PROJECT PROPOSAL

for

INTELLIGENT MULTI-AGENT SYSTEM FOR REPORTING ADVERSE DRUG REACTIONS

Engineering Design Team

Kevin Morris, Emily Parker, Andrew Rueckert, Reed Tompkins

Faculty Project Advisor

Dr. Yanqing Ji

Liaison Engineer

Dr. Yanqing Ji

Sponsored By

Gonzaga University

Center for Engineering Design

Project EE-3

April 2008
Design Team

Kevin Morris

Emily Parker

Steven Andrew Rueckert

Reed Tompkins

Faculty Advisor

Yanqing Ji
Table of Contents

Introduction and Overview ......................... 3
  Project Overview ................................................................. 3
  Intelligent Agents ................................................................. 4
  The Naranjo Algorithm ....................................................... 6
  Rule-Adding Framework ..................................................... 7
  System Requirements ......................................................... 8
  Deliverables ......................................................................... 9

System Architecture ................................. 10
  Hospital Database ............................................................... 10
  Web-Based Interface .......................................................... 12
  Agent Architecture ............................................................ 12
  “SOAP” Services ............................................................... 15
  Modified Naranjo Algorithm Rules ............................... 16
  Database of Rules ............................................................... 18
  Evaluating a Rule ............................................................... 19
  Overview of the Custom Language .............................. 20
  Rule Operators In The Custom Language ................... 21
  Rule Adding Website .......................................................... 22
  List/Diagram of Web Pages/Servlets ............................ 22
  Reporting Odds Ratio (ROR) .............................................. 26
Conclusion ............................................. 27
  Fulfillment of Goals ................................................................. 27
  Lessons Learned ................................................................. 27
  Future Development .............................................................. 28

Appendicies ........................................... 30
  Appendix A: References .......................................................... 30
  Appendix B: Database Tables .................................................. 30
  Appendix C: Screenshots of Web Interface ............................... 35
  Appendix D: Representative Code Samples .............................. 36
  Appendix E: FDA Spontaneous Reporting Form ....................... 36
Introduction and Overview

Background Information

Adverse Drug Reactions (ADRs) are unexpected side effects brought on by one or more drugs prescribed to treat a patient. In the United States, the Food and Drug Administration uses clinical trials to test drugs for adverse effects before they are released on the market, but the scope of these trials is necessarily limited by the sample size and duration of testing. Thus, the FDA relies on physicians’ spontaneous ADR reports, as they encounter them in the field, in order to identify drug side-effects and potentially pull dangerous drugs from the market.

Unfortunately, studies estimate that nearly 90% of all ADRs go unreported (Hazell). There are many varied reasons for this, among which are the physicians’ lack of time, uncertainty as to the causative drug behind the reaction, and the interpretation of ADR symptoms as new symptoms of the patient’s underlying disease. Our project aims to ameliorate these problems by implementing a system that will monitor a hospital's patient database, suggest possible ADRs for a physician to look at and follow up on, and streamline the ADR reporting process by auto-filling much of the existing ADR reporting forms.

Project Overview

Our project is a continuation of the effort by last year's team. We have enacted several key changes and additions, listed below:

- We’ve moved the project’s database information from Microsoft Access to MySQL.
MySQL, an open source industry standard, is more customizable than Access and provides more features.

- We’ve created a mock hospital database on which to test our algorithm. The database is based on a representative sample of patients taken from the FDA’s database, with additional information added to better simulate the electronic health record system of an actual hospital.

  We changed the project’s focus from statistical data mining to patient-specific calculation. Last year's team used an unmodified copy of the FDA's spontaneous reporting database, which contains tens of thousands of reports from all over the world. A single hospital's database, on which our project is intended to be implemented, will necessarily be less extensive and thus is less suited to the statistical approach. Under our implementation, one can still view the statistical correlation between a drug and a reaction, but we are also using a modified version of the Naranjo algorithm to calculate the probability of an ADR on a per-patient basis.

### Intelligent Agents

We have decided to encapsulate the majority of our server-side functionality within “Intelligent Agent” wrappers. Intelligent Agents, or just “Agents” represent a relatively new programming abstraction in the same vein as Object-Oriented classes; where a class seeks to encapsulate functionality, an agent seeks to encapsulate intentionality. Agents are autonomous, social, reactive, and proactive (Jennings). Agents are fully encapsulated, and have full control over all of their internal state. An outside source cannot forcibly invoke an agent's functionality; however, agents are capable of communicating among themselves as well as with human users.
and operators. A typical arrangement involves requesting a service from an agent, and then waiting for an acknowledgement, or request for additional information. Agents are reactive in that they monitor their computing environment, and certain behaviors will be triggered by external stimuli. Agents are proactive in their goal-oriented behaviors; an agent will not only respond to changes in its computing environment, but initiate communication and actions based on its operating criteria, often referred to as its “desires”.

The encapsulated nature of Intelligent Agents make them ideal for running the backend of our database, as well as performing data-mining and statistical activities. Because agent-interaction is mediated entirely through simple messages passed back and forth, there are very few points of interaction, making our system easily scalable. When a new type of agent is added to the system, it simply has to register the types of services that it is capable of providing with a yellow pages directory, where other agents are free to discover it. Agents are free to act as “black boxes” within our system, advertising services with predefined input and output formats. The agents (and programmers) requesting those services do not need to concern themselves with how or where to services are being performed – they can be confident that any service listed will meet its advertised specifications.

We have decided specifically to use the free and open-source Java Agent DEvelopement (JADE) framework. It provides much of the core code necessary to implement Intelligent Agents, including template agents, yellow pages directory service, debugging tools, and handling all message-passing services. This frees us from many of the implementation details of developing agents, and allows us to focus our time and energy on developing their behaviors and interactions. JADE is an extension to the core Java library, and is licensed under the GNU
The Naranjo Algorithm

The Naranjo Algorithm was developed in 1981, and is intended to help physicians discover and confirm potential adverse drug reactions. The Naranjo Algorithm is still frequently used today in order to confirm adverse drug reactions in studies (Naranjo).

The Naranjo Algorithm works thus: for each patient, a variety of questions are posed. An example question is “Did the adverse drug reaction occur before the drug is applied?” Each question has a set number of answers, each with a point value or “weight”. For instance, answers to the above question might be “Yes” (+2), “Unknown” (+0), and “No” (-1). The score of each patient is calculated by summing up the point values of each answer to each question. Depending on the total number of points accrued, a different label is applied to the adverse drug reaction, such as “likely” or “possible.”

Our team decided to use the Naranjo Algorithm to generate adverse drug reaction signals, but we found that we needed to modify it to suit our project. The Naranjo Algorithm was originally built so that a physician could sit down and use it. Thus, it includes questions such as “Are there alternative causes that might explain the reaction?” However, our system is incapable of possessing previous medical knowledge; questions such as the above example are unanswerable. More generally, our team had to throw out questions that could not be answered from the data available in a medical database.

Information on the specific questions used by our modified Naranjo Algorithm is included in the Systems Architecture portion of this paper.
Rule-Adding Framework

As we developed and implemented our modified Naranjo Algorithm, the project team began to discuss how we could make our system more useful. One suggestion was that since we were already using a modified version of the Naranjo Algorithm, why not let the physicians modify it further according to their knowledge? This simple idea could expand the scope of our project significantly; instead of constructing a system that used the Naranjo Algorithm to signal adverse drug reactions, we were going to construct a framework that allowed physicians to select which signals would be used to indicate a potential adverse drug reaction.

The team quickly rallied behind this idea and it became a central component of our project. Our framework comes with the modified Naranjo Algorithm included, but physicians can add new rules complete with questions based off of information available in their hospital database. Virtually any “question” that can be expressed in English and answered using information from a hospital database can be added to the framework! Physicians have the ability to add their medical expertise to our framework, vastly increasing its usefulness.

The ability to add new signals to the algorithm is presented through an easy-to-use website; there is no need for a physician to “code up” a new signal. Our web interface allows a physician to assign a name to each rule and access data in a hospital database.

More detailed information on the Rule-Adding framework is included in the Systems Architecture portion of this paper.
System Requirements

Our project is a continuation of the work from last year's team. We have enacted several changes and additions to accommodate this year's focus:

- Enhance web interface to focus on patient specific calculations. Probabilities of ADR's will now be calculated by using a modified version of the Naranjo algorithm on a per-patient basis.

- Moving database information from Microsoft Access to MySQL. MySQL is an open source industry standard, is more customizable than Access and provides more features.

- Creating a mock hospital database to store patient and drug information in which to test the modified Narajno algorithm. This database is based on a representative sample of patients taken from the FDA's database, to better simulate an actual hospital database.

- Allow health professionals to add rules that are used to generate probabilities of suspected ADRs. This rule system will allow flexibility when checking specific relationships of ADRs.

- Utilize an intelligent, decision-making multi-agent system to communicate between the web interface and the hospital database.

- Aid healthcare professionals in submitting an ADR by automatically filling relevant data required by the FDA reporting form.
Deliverables

The following list contains all items that will be made available to Gonzaga's School of Engineering and Applied Science:

- Web interface that allows health professionals to check and report suspected adverse drug events, and add rules used to determine associations between drugs and reactions.

- MySQL database containing a mock hospital data used by the web interface and accessed by the multi-agent system.

- Multi-agent system that communicates with the hospital database to provide relevant data to the web interface.

System Architecture

Hospital Database

Our Hospital Database is the basis upon which we will base all of our tests of our web-interface and data-mining algorithms. It will resemble the type of databases used by hospitals as closely as possible in both form and content. The database is a relational database, connected using various keys (Doctor_id, Patient_id, etc.), as well as indexed on certain often-used fields to improve access times. The database will run on the MySQL engine, an industry leader in terms of both functionality and ease-of-use, and will be indexed using the InnoDB algorithm. We decided to revisit last year's team's decision to use Microsoft Access for several reasons, chief among them...
our concerns with Microsoft SQL's less robust feature list. There are several data-mining
techniques that would be next to impossible to perform without the filtering functions present in
MySQL, but lacking in Microsoft's implementation.

It has been populated using raw data from the FDA Adverse Event Reporting System, by
utilizing the following algorithm:

1. Prune out patients missing any data regarding patient age, gender, weight, or drug name,
dose, or therapy start/end dates.
2. Standardize ages by converting from days, weeks, months, or decades into decimal
values of years.
3. Standardize weights by converting from kilograms or grams to pounds.
4. Generate hospital stay duration using a normal distribution with a mean value of 7 days
and a standard deviation of 10 days.
5. Reformat the data from plaintext format to MySQL, and insert it into the database.

Our database includes the following tables:

**ADRs:** This table contains information about the potential ADRs that we have already
discovered. It holds an ADR_Id, a Patient_Id, a Drugname, a Reaction, and a Score, used to
determine the likelihood of this particular ADR. It's primary index is ADR_Id, and it is also
indexed on Patient_Id, Drugname, and Score for faster searching.

**DoctorLogin:** This table contains information about the doctors using our system,
including their usernames and passwords. It holds a DocID, FirstName, LastName, Username,
Password, Address, Email, Phone, and Occupation. The Password field is 1-way MD5 encrypted,
to protect the doctors' privacy. This means that, given a candidate password, the database can
determine if it matches the encrypted password, but there is no way to turn the encrypted
password back into the doctor's plaintext password. The DoctorLogin table's primary index is
DocID, and it is also indexed on Username and Password, for quicker login times.

**Meds:** This table contains the medicines that patients have been on. It holds a PatientID, a
Med_Name, a Start_Date, a Schedule, and an End_Date. It is indexed on Patient_id and
Med_Name.

**Patient:** The patient table holds information regarding the patients in our hospital. It
holds a Patient_id, a DOB (date of birth), an Age, a Sex, a DOD (date of death), a
DOD_Comment, a WT (weight), and a doctor_id (the primary caregiver for the patient). Its
primary key is Patient_id, and it is also indexed on doctor_id for faster searching.

