administrative tools that will allow the organic chemistry faculty to maintain and observe the system. Reactions and molecules will need to be added to the system to build a larger knowledge base or to shape the focus of the problems. Faculty will need to be able to view statistics on individual students, groups of students, or the entire class. The tools will allow this kind of data to be viewed or added without any knowledge of SQL or the MySQL database.

The system was designed to house an intelligent tutoring agent to maximize the effectiveness of the tutorial for students. The tutoring agent will be responsible for selecting an appropriate problem each time for each student. For instance, if a student is having difficulty with a particular type of retrosynthetic reaction, then easier problems of the same type could be given until the student learns the concepts involved. This particular kind of agent would require the grouping of all retrosynthetic reactions by functional group interactions.

The design of the system allows for experimentation on different aspects of and approaches to intelligent tutoring behavior. Adding or removing a tutoring module would require only the deployment or un-deployment of the corresponding EJB. This allows us to combine, compare, and study implementations of different intelligent tutoring agents that use different theoretical approaches or techniques. These different approaches may include the use of fuzzy sets, neural networks, and/or evolutionary computation. The conclusions drawn from these proposed experiments should help us maximize the potential of this application in effectively supplementing undergraduate course material in retrosynthetic analysis.

## References

1. Cavaness, Chuck & Keeton, Brian (2001). *Special Edition Using Enterprise JavaBeans 2.0* [Electronic Version]. Indianapolis, Indiana: Que Publishing.
2. *The J2EE Tutorial* (2002). Retrieved February 9, 2002 from <http://java.sun.com/j2ee/tutorial/>
3. *JBoss 3.0 Documentation* (2001). Retrieved March 1, 2002 from <http://www.jboss.org/online-manual/HTML/index.html>
4. Alur, Deepak, Crupi, John & Malks, Dan (2001). *Core J2EE Patterns: Best Practices and Design Strategies*. Upper Saddle River, New Jersey: Prentice Hall.

## Acknowledgements

We would like to acknowledge the many and valued contributions of the following individuals to this project:

- Nancy Carpenter, our organic chemistry expert
- Janet Kinney, our artificial intelligence consultant
- Matt Hardy, the architect of the early OCT prototypes
- Jeremy Kallstrom, our lab manager who helped with the installation of JBoss