**PendingFDASubmission:** This table holds ADR reports which are in a queue to be
submitted to the FDA. It has 30 fields, which, for brevity's sake, I will not outline here. Its
primary and only index is ADR_ID.

**Reaction:** This table holds symptoms that a patient is experiencing, which may or may
not be related to an ADR. It holds a Patient_id, an Event_Date, an ICD_Code, an ICD_Category,
a Symptom, and a Release_Date. (ICD is a medical industry standard for consistently encoding
different types of symptoms and diseases.) This table is indexed on Patient_id and Symptom.

**RulesList:** This table contains all of the instructions necessary to implement the various
segments of our Modified Naranjo algorithm. It has a RuleNum, Code, RuleName, RuleDetails,
and Active flag. It is primarily indexed on RuleNum, and is also indexed on RuleName and
Active.
For further information regarding the tables in our project, see Appendix B.

Web-Based Interface

We implemented our web interface using Apache, an open-source industry standard server program that allows us to access the project’s web pages over the internet. Furthermore, we used an Apache program called Tomcat to run Java Server Pages (JSP’s) and Servlets. JSPs allow us to embed basic Java code in HTML pages, while Servlets provide the means to modularly run more complicated Java processes, such as database interactions, and forward the results to a JSP.

Agent Architecture

Our system includes five distinct agent-types, three of them coded specifically for this project, and two from the JADE library who serve to facilitate agent-agent and human-agent interactions. Each agent will be detailed below, along with a list of services advertised and behaviors performed.

**JADE Directory Facilitator (DF) Agent:** The DF agent manages the yellow pages directory of all agents and services. It provides the RegisterService and ServiceSearch functions; however, it does not advertise these, as all agents are created knowing how to contact the DF agent. This agent was provided as a standard component of the JADE framework.

**Web Service Integration Gateway (WSIG) Agent:** WSIG is an extension to JADE that allows for external communication with agents, a very necessary component of our system. The
WSIG agent creates a Tomcat web-application which listens for incoming SOAP messages on a specific port. Upon receiving a SOAP message, the WSIG agent “unmarshalls” that message into the format used internally by JADE agents, checks the message headers against the list of services maintained by the Directory Facilitator agent, and forwards the message to the appropriate agent. The WSIG agent waits for a response from that agent, “marshalls” the response back into a SOAP message, and returns it via Tomcat to whomever made the initial response. This agent was provided as an add-on to the JADE system; however some local modifications were made to the marshalling and unmarshalling process, as they were incompatible with our local version of Tomcat. WSIG advertises no services – Tomcat is incapable of accessing the yellow pages directory, and agents are incapable of initiating communications with an unknown host on the internet.

**Medical Authentication Agent:** This agent is the initial point of contact for users of the system. It handles login, signup, and authentication functionality. It is capable of authenticating a doctor’s plaintext password against the one-way encrypted copy that we keep in our database. The Medical Authentication Agent also has a “goal-oriented” behavior of making sure that the Medical Expert agent is kept informed of which doctors are currently using the system. When initialized, the Medical Authentication Agent will periodically poll the Directory Facilitator Agent until it discovers the name of an agent which provides the “medical-expert” service. Once it has found such an agent, it will inform it of any successful logins. The Medical Authentication Agent advertises the *NewUser* and *Login* services, both of which are available within the “MedicalAuthentication” namespace.

**Medical Lookup Agent:** The Medical Lookup Agent provides doctors with information
regarding their patients once they have successfully logged in. It is capable of searching through
the ADR table of the database, providing lists of suspected ADRs for all of the doctor's patients,
as well as calculating statistical information regarding specific ADRs that a doctor might wish to
examine further. It is also responsible for handling our Modified-Naranjo-Rules administration –
both displaying and updating existing rules and adding new ones. The Medical Lookup Agent
advertises the ADRListRequest, ADRDetailRequest, AddRuleRequest, RuleRequest, and
FDASubmissionRequest services, all of which are available within the “MedicalLookup”
namespace.

Medical Expert Agent: The Medical Expert agent is concerned with keeping the list of
ADRs in our database up to date. It is capable of searching through the Patient, Medicine, and
Reaction tables of our database to calculate the Modified-Naranjo score for every
Medicine/Symptom pair (and thus, the likelihood of an ADR) for a given patient, for a given
doctor, or for all patients in the database. As this last option is very computationally intensive, the
Medical Expert Agent typically waits for a Medical Lookup Agent to inform it that a particular
doctor is actively using the system, and at that point makes sure that the information for all of
that doctor's patients is current within the system. (Attempting to fulfil its “desire” to make sure
that no doctor sees out-of-date information.)
“SOAP” Services

We discovered that Agents, while excellent at communicating amongst themselves, are incredibly limited in their communication with the outside world. To combat this shortcoming, we decided to use the Web Service Integration Gateway (WSIG) add-on, available for the JADE framework, which allows for the interpretation of Simple Object Access Protocol (SOAP) requests as Agent-based messages. SOAP is a W3C communications standard, which allows for the publication and access of Web Services over the internet. SOAP provides a standard format for plain-text XML-formatted strings to be interpreted as requests, and for response strings to be interpreted as Object-Oriented responses. SOAP is often considered “middleware” - it is not a
programming language per-se, but rather a data format which allows remote service requests to be passed between different applications, even ones written in different programming languages.

The WSIG add-on mandated a very specific set of definitions be created for what is a “valid” request, and what is not, so we were forced to write a Hospital “Ontology,” consisting of all of the services provided by agents through the WSIG gateway, as well as a very precise definition of all of the objects which those services could be performed upon, and what objects would be returned as a result of those services. For more details regarding the Ontology, see code samples in Appendix D.

The WSIG add-on provided all of the necessary code an applications to interface agents with a SOAP application in the Tomcat server, so all that was required of us was to provide functions to “marshal” java objects into XML strings, and “unmarshal” XML strings back into those same objects on the other side. This is usually a standard service provided by the core javax.xml.soap library; however, we ran into an insurmountable version error between our SOAP library and Tomcat server, and were forced to implement these marshalling and unmarshalling functions ourselves on all “Ontology” classes ourselves. For more information regarding the marshalling/unmarshalling functions, see code examples given in Appendix D.

Modified Naranjo Algorithm Rules

A full list of Naranjo Algorithm questions can be found in the paper cited in the “references” section of this paper. We modified the algorithm twice: first of all, not every
question in the Naranjo Algorithm is answerable without strong medical knowledge, so we had
to eliminate those questions. Next, we presented this modified algorithm to a medical doctor,
who offered us feedback and suggested additional refinements.

Here we present the full list of modified Naranjo Algorithms included in this paper:

- When did the adverse drug reaction occur?
  - Prior to Drug Administration [-1]
  - Unknown [0]
  - Less than two weeks later [+3]
  - Less than one month later [+2]
  - Less than six months later [+1]
  - More than six months later [0]

- Is the patient 65 or older?
  - Yes [+1]
  - Unknown [0]
  - No [-1]

- Did the reaction stop if the drug was no longer administered?
  - Yes, within one week [+2]
  - Yes, within one month [+1]
  - Yes, more than one month later [0]
  - No [0]

- If the patient was given the drug again, did the reaction re-occur?
• Yes, within one month [+2]
• Yes, more than one month later [0]
• No [-3]

Then, the total score for each patient is recorded and a probability that an adverse drug reaction is assigned as follows:

<table>
<thead>
<tr>
<th>Score</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 1</td>
<td>Doubtful</td>
</tr>
<tr>
<td>2 – 4</td>
<td>Possible</td>
</tr>
<tr>
<td>5 – 8</td>
<td>Likely</td>
</tr>
<tr>
<td>9 – 10</td>
<td>Definite</td>
</tr>
</tbody>
</table>

These labels are presented to the physician so that they might decide to file an adverse drug reaction report.

**Database of Rules**

Any rule, whether a part of the modified Naranjo Algorithm or a custom rule added by a physician, is stored in a table in a database. A portion of the table is presented below:
A rule in the table consists of several parts:

- **Rule Name**: This is the rule’s name. An example is “Dechallenge.”

- **Code**: This is the rule’s code. We developed a “custom language” to evaluate rules. Details are in the next section.

- **Rule Details**: These are the specific descriptions of each question in each rule. For instance, the first part of a rule’s code may describe a question that could be translated, as “Is the patient older than 65 years old?” The rule description will then read, “Is the patient older than 65 years old?” The rule descriptions exist to make the understanding of rules simpler and do not affect the evaluation of the rule.

- **Active**: This field simply indicates whether a rule is used when calculating a patient’s total score.

For each active rule in the database, a score is calculated and then summed up to give a possibility that a drug/reaction pair is an adverse drug reaction.

### Evaluating a Rule

The “code” portion of the above table is critical, as it describes fully each rule and each question within the rule, including scores.

The “RuleEvaluator” class found in the appendices translates the code into a SQL query. JADE agents can then use this SQL query to interact with the database and return results. Results are used to calculate the patient’s score for each rule.

For instance, say we had a rule that asked “Is the patient greater than 65 years old?” The
code for this rule would be translated into a SQL query that would return each row in the 
database that contains the desired patient’s id and an age value greater than 65. If a row was 
correctly returned, we know the patient is older than 65 years, so the appropriate score should be 
assigned. However, if no row was returned, we know that the patient is younger than 65, so a 
different score should be assigned for this rule.

Questions within a rule follow an “if-then” form. IF a patient is greater than 65, THEN the 
score is 2. IF the reaction occurred within two weeks of drug administration, THEN the score is 
3. An “if-then” format makes evaluating rules simple; if the SQL query correctly returns a row 
for the given parameters, then the “if” portion of the question evaluated true. We can then assign 
a score for that given rule. If no rows were returned, the question evaluated false, so we must 
move on.

Overview of the Custom Language

This section describes the custom language that is used to make up the “Code” section of a 
rule. As explained previously, this custom language is translated into a SQL command that is 
then executed.

Generally, a rule follows this format: First of all, a table and a value within that table are 
listed. Then an operator is listed, followed by a constant. The table/value, the operator, and the 
constant combine to form a question. A rule can have multiple questions within it, and questions 
are almost always in “if-then” form. Example questions are ‘is the patient taking drug X,” and 
“Is the patient greater than 85 years old?” A score follows each question. If the question is 
answered “yes”, then an appropriate score is assigned to the rule. Otherwise, a new question can 
be evaluated. If there are no questions left, the default score is assigned to the rule. Here is an
example rule:

Age  
This refers to a value within a table (below)

Patient  
This refers to a table within the hospital database

less than  
This is an operator. So far, we can translate the question as “is the patient’s age less than…"

65  
This is a constant. Constants follow operators (typically). Now can translate the question as “Is the patient’s age less than 65?”

2  
This is the score that is assigned if the above question translates as true.

drule  
This marks the end of all the questions in the rule. This rule had just one question and one score, but rules could have more.

0  
This is the default score. This score is returned if no question in the rule is evaluated as true.

Rule Operators In The Custom Language

The following is a list of every operator that is usable in our custom rule language.

- Less than
- Less / equal
- Greater than
- Greater / equal
- Is
- Is not
• Between [“Between” uses two operators]
• Diff [“Diff” operates on two different values (i.e., drug date and reaction date) and returns the difference between the two]

Rule Adding Website

The rule-adding website allows a physician to describe and add new rules to the rules database. Physicians are able to specify any number of questions (consisting of tables, values, operators, and constants) within a single rule. The score for each question and the default score are configurable. Physicians may also configure the name of the rule and descriptions of questions in the rule. A screenshot of the rule-adding page is included in the appendices.

List/Diagram of Web Pages/Servlets

Our web interface utilizes the Java programming language. All web pages used in this project are JSP pages. JSP pages enable the use of Java to create dynamically generated web pages based on HTML, which looks extremely similar to a web page seen on the internet. We then use servlets to pass data from JSP pages to agents to serve database requests. Our purpose in using Java is to obtain relevant information from the database and present the data to health professionals dynamically. In this way we can retrieve data from a database that could be constantly changing. NetBeans serves as our integrated development environment to write all of our code for both the web pages and agents. The web application follows the MVC pattern (figure 1), which stands for Model-View-Controller. The model is essentially a Java class, which serves as communication with the database, in which our agents provide the interface for. The view acts as a visible part of the application, where the user can see the data given by the
controller portion in their browser. The controller serves as the middle layer, consisting of servlets, which receive input from JSP pages and provides a response by interfacing with classes in the model layer, which are our agents.

![Block Diagram of MVC Model](image)

**Figure 1 - Block Diagram of MVC Model**

The web interface begins with a login page. Authorized users will enter username and password to gain access to the next pages. For administrators, upon logging in, there will be four options to choose from. The first option, view rules, allows an administrator to see a list of all the rules in the system, along with whether they are active or not. The second option, add rule, allows administrators to create new rules to evaluate suspected ADRs based on specific criteria. The next option, set rules, allows administrators to turn on and off rules. The last option allows administrators to add a new doctor to the authorized users that can login. Authorized doctors will have a homepage that displays a navigation pane, and a table of the doctor's own patients with ADR id numbers, patient id numbers, drug names, reaction names, and the score calculated using the modified Naranjo algorithm. The navigation pane allows the user to return to the home page, submit a report, get help for a certain page, change their settings, and log out.
The different files in the web interface and their purposes are as follows:

**docLogin.jsp** - Allows an authorized doctor or administrator to log in.

**loginServlet.java** - Interacts with RequestBroker to validate login.
docHomePage.jsp - Homepage for doctors to see patients and their ADRs.
viewDetails.jsp - Allows user to view more details on a specific patient's ADR.
docHelp.jsp - Explains meanings of various web pages seen throughout the website.
docSettings.jsp - Currently not implemented.
viewReport.jsp - Allows user to submit a report for a suspected ADR.
viewDetailsServlet.java - Interacts with RequestBroker to get details on patient's ADR.
submitReportServlet.java - Interacts with RequestBroker to process a ADR report.
adminHomePage.jsp - Homepage for administrators.
viewRules.jsp - Allows administrator to view all rules and their current on/off state.
viewRulesServlet.java - Interacts with RequestBroker to obtain all rules.
addRule.jsp - Allows administrator to create a new rule.
processRuleServlet.java - Interacts with RequestBroker to add a new rule.
addRulesServlet.java - Serves as a intermediate between the JSP pages.
setRules.jsp - Allows administrator to change on/off states of rules.
processSetRules.java - Interacts with RequestBroker to change on/off state of rules.
setRulesServlet.java - Serves as a intermediate between the JSP pages.
addDoctor.jsp - Allows Administrator to add a new doctor to the system.
processDoctorServlet.java - Interacts with RequestBroker to add a doctor to the system.
addDoctorServlet.java - Serves as a intermediate between the JSP pages.
RequestBroker.java - Interacts with agents to perform the duties requested by servlets.
Agents - Contains all the classes needed to interact with the database and provide information back to the RequestBroker.
Reporting Odds Ratio (ROR)

The reporting odds ratio (ROR) is a tool used to measure causal relationships between an adverse event and a drug. Based on statistical data gathered from ADR Reports provided by the FDA, a two by two contingency table can be populated in which to calculate the ROR of a given drug and ADR:

<table>
<thead>
<tr>
<th>Reports with the suspected</th>
<th>Reports without the suspected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reports with the suspected</td>
<td>ADR</td>
</tr>
<tr>
<td>drug</td>
<td>a</td>
</tr>
<tr>
<td>All other reports</td>
<td>c</td>
</tr>
</tbody>
</table>

With this data, ROR is expressed as:

\[
ROR = \frac{A \times D}{B \times C}
\]

The calculated ROR value represents a signal strength related to the data provided. The higher the ROR, the stronger the signal or causal relationship of a drug and ADR pair is. Unmodified from last year's project, a signal strength of 19 or higher indicates a strong causal relationship between a drug and ADR pair, otherwise the pair is considered as having a weak causal relationship.
Conclusion

Fulfillment of Goals

Our goals for this project were to implement the Naranjo algorithm, integration of intelligent agents into the system, and to improve website design and functionality. Implementation of the Naranjo algorithm was successful. Through our advisor we got in contact with a doctor and received expert opinions on the operation of the algorithm. This included revision of some the rules, and application of different weights for certain criteria. Integration of Intelligent agents also was a success. Our agents successfully communicate with our servlets to provide data from the database. With agents we have the ability to seamlessly introduce more agents that provide different services, providing a large degree of scalability. Another success was the overhaul of the web interface. By giving administrators the ability to modify the criteria in which ADRs are found by adding rules, we have more flexibility and adaptability to changes over time. Also, creating a homepage for doctors to immediately view their patients and associated ADRs, as well as providing a more intuitive interface to interact with makes the website more functional. Overall, the project was a success, meeting all the goals we set out and producing a working system.

Lessons Learned

The most important lesson taken away from this project relates to the difficulty of integration. For the bulk of the project, the team was split up until into sub-teams; one worked primarily with the website, Naranjo Algorithm, and creating custom rules, whereas the other
worked primarily with the JADE Agents. The team correctly identified ahead of time that integrating these two portions of the project would be non-trivial, which absolutely turned out to be true. Integrating the two fully functional halves of the projects took over three weeks, which was especially hard as it came at the end of the school semester. Fortunately, the team had built into the schedule extra time to deal with integration, but the team still had to put in twelve-hour days towards the end.

Closely related to the problem of integration was the problem of communication. Forming sub-teams allowed each sub-team to focus on what they needed to; however, sometimes communication between the two groups was not ideal. This lead to some confusion and wasted work as one sub-team would discover that the other had expected something to work differently. This problem improved dramatically towards the end of the project, however.

Lastly, the team learned the importance of weekly status meetings and having a clear schedule. Though there was some crunch time at the end, discussing upcoming tasks at weekly meetings and keeping to the schedule meant that the project was always under control. At no point did the team feel like the project was not going to work out, which helped keep morale up and lead to what we believe is a solid senior design project.

Future Development

While complete in its own respect, this project can be further improved upon by the efforts of another team next year. We’d suggest they start with the following:

- Expansion of the mock hospital database. There are a significant amount of details requested by the FDA’s submission request process that the database does not currently provide. In particular, the Drug table should include information about the manufacturer,
expiration date, etc. Also, a table detailing various lab tests ordered for each patient could be instrumental in adding new rules to the Naranjo algorithm since many previous computerized ADR-detection programs relied extensively on lab tests.

- Automating the electronic submission of ADR information to the FDA, or to a printable PDF. Our project has provided a framework for gathering the information needed in an ADR report and saving it in a table in the database. The next step is to make it easy to send this information directly to the FDA.

- Continual refinement of the Naranjo algorithm. We’d suggest meeting with a physician, if possible, to formulate additional rules and refine the ADR-detection process.

- Further development of agent autonomy. Currently, the agents perform tasks in a pre-programmed manner, while the JADE framework allows for a more creative application of agents.
Appendicies

Appendix A: References


Appendix B: Database Tables

**ADRs**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADR_Id</td>
<td>int(11)</td>
</tr>
<tr>
<td>Patient_Id</td>
<td>int(11)</td>
</tr>
<tr>
<td>Drugname</td>
<td>text</td>
</tr>
<tr>
<td>Reaction</td>
<td>text</td>
</tr>
<tr>
<td>Score</td>
<td>double</td>
</tr>
</tbody>
</table>

**DoctorLogin**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DocID</td>
<td>int(11)</td>
</tr>
<tr>
<td>FirstName</td>
<td>text</td>
</tr>
<tr>
<td>LastName</td>
<td>text</td>
</tr>
<tr>
<td>Username</td>
<td>text</td>
</tr>
<tr>
<td>Password</td>
<td>text</td>
</tr>
<tr>
<td>Address</td>
<td>text</td>
</tr>
<tr>
<td>Email</td>
<td>text</td>
</tr>
<tr>
<td>Phone</td>
<td>text</td>
</tr>
<tr>
<td>Occupation</td>
<td>text</td>
</tr>
</tbody>
</table>
### Meds

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient_id</td>
<td>int(10)</td>
</tr>
<tr>
<td>Med_Name</td>
<td>text</td>
</tr>
<tr>
<td>Start_Date</td>
<td>date</td>
</tr>
<tr>
<td>Schedule</td>
<td>text</td>
</tr>
<tr>
<td>End_Date</td>
<td>date</td>
</tr>
</tbody>
</table>

### Patient

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient_id</td>
<td>int(11)</td>
</tr>
<tr>
<td>id</td>
<td>int(10)</td>
</tr>
<tr>
<td>DOB</td>
<td>date</td>
</tr>
<tr>
<td>Age</td>
<td>int(11)</td>
</tr>
<tr>
<td>Sex</td>
<td>varchar(1)</td>
</tr>
<tr>
<td>DOD</td>
<td>date</td>
</tr>
<tr>
<td>DOD_Comment</td>
<td>text</td>
</tr>
<tr>
<td>WT</td>
<td>double</td>
</tr>
<tr>
<td>doctor_id</td>
<td>int(11)</td>
</tr>
</tbody>
</table>
PendingFDASubmission

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADR_ID</td>
<td>int(11)</td>
<td>Drug_Frequency</td>
<td>text</td>
</tr>
<tr>
<td>Patient_ID</td>
<td>text</td>
<td>Drug_Route</td>
<td>text</td>
</tr>
<tr>
<td>PatientDOB</td>
<td>text</td>
<td>Drug_Start_Date</td>
<td>text</td>
</tr>
<tr>
<td>Patient_Sex</td>
<td>text</td>
<td>Drug_End_Date</td>
<td>text</td>
</tr>
<tr>
<td>Patient_Weight</td>
<td>text</td>
<td>Drug_Reason_for_Use</td>
<td>text</td>
</tr>
<tr>
<td>Patient_DOD</td>
<td>text</td>
<td>Drug_Dechallenge</td>
<td>text</td>
</tr>
<tr>
<td>Reaction_Date</td>
<td>text</td>
<td>Drug_Rechallenge</td>
<td>text</td>
</tr>
<tr>
<td>Report_Date</td>
<td>text</td>
<td>Drug_Lot_Num</td>
<td>text</td>
</tr>
<tr>
<td>Reaction_Desc</td>
<td>text</td>
<td>Drug_Expiration_Date</td>
<td>text</td>
</tr>
<tr>
<td>Relevant_Tests</td>
<td>text</td>
<td>Doctor_Name</td>
<td>text</td>
</tr>
<tr>
<td>Other_Relevant_History</td>
<td>text</td>
<td>Doctor_Address</td>
<td>text</td>
</tr>
<tr>
<td>Drug_Name</td>
<td>text</td>
<td>Doctor_Email</td>
<td>text</td>
</tr>
<tr>
<td>Drug_Strength</td>
<td>text</td>
<td>Doctor_Phone</td>
<td>text</td>
</tr>
<tr>
<td>Drug_Manufact</td>
<td>text</td>
<td>Doctor_Occupation</td>
<td>text</td>
</tr>
<tr>
<td>Drug_Dose</td>
<td>text</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reaction

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient_id</td>
<td>int(10)</td>
</tr>
<tr>
<td>Event_Date</td>
<td>date</td>
</tr>
<tr>
<td>ICD_Code</td>
<td>float</td>
</tr>
<tr>
<td>ICD_Category</td>
<td>int(11)</td>
</tr>
<tr>
<td>Symptom</td>
<td>text</td>
</tr>
<tr>
<td>Release_Date</td>
<td>date</td>
</tr>
</tbody>
</table>
RulesList

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>RuleNum</td>
<td>int(11)</td>
</tr>
<tr>
<td>Code</td>
<td>text</td>
</tr>
<tr>
<td>RuleName</td>
<td>text</td>
</tr>
<tr>
<td>RuleDetails</td>
<td>text</td>
</tr>
<tr>
<td>Active</td>
<td>tinyint(1)</td>
</tr>
</tbody>
</table>

Appendix C: Screenshots of Web Interface

![Screenshot of Adverse Drug Reaction Monitoring System](image-url)
Appendix D: Representitive Code Samples

Due to the incredibly large volume of code in conjunction with the highly formulaic nature of many segments of code, we had decided to include representitive samples of code, rather than the source of our project in its entirety. All code samples are attached at the end of this paper.

Appendix E: FDA Spontaneous Reporting Form

Attached, find the FDA Spontaneous Reporting Form required to submit an ADR report.
package edu.gonzaga.cpen.hospitalagents;

import jade.content.AgentAction;
import jade.content.ContentElement;
import jade.content.lang.sl.SLCodec;
import jade.content.onto.basic.Action;
import jade.content.onto.basic.Done;
import jade.content.onto.basic.Result;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.domain.DFService;
import jade.domain.FIPANames;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.FIPAManagementOntology;
import jade.domain.FIPAAgentManagement.Property;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.util.leap.ArrayList;
import java.sql.*;
import java.util.Date;
import edu.gonzaga.cpen.rules.RuleWriter;
import java.util.Vector;
import org.apache.log4j.Logger;

public class LookupAgent extends Agent {
    private DatabaseBroker dbb = new DatabaseBroker();
    private static final boolean debug = true;

    public static final String WSIG_FLAG = "wsig";
    public static final String WSIG_MAPPER = "wsig-mapper";
    public static final String WSIG_PREFIX = "wsig-prefix";

    private Connection dbCon; //db connection

    private Logger log = Logger.getLogger(LookupAgent.class.getName());
    private SLCodec codec = new SLCodec();
    private Date startDate;

    private static final String[] services = {"MedicalLookup"};

    @Override
    protected void setup() {
        log.info("LookupAgent starting...");
        log.info("Agent name: "+getLocalName());

        // Get agent arguments
        Object[] args = getArguments();

        // Register codec/onto
        getContentManager().registerLanguage(codec);
        getContentManager().registerOntology(FIPAManagementOntology.getInstance());
        getContentManager().registerOntology(HospitalOntology.getInstance());
    }
}
// Prepare a DFAgentDescription
DFAgentDescription dfad = new DFAgentDescription();
dfad.setName(this.getAID());
dfad.addLanguages(codec.getName());
dfad.addProtocols(FIPANames.InteractionProtocol.FIPA_REQUEST);

// publish the MedicalLookup service
for (String serviceName : services) {
    ServiceDescription sd;
    sd = new ServiceDescription();
    sd.addLanguages(codec.getName());
    sd.addProtocols(FIPANames.InteractionProtocol.FIPA_REQUEST);
    sd.setType("LookupAgent");
    sd.setOwnership("LookupOwner");
    sd.addOntologies(HospitalOntology.getInstance().getName());

    // WSIG properties
    sd.addProperties(new Property(WSIG_FLAG, "true"));

    // Service name
    if (args.length >= 1) {
        serviceName = "MedicalLookup";
    }
    log.info("Service name: "+serviceName);
    sd.setName(serviceName);
    dfad.addServices(sd);
}
// DF registration
try {
    DFService.register(this, dfad);
} catch (Exception e) {
    log.error("Problem during DF registration", e);
    e.printStackTrace();
    doDelete();
}

log.info("LookupAgent started");
startDate = new Date();

// Add math behaviour
this.addBehaviour(new CyclicBehaviour(this) {
    private MessageTemplate template =
        MessageTemplate.MatchOntology(
            HospitalOntology.getInstance().getName());

    // if LookupAgent receives a message posted by WSIG, it will
    // execute that request.
    // Else, it will wait until a message is received.
    public void action() {
        log.info("Action started");
        ACLMessage msg = myAgent.receive(template);
        if (msg != null) {
            Action actExpr;
            try {
                actExpr =
                    (Action) myAgent.getContentManager().extractContent(msg);
AgentAction action = (AgentAction) actExpr.getAction();

log.info("Execute action: "+action.getClass().getSimpleName());
if (action instanceof ADRListRequest){
    serveADRListRequest((ADRListRequest) action, actExpr, msg);
} else if (action instanceof ADRDetailRequest) {
    serveADRDetailRequest((ADRDetailRequest) action, actExpr, msg);
} else if (action instanceof AddRule) {
    serveAddRuleRequest((AddRule) action, actExpr, msg);
} else if (action instanceof RuleRequest) {
    serveRuleRequest((RuleRequest) action, actExpr, msg);
} else if (action instanceof RuleListRequest) {
    serveRuleListRequest((RuleListRequest) action, actExpr, msg);
} else if (action instanceof GetAgentInfo) {
    serveGetAgentInfoAction((GetAgentInfo) action, actExpr, msg);
} else if (action instanceof FDASubmissionRequest) {
    serveFDASubmissionRequest((FDASubmissionRequest) action, actExpr, msg);
} catch (Exception e) {
    e.printStackTrace();
    log.error("Error serving action", e);
private void serveADRListRequest(ADRListRequest lr, Action actExpr, ACLMessage msg) {
    ADRList result = dbb.serveADRListRequest(lr);
    if (debug) {
        System.out.println("Got " + result.getVectorADRs().size() + " results!");
    }
    sendNotification(actExpr, msg, ACLMessage.INFORM, result);
}

private void serveADRDetailRequest(ADRDetailRequest dr, Action actExpr, ACLMessage msg) {
    ADRRecord result = dbb.serveADRDetailRequest(dr);
    System.out.println("adrID: " + result.getAdrID() + \\
                       "Drug: " + result.getDrug() + \\
                       "Patient: " + result.getPatientID() + \\
                       "Result: " + result.getReaction());
    sendNotification(actExpr, msg, ACLMessage.INFORM, result);
}

private void serveAddRuleRequest(C/ADRRequest req, Action actExpr, ACLMessage msg) {
    System.out.println("Processing AddRuleRequest..." + req);
AddRule ar, Action actExpr, ACLMessage msg) {
    Rule result = dbb.serveAddRuleRequest(ar);
    if (debug) {System.out.println("ID: "+result.getId());}
    sendNotification(actExpr, msg, ACLMessage.INFORM, result);
}

private void serveRuleRequest(
    RuleRequest rr, Action actExpr, ACLMessage msg) {
    Rule result = dbb.serveRuleRequest(rr);
    sendNotification(actExpr, msg, ACLMessage.INFORM, result);
}

private void serveRuleListRequest(
    RuleListRequest rlr, Action actExpr, ACLMessage msg) {
    RuleList result = dbb.serveRuleListRequest(rlr);
    sendNotification(actExpr, msg, ACLMessage.INFORM, result);
}

private void serveFDASubmissionRequest(
    FDASubmissionRequest fdasub, Action actExpr, ACLMessage msg) {
    FDASubmissionResponse result =
        dbb.serveFDASubmissionRequest(fdasub);
    sendNotification(actExpr, msg, ACLMessage.INFORM, result);
}

private void serveGetAgentInfoAction(
    GetAgentInfo getAgentInfo, Action actExpr, ACLMessage msg) {
    AgentInfo result = new AgentInfo();
    result.setAgentAid(getAID());
}
result.setStartDate(startDate);
sendNotification(actExpr, msg, ACLMessage.INFORM, result);
}

private void sendNotification(
    Action actExpr, ACLMessage request, int performative, Object result) {
    // Send back a proper reply to the requester
    ACLMessage reply = request.createReply();
    if (performative == ACLMessage.INFORM) {
        reply.setPerformative(ACLMessage.INFORM);
        try {
            ContentElement ce = null;
            if (result != null) {
                // If the result is a java.util.List, convert it
                // into a jade.util.leap.List to make the ontology happy
                if (result instanceof java.util.List) {
                    ArrayList l = new ArrayList();
                    l.fromList((java.util.List) result);
                    result = l;
                }
                ce = new Result(actExpr, result);
            } else {
                ce = new Done(actExpr);
            }
            getContentManager().fillContent(reply, ce);
        }
        catch (Exception e) {
            e.printStackTrace();
            log.error("Agent " + getName()
                + ": Unable to send notification", e);
        }
    }
}
} else {
    reply.setPerformative(performative);
    if (result instanceof String) {
        reply.setContent((String)result);
    }
    reply.addUserDefinedParameter(ACLMessage.IGNORE_FAILURE, "true");
    send(reply);
}

protected void takeDown() {
    // Deregister from the DF
    try {
        DFService.deregister(this);
    } catch (Exception e) {
        log.error("Error in DF deregistration", e);
    }
    log.info("Listener stopped");
}
package edu.gonzaga.cpen.hospitalagents;

import java.util.AbstractMap;
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.core.behaviours.TickerBehaviour;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.*;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import java.util.Vector;
import edu.gonzaga.cpen.rules.RuleEvaluator;

public class MedicalExpertAgent extends Agent {
    //todo
private DatabaseBroker dbb = new DatabaseBroker();

public void setup() {
    // publish the MedicalExpert service
    DFAgentDescription dfd = new DFAgentDescription();
    dfd.setName(getAID());
    ServiceDescription sd = new ServiceDescription();
    sd.setType("medical-expert");
    sd.setName(getLocalName()+"-medical-expert");
    dfd.addServices(sd);
    try {
        DFService.register(this, dfd);
    } catch (FIPAException fe) {
        fe.printStackTrace();
    }
    // add the listening behavior
    addBehaviour(new ListenForQueries());

    // Load database driver
    try {
        Class.forName("com.mysql.jdbc.Driver");
    } catch (Exception e) {
        e.printStackTrace();
        return;
    }
}

// update the ADRs table
public void updateTable() {

RuleEvaluator re = new RuleEvaluator();
int numPatients;
Vector<ADRRecord> alladrs = new Vector<ADRRecord>();
numPatients = dbb.getMaxPatientId();

for (int i=1; i<=numPatients; i++) {
    ADRList patientadrs = re.returnScore(i);
    alladrs.addAll(patientadrs.getVectorADRs());
}

for (ADRRecord r : alladrs) {
    dbb.addADRRecord(r);
}

//update the ADRs for a particular patient
public void updatePatient(String patientid) {
    RuleEvaluator re = new RuleEvaluator();
    ADRList adrs = re.returnScore(Integer.parseInt(patientid));
    for (Object r : adrs.getVectorADRs()) {
        dbb.addADRRecord((ADRRecord)r);
    }
}

//update the ADRs table for a doctor as they log in
public void updateForDoctor(String doctorID) {
    Vector<String> patientIDs = dbb.getPatientsByDoctor(doctorID);
    for (String id : patientIDs) {
        updatePatient(id);
    }
}
class ListenForQueries extends CyclicBehaviour {
    // we're listening for queries
    private MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
    public void action() {
        ACLMessage msg = receive(mt);
        if (msg != null) {
            // do things
            ACLMessage response = msg.createReply();
            if (msg.getContent().contains("ALL")) {
                response.setPerformative(ACLMessage.CONFIRM);
                myAgent.send(response);
                updateTable();
            } else if (msg.getContent().contains("Patient:")) {
                response.setPerformative(ACLMessage.CONFIRM);
                myAgent.send(response);
                String patientid = msg.getContent().split(":")[1];
                updatePatient(patientid);
            } else if (msg.getContent().contains("Doctor:")) {
                String doctorid = msg.getContent().split(":")[1];
                updateForDoctor(doctorid);
            } else {
                response.setPerformative(ACLMessage.DISCONFIRM);
                myAgent.send(response);
            }
        }
    }
}
public void takeDown() {
    try {
        DFService.deregister(this);
    } catch (FIPAException fe) {
        fe.printStackTrace();
    }
}
package edu.cpen.gonzaga.agentbroker;
import edu.gonzaga.cpen.hospitalagents.*;
import javax.xml.soap.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import java.util.*;
import java.io.*;
import java.net.*;

public class RequestBroker
{
    //This function is called by all public methods and is responsible for
    //marshalling and sending the request object and unmarshalling the response
    private static void sendRequest(
        SOAPSerializable response, SOAPSerializable request, String namespace)
    {
        try
        {
            XMLParser xp = new XMLParser();

            //build the connection
SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
SOAPConnection con = scf.createConnection();

// build the message
MessageFactory mf = MessageFactory.newInstance();
SOAPMessage message = mf.createMessage();

// define the parts of the message
SOAPPart sp = message.getSOAPPart();
javax.xml.soap.SOAPEnvelope env = sp.getEnvelope();
SOAPBody body = env.getBody();

// add the wsig namespaces to the envelope
env.addNamespaceDeclaration("urn", namespace);
env.addNamespaceDeclaration("xsd", "http://www.w3.org/2001/XMLSchema");
env.addNamespaceDeclaration("xsi", "http://www.w3.org/2001/XMLSchema-instance");

// marshal the request object as child element of Body
request.bodyMarshal(body, env);

// save our message
message.saveChanges();

// send the message and get a reply
ByteArrayOutputStream bis = new ByteArrayOutputStream();
message.writeTo(bis);
String smsg = bis.toString();
System.out.println("SMSG: "+smsg);
String reply = sendHTTPRequest(smsg);

if (reply.equals("FAILURE")) {
    System.out.println("SOAP Request Failed: "+reply);
} else {
    System.out.println("SOAP SUCCESS! "+reply);
    response.unmarshal(reply);
}

// clean up our connection
con.close();

} catch (javax.xml.soap.SOAPException ex) {
    ex.printStackTrace();
} catch (Exception e) {
    // oh god I hope this works.
    e.printStackTrace();
}

private static String sendHTTPRequest(String req) {
    HttpURLConnection httpConn = null;
    byte[] byteMessage = req.getBytes();
    String resp = "FAILURE";
    try {
        httpConn = (HttpURLConnection) new URL(url).openConnection();
        httpConn.setDoOutput(true);
        httpConn.setRequestMethod("POST");
        httpConn.setRequestProperty("Content-Type", "text/xml");
        httpConn.setRequestProperty("Content-Length", String.valueOf(byteMessage.length));
        httpConn.getOutputStream().write(byteMessage);
        httpConn.getOutputStream().flush();
        httpConn.getOutputStream().close();
        InputStream in = httpConn.getInputStream();
        BufferedReader br = new BufferedReader(new InputStreamReader(in));
        String inputLine = null;
        String result = "FAILURE";
        while ((inputLine = br.readLine()) != null) {
            result = inputLine;
        }
        br.close();
        in.close();
        return result;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return resp;
}
// Create the connection
URL url = new URL("http://localhost:8080/wsig/ws");
URLConnection connection = url.openConnection();
httpConn = (HttpURLConnection) connection;

// Set the appropriate HTTP parameters.
httpConn.setRequestProperty("Content-Length", String.valueOf( byteMessage.length ) );
httpConn.setRequestProperty("Content-Type", "text/xml; charset=utf-8");
httpConn.setRequestProperty("SOAPAction", "");
httpConn.setRequestMethod( "POST" );
httpConn.setDoOutput(true);
httpConn.setDoInput(true);

// Send the soap message
OutputStream out = httpConn.getOutputStream();
out.write(byteMessage);
out.close();

if (httpConn.getResponseCode() == HttpURLConnection.HTTP_OK) {
    System.out.println("Uuhhhh, HTTP_OK, des!");
    // Read the response
    InputStreamReader isr =
        new InputStreamReader(httpConn.getInputStream());
    BufferedReader in = new BufferedReader(isr);

    String inputLine;
    StringBuffer sb = new StringBuffer();
while ((inputLine = in.readLine()) != null) {
    sb.append(inputLine);
}

in.close();

resp = sb.toString();
System.out.print("RESPONSE: "+resp);
} else {
    System.out.println("ERROR MESSAGE: 
    +httpConn.getResponseMessage()
    +"\n\tCode :"+httpConn.getResponseCode());
}
}
catch (Exception e) {
    System.out.println("sendHTTPRequest failed");
e.printStackTrace();
}
return resp;
}

public static ADRRecord serveADRDetailRequest(ADRDetailRequest request)
{
    ADRRecord response = new ADRRecord();
    sendRequest(response, request, "MedicalLookup");
    return response;
public static ADRList serveADRListRequest(ADRListRequest request) {
    ADRList response = new ADRList();
    sendRequest(response, request, "MedicalLookup");
    return response;
}

public static Rule serveAddRule(AddRule request) {
    Rule response = new Rule();
    sendRequest(response, request, "MedicalLookup");
    return response;
}

public static Doctor serveLoginRequest(Login login) {
    Doctor doctor = new Doctor();
    doctor.setFailure(true);
    sendRequest(doctor, login, "MedicalAuthentication");

    return doctor;
}

public static Doctor serveNewUserRequest(NewUser request) {
    Doctor response = new Doctor();
    sendRequest(response, request, "MedicalAuthentication");
public static RuleList serveRuleListRequest(RuleListRequest request)
{
    RuleList response = new RuleList();
    sendRequest(response, request, "MedicalLookup");
    return response;
}

double return response;

double public static RuleList serveRuleListRequest(RuleListRequest request)
{
    RuleList response = new RuleList();
    sendRequest(response, request, "MedicalLookup");
    return response;
}

public static Rule serveRuleRequest(RuleRequest request)
{
    Rule response = new Rule();
    sendRequest(response, request, "MedicalLookup");
    return response;

}

public static FDASubmissionResponse serveFDASubmissionRequest(
    FDASubmissionRequest request)
{
    FDASubmissionResponse response = new FDASubmissionResponse();
    sendRequest(response, request, "MedicalLookup");
    return response;

}
package edu.gonzaga.cpen.rules;
import edu.gonzaga.cpen.hospitalagents.*;
import java.sql.*;
import java.util.*;
import java.lang.*;

/**
 * @author R. Tompkins & K. Morris
 */
public class RuleEvaluator {

    private Connection con;
    final static boolean debug = false;

    /*This function, given a patientID, returns an ADRList with the Score, Drug, and Reaction information filled out appropriately */
    public ADRList returnScore(int patientID)
    {
        int recordCounter = 0;
    }
/* Classes for Agents */
ADRList tempList = new ADRList();
Vector<ADRRecord> tempRecords = new Vector<ADRRecord>();

/* Holds the necessary data */
Vector<Integer> activeRuleIndexes = new Vector<Integer>();
Vector<String> drugNames = new Vector<String>();
Vector<String> reacNames = new Vector<String>();
Vector<Double> scoreList = new Vector<Double>();
/*Get the rule numbers for every active rule */
activeRuleIndexes = returnActiveRuleIndexes();

/* Get drug names for our patient */
drugNames = returnDrugNamesFromDB(patientID);

/* Get list of reactions for our patient */
reacNames = returnReacNamesFromDB(patientID);

/* Load up the tempRecords vector! */
for(int i = 0; i < activeRuleIndexes.size(); i++, recordCounter++)
{
    for(int j = 0; j < reacNames.size(); j++, recordCounter++)
    {
        for(int k = 0; k < drugNames.size(); k++, recordCounter++)
        {
            tempRecords.add(
                returnScore(
                        patientID, drugNames.get(k), reacNames.get(j) ));
        }
    }
}
tempList.setADRs(tempRecords);

return tempList;

/*This function, given a patientID, returns an ADRLRecord with the Score, Drug, and Reaction information filled out appropriately */
public ADRRecord returnScore(int patientID, String Drug, String Reaction)
{
    int totalScore;

    /* Classes for Agents */
    ADRRecord tempRecord = new ADRRecord();

    /* Holds the necessary data */
    Vector<Integer> activeRuleIndexes = new Vector<Integer>();

    Vector<RuleResult> resultList = new Vector<RuleResult>();
    /*Get the rule numbers for every active rule */
    activeRuleIndexes = returnActiveRuleIndexes();

    /* Load up the resultList vector! */
    for(int i = 0; i < activeRuleIndexes.size(); i++)
    {
        resultList.add(
returnRuleResult(
    patientID, activeRuleIndexes.get(i), Drug, Reaction));
}

tempRecord.setPatientID(Integer.toString(patientID));
tempRecord.setDrug(Drug);
tempRecord.setReaction(Reaction);
tempRecord.setScoreList(resultList);
totalScore = 0;
/* Calculate the total score */
    for(int i = 0; i < resultList.size(); i++)
        totalScore += resultList.get(i).getScore();
tempRecord.setTotalScore(totalScore);

    return tempRecord;
}

/* Given a patientID, this function accesses the "Meds" DB
 * and returns a Vector
 * containing strings of each Med_Name */
public Vector<String> returnDrugNamesFromDB(int patientID)
{
    Statement stmt = null;
    ResultSet result = null;
    Vector<String> drugNames = new Vector<String>();

    //open the connection
    openConnection();
//Create a query to get all drug names for the given patient
String queryString =
    new String("SELECT Med_Name FROM Meds WHERE Patient_id = "
    + Integer.toString(patientID)+ " GROUP BY Med_Name");

//create a statement
try {
    stmt = con.createStatement();
} catch (SQLException sqle){
    System.err.println("Failed to create a statement "+ sqle);
}

//execute query
try {
    result = stmt.executeQuery(queryString);
} catch (Exception e) {
    e.printStackTrace();
}

//put the drug names into the Vector
try {
    while(result.next() ) {
        drugNames.add(result.getString("Med_Name"));
    }
}
catch(SQLException sqle) {
    System.err.println("Failed to get field data "+ sqle);
    return null; //"unknown"
}

//Return the vector
return drugNames;
}

/* Given a patientID, this function accesses the "Reaction" DB and
 * returns a Vector
 * containing strings of each Symptom */
public Vector<String> returnReacNamesFromDB(int patientID)
{
    Statement stmt = null;
    ResultSet result = null;
    Vector<String> reacNames = new Vector<String>();

    //open the connection
    openConnection();

    //Create a query to get all reaction strings for the given patient
    String queryString =
        new String("SELECT Symptom FROM Reaction WHERE Patient_id = "
        + Integer.toString(patientID) + " GROUP BY Symptom");
try {
    stmt = con.createStatement();
} catch (SQLException sqle) {
    System.err.println("Failed to create a statement "+ sqle);
}

//execute query
try {
    result = stmt.executeQuery(queryString);
} catch (Exception e) {
    e.printStackTrace();
}

//put the reactions into the Vector
try {
    while(result.next()) {
        reacNames.add(result.getString("Symptom"));
    }
} catch(SQLException sqle) {
    System.err.println("Failed to get field data "+ sqle);
    return null;    //"unknown"
}
//Return the vector
    return reacNames;

    }

    /* This function queries the "RuleLists" database and returns the RuleNum
     * for each rule that is active */
    private Vector<Integer> returnActiveRuleIndexes()
    {
        Statement stmt = null;
        ResultSet result = null;
        Vector<Integer> ruleIndices = new Vector<Integer>();

        //open the connection
        openConnection();

        //create a string to select all rule nums
        String queryString =
            new String("SELECT RuleNum FROM RulesList WHERE Active = 1");

        //create a statement
        try {
            stmt = con.createStatement();
        }
        catch (SQLException sqle){
            System.err.println("Failed to create a statement "+ sqle);
        }

        //execute query
        C:/Users/Andrew/Documents/School/Senior Design/final paper/RuleEvaluator.java
    }

2010.04.21  23:29:01
try {
    result = stmt.executeQuery(queryString);
}
catch (Exception e) {
    e.printStackTrace();
}

//put the active rule numbers into the Vector
try {
    while(result.next() ) {
        ruleIndices.add(result.getInt("RuleNum"));
    }
}
catch(SQLException sqle) {
    System.err.println("Failed to get field data "+ sqle);
    return null;   //"unknown"
}

//Return the vector
return ruleIndices;

/* This function queries the "RuleLists" database and returns the RuleNum
 * for each rule that is active */
private Vector<String> returnActiveRuleNames()
{
    Statement stmt = null;

ResultSet result = null;
Vector<String> ruleNames = new Vector<String>();

//open the connection
openConnection();

//create a string to select all rule nums
String queryString =
    new String("SELECT RuleName FROM RulesList WHERE Active = 1");

//create a statement
try {
    stmt = con.createStatement();
} catch (SQLException sqle) {
    System.err.println("Failed to create a statement "+ sqle);
}

//execute query
try {
    result = stmt.executeQuery(queryString);
} catch (Exception e) {
    e.printStackTrace();
}

//put the active rule numbers into the Vector
try {

}
while(result.next() ) {
    ruleNames.add(result.getString("RuleName"));
}

} catch(SQLException sqle) {
    System.err.println("Failed to get field data " + sqle);
    return null;  //"unknown"
}

//Return the vector
return ruleNames;

/*This function, given a patientID and ruleNumber, returns the score for
 * that patient for that rule for that med/reaction pair */
public double returnRuleScore(int patientID, int ruleNumber, String drugName, String reacName) {
    Statement stmt = null;
    ResultSet result = null;
    Vector<String> ruleDefinition = new Vector<String>();
    String totalCode = new String();
    String[] splitCode = new String[50];
    int score;
// open connection
openConnection();

// create string
String queryString = new String("SELECT Code" +
    " FROM RulesList WHERE RuleNum = " + Integer.toString(ruleNumber));

try {
    stmt = con.createStatement(); // create a statement
}
catch (SQLException sqle) {
    System.err.println("Failed to create a statement "+ sqle);
}

// execute query
try {
    result = stmt.executeQuery(queryString);
}
catch (Exception e) {
    e.printStackTrace();
}

// load up the code in the ruleslist
try {
    while(result.next() ) {
        totalCode = result.getString("Code");
    }
}

catch(SQLException sqle) {
    System.err.println("Failed to get field data "+ sqle);
    return (Integer)null;  //"unknown"
}

//split the code into an array separated by newlines
splitCode = totalCode.split("\r\n");

for(int i = 0; i < splitCode.length; i++)
{
    ruleDefinition.add(splitCode[i]);
}
/*ruleDefinition now contains ruleNumber's definition */

score = returnRuleDecipher(ruleDefinition, patientID, drugName, reacName);

closeConnection();

return score;
}

/*This function, given a patientID and ruleNumber, returns the score for
 * that patient for that rule for that med/reaction pair */
public RuleResult returnRuleResult(
    int patientID, int ruleNum, String drugName, String reacName)
{
    Statement stmt = null;
ResultSet result = null;
Vector<String> ruleDefinition = new Vector<String>();
String totalCode = new String();
String[] splitCode = new String[50];
String ruleName = new String();
RuleResult ruleResult = new RuleResult();
int score;

//open connection
openConnection();

//create string
String queryString = new String("SELECT Code, RuleName" +
   " FROM RulesList WHERE RuleNum = " + ruleNum);

try {
    stmt = con.createStatement(); //create a statement
} catch (SQLException sqle){
    System.err.println("Failed to create a statement "+ sqle);
}

//execute query
try {
    result = stmt.executeQuery(queryString);
} catch (Exception e) { 
    e.printStackTrace();
}
try {
    while(result.next()) {
        totalCode = result.getString("Code");
        ruleName = result.getString("RuleName");
    }
}

catch(SQLException sqle) {
    System.err.println("Failed to get field data " + sqle);
    return (RuleResult)null;    //"unknown"
}

//split the code into an array separated by newlines
splitCode = totalCode.split("\r\n");

for(int i = 0; i < splitCode.length; i++)
    { ruleDefinition.add(splitCode[i]);
    } /*ruleDefinition now contains ruleNumber's definition */

score = returnRuleDecipher(ruleDefinition, patientID, drugName, reacName);

ruleResult.setRuleName(ruleName);
ruleResult.setScore(score);
/* Given a Vector containing the definition of the rule and the patientID, this
 * function returns the score for that rule */
private int returnRuleDecipher(
    Vector<String> ruleDefinition, int patientID, String drugName, String reacName)
{
    Statement stmt = null;
    ResultSet result = null;
    int score;
    int i = 0;
    boolean[] resultFound = new boolean[100];
    boolean statementTrue = true;
    for(int j = 0; j < 100; j++)
        resultFound[j] = false;
    int andIndex = 0;
    String queryStr = null;

    /* Make sure drugName, medName are okay with apostophees */

drugName = drugName.replace("'", "''");
reacName = reacName.replace("'", "''");

    Vector<String> parameter = new Vector<String>();
    Vector<String> table = new Vector<String>();
    Vector<String> operator = new Vector<String>();
for(int j = 0; j < ruleDefinition.size(); j++)
{
    if(debug == true) System.out.println(ruleDefinition.get(j));
}

while((ruleDefinition.get(i).compareTo("endrule")) != 0)
{
    queryStr = null;
    parameter.clear();
    table.clear();
    operator.clear();

    do {
        //**********************************************************
        //Get parameters, associated tables
        //**********************************************************
        parameter.add(ruleDefinition.get(i).replace("'", "'"));
        i++;
        table.add(ruleDefinition.get(i).replace("'", "'"));
        i++;
        operator.add(ruleDefinition.get(i).replace("'", "'"));
        i++;
    } while (operator.lastElement().equalsIgnoreCase("diff")
             );

    queryString = "SELECT ";
    for(int j = 0; j < parameter.size(); j++)
    {

if(j != 0) {
    queryStr = queryStr.concat(",
    queryStr = queryStr.concat(table.get(j) + "." + parameter.get(j));
}

/* We need to pull from the Meds and Reaction Tables, because we always
* need to make sure we are checking for a particular drug and reaction */
queryStr = queryStr.concat(" FROM Meds, Reaction");
for(int j = 0; j < table.size(); j++) {
    /* Pull from other tables if necessary*/
    if((table.get(j).compareTo("Meds") != 0)
        && (table.get(j).compareTo("Reaction") != 0)) {
        queryStr = queryStr.concat(",
        queryStr = queryStr.concat (table.get(j));
    }
}

//**************
//END parameters, associated tables
//**************

//operator

queryStr = queryStr.concat(" WHERE ");
if (debug == true) System.out.println(operator.lastElement());
    //
if(operator.get(0).equals("diff"))
{
    queryStr = queryStr.concat("DATEDIFF(" + table.get(0) + ", "
            + parameter.get(0) + ", " + table.get(1) + ", "
            + parameter.get(1) + ")");
}
else
{
    queryStr = queryStr.concat(table.lastElement() + "."
            + parameter.lastElement());
}

if(operator.lastElement().equals("greater than"))
{
    queryStr = queryStr.concat(" > " + ruleDefinition.get(i));
    i++;
}
else if(operator.lastElement().equals("less than"))
{
    queryStr = queryStr.concat(" < " + ruleDefinition.get(i));
    i++;
}
else if(operator.lastElement().equals("less / equal"))
{
    queryStr = queryStr.concat(" <= " + ruleDefinition.get(i));
    i++;
}
else if(operator.lastElement().equals("great / equal"))
{
    queryStr = queryStr.concat(" >= " + ruleDefinition.get(i));
    i++;
}
else if(operator.lastElement().equals("is"))
{
    queryStr = queryStr.concat(" = " + ruleDefinition.get(i));
    i++;
}
else if(operator.lastElement().equals("is not"))
{
    queryStr = queryStr.concat(" != " + ruleDefinition.get(i));
    i++;
}
else if(operator.lastElement().equals("between"))
{
    queryStr = queryStr.concat(" BETWEEN " + ruleDefinition.get(i));
    i++;
    queryStr = queryStr.concat(" AND " + ruleDefinition.get(i));
    i++;
}
queryStr = queryStr.concat(" AND ");
{
    for(int j = 0; j < (table.size() - 1); j++)
    {
        queryStr = queryStr.concat(table.get(j) + ".Patient_id = 
                     + table.get(j+1) + ".Patient_id AND ");
    }
    queryStr = queryStr.concat(table.get(0) + ".Patient_id = "
                      + Integer.toString(patientID) + " AND Meds.Med_Name = " + drugName + 
                      + "' AND Reaction.Symptom = " + reacName + "' GROUP BY "
                      + table.get(0) + ".Patient_id");
}

//Gather results
try {
    stmt = con.createStatement(); //create a statement
}
catch (SQLException sqle){
    System.err.println("Failed to create a statement "+ sqle);
}

//execute query
try {
    result = stmt.executeQuery(queryStr);
}
catch (Exception e) {
e.printStackTrace();
}

if(debug == true) System.out.println(queryStr);

// check results
try {
    while(result.next()) {
        resultFound[andIndex] = true;
    }
}
catch(SQLException sqle) {
    System.err.println("Failed to get field data "+ sqle);
    return (Integer)null;    // "unknown"
}

// dealing with AND
if(ruleDefinition.get(i).equalsIgnoreCase("and")) {
    andIndex++;
    i++;
    continue; // go back to top of while loop
}

for(int j = 0; j < andIndex + 1; j++) {
    if(resultFound[j] && statementTrue )
        statementTrue = true;
else
{
    statementTrue = false;
    break;
}

if(statementTrue == true )
    return Integer.parseInt(ruleDefinition.get(i));
else
    i++;       //move to next test
//DONE cleaning........................

} //end while
i++;
return Integer.parseInt(ruleDefinition.get(i));

} //end function

private void openConnection() {
    try {
        try {

Class.forName("com.mysql.jdbc.Driver");
}
catch (Exception e) {
    System.out.println("Failed to load JDBC driver." + e);
}
String dbURL = "jdbc:mysql://localhost/Hospital"; //Database location
String username = "root";
String password = "wp2dgwty";
con = DriverManager.getConnection(dbURL, username, password);
}
catch (Exception e) {
    e.printStackTrace();
}

private void closeConnection() {
    try {
        con.close();
    }
catch (Exception e) {
        e.printStackTrace();
    }
}
package edu.gonzaga.cpen.rules;

import java.sql.*;
import java.util.*;
import java.lang.*;

public class RuleWriter {

    private Connection con;
    private Vector<String> code;
    private Vector<String> ruleName;
    private Vector<String> ruleDetails;
    final static boolean debug = false;

    public void setCode(Vector<String> codeVector) {
        code = codeVector;
    }

    public void setRuleName(Vector<String> rulenameVector) {
        ruleName = rulenameVector;
    }
}
public void setRuleDetails(Vector<String> ruledetailsVector) {
    ruleDetails = ruledetailsVector;
}

public void insertRule() {
    Statement stmt = null;
    openConnection();

    String insertString =
        new String("INSERT INTO RulesList (Code, RuleName, RuleDetails, Active) VALUES ('");
    for (int i = 0; i < code.size(); i++) {
        insertString = insertString.concat(code.get(i) + "\r\n");
    }
    insertString = insertString.concat("','");
    for (int i = 0; i < ruleName.size(); i++) {
        insertString = insertString.concat(ruleName.get(i) + "\r\n");
    }
    insertString = insertString.concat("','");
for(int i = 0; i < ruleDetails.size(); i++)
{
    insertString = insertString.concat(ruleDetails.get(i) + "\n\n");
}
insertString = insertString.concat(", '1'");
if(debug == true) System.out.println(insertString);
try {
    stmt = con.createStatement(); //create a statement
}
catch (SQLException sqle) {
    System.err.println("Failed to create a statement "+ sqle);
}
//execute update
try {
    stmt.executeUpdate(insertString);
}
catch (Exception e) {
    e.printStackTrace();
}
}

public void openConnection() {
try {
    try {
    Class.forName("com.mysql.jdbc.Driver");
    }
catch (Exception e) {
    System.out.println("Failed to load JDBC driver." + e);
}
String dbURL = "jdbc:mysql://localhost/Hospital";  // Database location
String username = "root";
String password = "wp2dgwty";
con = DriverManager.getConnection(dbURL, username, password);
}
catch (Exception e) {
    e.printStackTrace();
}
}

private void closeConnection() {
    try {
        con.close();
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
ProcessSetRules Servlet
Functions that package new and updated rules, and send them to the Agents to be saved in the database.
By Reed Tompkins & Kevin Morris
*************************************************************************

package Servlets;

import edu.cpen.gonzaga.agentbroker.RequestBroker;
import edu.gonzaga.cpen.hospitalagents.AddRule;
import edu.gonzaga.cpen.hospitalagents.Rule;
import edu.gonzaga.cpen.hospitalagents.RuleList;
import edu.gonzaga.cpen.hospitalagents.RuleListRequest;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Vector;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**************************************************************************
* @author adr
*
/**************************************************************************

public class processSetRules extends HttpServlet {

/**
 * Processes requests for both HTTP <code>GET</code> and <code>POST</code> methods.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */

protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        RuleListRequest ruleListRequest = new RuleListRequest();
        RuleList ruleList = new RuleList();
        String ruleID = (String) request.getParameter("ruleID");
        String returnMessage = new String();
        int position = -1;

        ruleList = RequestBroker.serveRuleListRequest(ruleListRequest);

        Vector<Rule> rules = new Vector(ruleList.getVectorRules());

        System.out.println("rule ID: " + ruleID);

        for(int i = 0; i < rules.size(); i++)
        {
            if(rules.get(i).getId().equals(ruleID))
            {...
if(position == -1)
{
    returnMessage = "Error setting rule!";
    return;
}
else
{
    if(rules.get(position).getActive() == true)
    {
        rules.get(position).setActive(false);
        returnMessage = "Rule # " + rules.get(position).getId() + " has been turned off!";
    }
    else
    {
        rules.get(position).setActive(true);
        returnMessage = "Rule # " + rules.get(position).getId() + " has been turned on!";
    }
    /* Have to convert from a rule to AddRule for agents to work */
    AddRule updatedRule = new AddRule();
    updatedRule.setActive(rules.get(position).getActive());
    updatedRule.setCode(rules.get(position).getCode());
    updatedRule.setName(rules.get(position).getName());
    updatedRule.setRuleID(rules.get(position).getId());
    Vector<String> tempRules = new Vector<String>(rules.get(position).getVectorTestDescriptions());
    updatedRule.setTestDescriptions(tempRules);
RequestBroker.serveAddRule(updatedRule);

request.setAttribute("returnMessage", returnMessage);

String url = "/adminHomePage.jsp";
RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(url);
dispatcher.forward(request, response);
} finally {
    out.close();
}

/**
 * Handles the HTTP <code>GET</code> method.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
processRequest(request, response);
}

/**
 * Handles the HTTP <code>POST</code> method.
 * @param request servlet request
 * @param response servlet response
 * @throws ServletException if a servlet-specific error occurs
 * @throws IOException if an I/O error occurs
 */

* @param response servlet response
* @throws ServletException if a servlet-specific error occurs
* @throws IOException if an I/O error occurs
*/
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    processRequest(request, response);
}

/**
 * Returns a short description of the servlet.
 * @return a String containing servlet description
 */
@Override
public String getServletInfo() {
    return "Short description";
} // </editor-fold>
<!-- Document : addRule.jsp
   Created on : Feb 7, 2010, 1:25:00 PM
   Author : Kevin Morris and Reed Tompkins
   Description: Web page that allows doctors to add/edit rules
-->

%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
%@ page import="java.sql.*" %>
%@ page import="java.io.*" %>
%@ page import="java.util.*" %>

<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Administrator Page</title>
      <script type="text/javascript">
         // functions go here
         // this function disables/enables elements as user fills form
         function IDVisible(id)
         {
            document.getElementById(id).style.display = "table-row";
         }
         function IDInvisible(id)
         {
            document.getElementById(id).style.display = "none";
         }
      </script>
   </head>
   <body>
      <!-- body content goes here -->
   </body>
</html>
// this function runs when page starts
function DisableAll()
{
  var i;
  for(i = 1; i < 140; i++)
  {
    document.getElementById(i).style.display = "none";
  }
}
</script>
</head>
<body onload="DisableAll()">
<%
Vector<String> tableName = new Vector<String>();
Vector<Vector<String>> colNames = new Vector<Vector<String>>();
try {
  String connectionURL = "jdbc:mysql://localhost/Hospital";
  Connection connection = null;
  String[] types = {"TABLE"};
  Class.forName("com.mysql.jdbc.Driver");
  // remember to hide this
  String username = "root";
  String password = "wp2dgwty";
  connection = DriverManager.getConnection(connectionURL, username, password);

  // code to interact with database goes here
} catch (Exception e) {
  e.printStackTrace();
}

</%>
// Gets the database metadata
DatabaseMetaData dbmd = connection.getMetaData();
// Specify the type of object; in this case we want tables
ResultSet resultSet = dbmd.getTables(null, null, "%", types);
// Get the table names
while (resultSet.next())
{
    // Get the table name
    tableName.add(resultSet.getString(3));
}

// Create a result set
Statement stmt = connection.createStatement();
ResultSetMetaData rsmd;
int numColumns;

//comment this well
Vector<String> temp = new Vector<String>();
for(int j = 0; j < tableName.size(); j++)
{
    colNames.add(temp);
    resultSet = stmt.executeQuery("SELECT * FROM " + tableName.get(j));
    rsmd = resultSet.getMetaData();
    numColumns = rsmd.getColumnCount();
    // Get the column names; column indices start from 1
    for (int i=1; i<numColumns+1; i++)
    {
        colNames.get(j).add(rsmd.getColumnName(i));
    }
}
// close all the connections.
resultSet.close();
stmt.close();
connection.close();
} catch (Exception ex) {
    out.println("Unable to connect to database.");
}

<h1>Hello Administrator!</h1>

<form action="processRuleServlet" method="post">
    <table>
        <tr><td>Name of Rule:</td></tr>
        <tr><td><input type="text" name="RuleName" onkeypress="IDVisible(1)"></td></tr>

        <%  int i = 1; int id = 1;
        while(i < 10)
        {
            %>
            <tr id="<%=id%>"><td>Description of test number <%=i%>:
                <input type="text" name="RuleDesc<%=i%>" onkeypress="IDVisible(<%=id+1%>)"></td></tr>
            <%id++;%>
            </td></tr>
        </td></tr>
    </table>

    <tr><td>What variable?:
        <select name="RuleVar<%=i%>" onchange="IDVisible(<%=id+1%>)">
            <option>--SELECT--</option>
        </select></td></tr>
</form>
<%for(int j = 0; j < tableName.size(); j++){
    for(int k = 0; k < colNames.get(j).size(); k++){%>
        <option><%=tableName.get(j)%>.<%=colNames.get(j).get(k)%></option>
    <%}}%>
</select></td></tr>
<%id++;%>
<tr id="<%=id%>"><td>What operator do you want?:<br>
<select name="Rule<%=i%>">
    <option>--SELECT--</option>
    <option onclick="IDVisible(<%=id+1%>);
        IDInvisible(<%=id+2%>);
        IDInvisible(<%=id+3%>);
        IDInvisible(<%=id+4%>);
        IDInvisible(<%=id+5%>);
        IDInvisible(<%=id+6%>);">greater than</option>
    <option onclick="IDVisible(<%=id+1%>);
        IDInvisible(<%=id+2%>);
        IDInvisible(<%=id+3%>);
        IDInvisible(<%=id+4%>);
        IDInvisible(<%=id+5%>);
        IDInvisible(<%=id+6%>);">less than</option>
    <option onclick="IDVisible(<%=id+1%>);
        IDInvisible(<%=id+2%>);
        IDInvisible(<%=id+3%>);
        IDInvisible(<%=id+4%>);
        IDInvisible(<%=id+5%>);
        IDInvisible(<%=id+6%>);">great / equal</option>
    </select>
</tr>
IDInvisible(<%=id+3%>); 
IDInvisible(<%=id+4%>); 
IDInvisible(<%=id+5%>); 
IDInvisible(<%=id+6%>); "<option onclick="IDVisible(<%=id+1%>); 
IDInvisible(<%=id+2%>); 
IDInvisible(<%=id+3%>); 
IDInvisible(<%=id+4%>); 
IDInvisible(<%=id+5%>); 
IDInvisible(<%=id+6%>); ">less / equal</option> 
<option onclick="IDVisible(<%=id+1%>); 
IDInvisible(<%=id+2%>); 
IDInvisible(<%=id+3%>); 
IDInvisible(<%=id+4%>); 
IDInvisible(<%=id+5%>); 
IDInvisible(<%=id+6%>); ">is</option> 
<option onclick="IDVisible(<%=id+1%>); 
IDInvisible(<%=id+2%>); 
IDInvisible(<%=id+3%>); 
IDInvisible(<%=id+4%>); 
IDInvisible(<%=id+5%>); 
IDInvisible(<%=id+6%>); ">is not</option> 
<option onclick="IDVisible(<%=id+2%>); 
IDVisible(<%=id+3%>); 
IDInvisible(<%=id+1%>); 
IDInvisible(<%=id+4%>); 
IDInvisible(<%=id+5%>); 
IDInvisible(<%=id+6%>); ">between</option> 
<option onclick="IDVisible(<%=id+4%>); 
IDInvisible(<%=id+1%>); 
IDInvisible(<%=id+2%>); 
IDInvisible(<%=id+3%>); ">diff</option> 
</select></td></tr> 
<%id++;%> 
<tr id="<%=id%>"><td>Constant:

C:/Users/Andrew/Documents/School/Senior Design/workspaces/HospitalAgents/src/NewClass.java

6.1 of 9 2010.04.21 23:38:31
<table>
<thead>
<tr>
<th>ID</th>
<th>Note: If constant is not a number, put single quotes around your entry. (ex. 'text')</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>&lt;input type=&quot;text&quot; name=&quot;RuleConst%=$i%&quot; onkeypress=&quot;IDVisible(%=$id+8%)&quot;&gt;</td>
</tr>
<tr>
<td>ID</td>
<td>&lt;tr id=&quot;%=$id%&quot;&gt;Between Constant 1:&lt;br&gt; &lt;input type=&quot;text&quot; name=&quot;Between%=$i%Const1&quot; onkeypress=&quot;IDVisible(%=$id+7%)&quot;&gt;</td>
</tr>
<tr>
<td>ID</td>
<td>&lt;tr id=&quot;%=$id%&quot;&gt;Between Constant 2:&lt;br&gt; &lt;input type=&quot;text&quot; name=&quot;Between%=$i%Const2&quot; onkeypress=&quot;IDVisible(%=$id+6%)&quot;&gt;</td>
</tr>
<tr>
<td>ID</td>
<td>&lt;tr id=&quot;%=$id%&quot;&gt;What variable?:&lt;br&gt; &lt;select name=&quot;DiffVar%=$i%&quot; onchange=&quot;IDVisible(%=$id+1%)&quot;&gt; %for(int j = 0; j &lt; tableName.size(); j++){ %for(int k = 0; k &lt; colNames.get(j).size(); k++){ %option&gt;&lt;%=tableName.get(j)%&gt;.&lt;%=colNames.get(j).get(k)%&gt;&lt;/option&gt; %} %&lt;/select&gt;</td>
</tr>
<tr>
<td>ID</td>
<td>&lt;tr id=&quot;%=$id%&quot;&gt;What operator do you want?:&lt;br&gt; &lt;select name=&quot;DiffOp%=$i%&quot; onchange=&quot;IDVisible('%=$id+1%');&quot;&gt; %option&gt;--SELECT--&lt;/option&gt; %option onclick=&quot;IDVisible('%=$id+1%'); IDInvisible('%=$id+2%'); IDInvisible('%=$id+3%');&quot;&gt;greater than&lt;/option&gt; %option onclick=&quot;IDVisible('%=$id+1%'); IDInvisible('%=$id+2%');&quot;</td>
</tr>
</tbody>
</table>

```java
C:/Users/Andrew/Documents/School/Senior Design/workspaces/HospitalAgents/src/NewClass.java
```

C:/Users/Andrew/Documents/School/Senior Design/workspaces/HospitalAgents/src/NewClass.java

```
```
<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Less than</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Greater / equal</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Less / equal</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Is</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Is not</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Between</td>
<td></td>
</tr>
</tbody>
</table>

```
IDInvisible(<<%=id+3%>>);">less than</option>
<option onclick="IDVisible(<<%=id+1%>>);
IDInvisible(<<%=id+2%>>);
IDInvisible(<<%=id+3%>>);">great / equal</option>
<option onclick="IDVisible(<<%=id+1%>>);
IDInvisible(<<%=id+2%>>);
IDInvisible(<<%=id+3%>>);">less / equal</option>
<option onclick="IDVisible(<<%=id+1%>>);
IDInvisible(<<%=id+2%>>);
IDInvisible(<<%=id+3%>>);">is</option>
<option onclick="IDVisible(<<%=id+1%>>);
IDInvisible(<<%=id+2%>>);
IDInvisible(<<%=id+3%>>);">is not</option>
<option onclick="IDVisible(<<%=id+2%>>);
IDVisible(<<%=id+3%>>);
IDInvisible(<<%=id+1%>>);">between</option>
</select></td></tr>
<%id++;%>
<tr id="<%=id%>"
<td>Constant:<br>
<input type="text" name="DiffConst<%=i%>"
onkeypress="IDVisible(<<%=id+3%>>)"/>
<%id++;%>

<tr id="<%=id%>"
<td>Diff Constant 1:<br>
<input type="text" name="Diff<%=i%>Between1"
onkeypress="IDVisible(<<%=id+2%>>)"/>
<%id++;%>

<tr id="<%=id%>"
<td>Diff Constant 2:<br>
<input type="text" name="Diff<%=i%>Between2"
onkeypress="IDVisible(<<%=id+1%>>)"/>
```
<tr id="<%=id%>"><td>Score:<br>
<input type="text" name="Score<%=i%>"
    onkeypress="IDVisible(<%=id+1%>); IDVisible(118)""></td></tr>
<tr id="<%=id%>"><td><input type="checkbox" name="Option<%=i%>"
    onclick="IDVisible(<%=id+1%>); IDInvisible(118); " +
    " IDInvisible(119); IDInvisible(<%=id%>)">
    Check here to add another test case!</td></tr>
<tr id="118"><td>Default Score:<br>
<input type="text" name="RuleDefaultScore"
    onkeypress="IDVisible(119)"></td></tr>
<tr id="119"><td><input type="submit" value="Submit"></td></tr>
</table>
</form>
</body>
</html>
Web.xml file - instructs the Tomcat serve how to interpret and serve requests

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee " +
   "http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>loginServlet</servlet-name>
    <servlet-class>Servlets.loginServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>viewDetailsServlet</servlet-name>
    <servlet-class>Servlets.viewDetailsServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>addDoctorServlet</servlet-name>
    <servlet-class>Servlets.addDoctorServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>processDoctorServlet</servlet-name>
    <servlet-class>Servlets.processDoctorServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>addRulesServlet</servlet-name>
    <servlet-class>Servlets.addRulesServlet</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>processRuleServlet</servlet-name>
    <servlet-class>Servlets.processRuleServlet</servlet-class>
  </servlet>
</web-app>
```
<servlet-class>Servlets.processRuleServlet</servlet-class>
</servlet>
<servlet>
  <servlet-name>viewRulesServlet</servlet-name>
  <servlet-class>Servlets.viewRulesServlet</servlet-class>
</servlet>
<servlet>
  <servlet-name>deleteRulesServlet</servlet-name>
  <servlet-class>Servlets.deleteRulesServlet</servlet-class>
</servlet>
<servlet>
  <servlet-name>setRulesServlet</servlet-name>
  <servlet-class>Servlets.setRulesServlet</servlet-class>
</servlet>
<servlet>
  <servlet-name>processSetRules</servlet-name>
  <servlet-class>Servlets.processSetRules</servlet-class>
</servlet>
<servlet>
  <servlet-name>submitReportServlet</servlet-name>
  <servlet-class>Servlets.submitReportServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>loginServlet</servlet-name>
  <url-pattern>/loginServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>viewDetailsServlet</servlet-name>
  <url-pattern>/viewDetailsServlet</url-pattern>
</servlet-mapping>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>addDoctorServlet</servlet-name>
    <url-pattern>/addDoctorServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>processDoctorServlet</servlet-name>
    <url-pattern>/processDoctorServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>addRulesServlet</servlet-name>
    <url-pattern>/addRulesServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>processRuleServlet</servlet-name>
    <url-pattern>/processRuleServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>viewRulesServlet</servlet-name>
    <url-pattern>/viewRulesServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>setRulesServlet</servlet-name>
    <url-pattern>/setRulesServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>deleteRulesServlet</servlet-name>
    <url-pattern>/deleteRulesServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>processSetRules</servlet-name>
  <url-pattern>/processSetRules</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>submitReportServlet</servlet-name>
  <url-pattern>/submitReportServlet</url-pattern>
</servlet-mapping>

<session-config>
  <session-timeout>30</session-timeout>
</session-config>

<welcome-file-list>
  <welcome-file>docLogin.jsp</welcome-file>
</welcome-file-list>
</web-app>
package edu.gonzaga.cpen.hospitalagents;

import java.sql.*;
import java.util.Vector;
import edu.gonzaga.cpen.rules.RuleEvaluator;
import edu.gonzaga.cpen.statistics.Patient;

/**
 * @author adr
 */

public class DatabaseBroker
{
    Connection con;
    final static String dbURL = "jdbc:mysql://localhost/Hospital";
    final static String dbDriver = "com.mysql.jdbc.Driver";
    final static String dbLogin = "root";
    final static String dbPass = "wp2dgwty";
    final static boolean debug = false;

    public Doctor serveLoginRequest(Login login) {
        Doctor result = new Doctor();
        
        return result;
    }
}
Statement stmt = null;
ResultSet resultSet = null;

//open the database connection
openConnection();

//create lookup query
String query = "SELECT * FROM DoctorLogin WHERE Username = '"
    + login.getUsername().replace("'", "\'\") + "' AND Password = MD5('"
    + login.getPassword() + "isadoc'\")";

try {
    //get query results
    if (debug) {System.out.println("Query: "+query);} 
    stmt = con.createStatement();
    resultSet = stmt.executeQuery(query);

    //if results, grab the max
    if (resultSet.next()) {
        result.setFirstName(resultSet.getString("FirstName"));
        result.setLastName(resultSet.getString("LastName"));
        result.setUsername(resultSet.getString("Username"));
        result.setDocID(resultSet.getString("DocID"));
        result.setDocAddress(resultSet.getString("Address"));
        result.setDocEmail(resultSet.getString("Email"));
        result.setDocPhone(resultSet.getString("Phone"));
        result.setDocOccupation(resultSet.getString("Occupation"));

        result.setFailure(false);
    }
}
public ADRRecord serveADRDetailRequest(ADRDetailRequest dr) {
    ADRRecord result = new ADRRecord();
    Statement stmt = null;
    ResultSet resultset = null;
    RuleEvaluator re = new RuleEvaluator();

    // Load database driver
    openConnection();

    //define query
    String query = "SELECT * FROM ADRs WHERE ADR_Id = "
        + dr.getAdrID();
    try {
        //get query results
        if (debug) {System.out.println("Query: "+query);}
        stmt = con.createStatement();
        resultset = stmt.executeQuery(query);
//load each result into an ADRRecord, add it to the vector
if (resultset.next()) {
    result = re.returnScore(resultset.getInt("Patient_Id"),
        resultset.getString("Drugname"), resultset.getString("Reaction"));
    //find ror
    Patient ror =
        new Patient(resultset.getString("Drugname"),resultset.getString("Reaction"));
    result.setCorrelation(ror.getROR());
    result.setAdrID(resultset.getString("ADR_Id"));
}
}
catch (Exception e) {
    e.printStackTrace();
}

//close db connection
closeConnection();

return result;
}

public ADRList serveADRListRequest(ADRListRequest lr) {
    ADRList result = new ADRList();
    Vector<ADRRecord> ADRs = new Vector<ADRRecord>();
    Statement stmt = null;
    ResultSet resultset = null;

    // Load database driver
openConnection();

//define query
String query = "SELECT * FROM ADRs WHERE Patient_Id in " +
    " (SELECT Patient_Id FROM Patient" +
    " WHERE Doctor_Id = " + lr.getDoctorID() + ")";
try {
    //get query results
    if (debug) {System.out.println("Query: " + query);}
    stmt = con.createStatement();
    resultset = stmt.executeQuery(query);

    //load each result into an ADRRecord, add it to the vector
    while (resultset.next()) {
        ADRRecord r = new ADRRecord();
        r.setAdrID(resultset.getString("ADR_Id"));
        r.setDrug(resultset.getString("Drugname"));
        r.setPatientID(resultset.getString("Patient_ID"));
        r.setReaction(resultset.getString("Reaction"));
        r.setTotalScore(resultset.getDouble("Score"));
        r.setCorrelation(0.0);
        ADRs.add(r);
    }
}
} catch (Exception e) {
    e.printStackTrace();
}

//close db connection
closeConnection();

    //fill the ADRList with the vector
    result.setADRs(ADRs);

    return result;
}

public Rule serveRuleRequest(RuleRequest rr) {
    Rule result = new Rule();
    Statement stmt = null;
    ResultSet resultset = null;

    // Load database driver
    openConnection();

    //define query
    String query = "SELECT * FROM RulesList WHERE RuleNum = "+rr.getRuleID()+"";
    try {
        //get query results
        if (debug) {System.out.println("Query: "+query);}  
        stmt = con.createStatement();
        resultset = stmt.executeQuery(query);

        //load each result into an ADRRecord, add it to the vector
        if (resultset.next()) {
            Vector<String> descriptions = new Vector<String>();
            //...
result setActive(resultset.getBoolean("Active"));
result setCode(resultset.getString("Code").replace("\r\n", "\r\r\n");
result setId(resultset.getString("RuleNum");
result setName(resultset.getString("RuleName");
for (String s : resultset.getString("RuleDetails").split("\r\n") { 
   descriptions.add(s);
}
result setTestDescriptions(descriptions); 
}
catch (Exception e) { 
   e.printStackTrace();
}

//close db connection 
closeConnection();
return result;
}

public RuleList serveRuleListRequest(RuleListRequest rlr) { 
   RuleList result = new RuleList();
   Vector<Rule> resultvec = new Vector<Rule>();
   Statement stmt = null;
   ResultSet resultset = null;

   // Load database driver 
   openConnection();

   //define query
String query = "SELECT * FROM RulesList";
try {
    //get query results
    if (debug) {System.out.println("Query: "+query);} 
    stmt = con.createStatement(); 
    resultset = stmt.executeQuery(query);
    //load each result into an ADRRecord, add it to the vector 
    while (resultset.next()) {
        Rule r = new Rule();
        Vector<String> descriptions = new Vector<String>();
        r setActive(resultset.getBoolean("Active"));
        r.setCode(resultset.getString("Code").replace("\r\n", "\\r\\n");
        r.setId(resultset.getString("RuleNum"));
        r.setName(resultset.getString("RuleName"));
        for (String s : resultset.getString("RuleDetails").split("\r\n")) {
            descriptions.add(s);
        }
        r.setTestDescriptions(descriptions);
        resultvec.add(r);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
    //close db connection
    closeConnection();
}
```java
result.setRules(resultvec);

return result;
}

public int getMaxPatientId() {
    int result = 0;
    Statement stmt = null;
    ResultSet resultset = null;

    // Load database driver
    openConnection();

    // Define query
    String query = "SELECT MAX(Patient_Id) as MaxId From Patient";

    try {
        // Get query results
        if (debug) {System.out.println("Query: "+query);}
        stmt = con.createStatement();
        resultset = stmt.executeQuery(query);

        // If results, grab the max
        if (resultset.next()) {
            result = resultset.getInt("MaxID");
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    // Close db connection
```
public Vector<String> getPatientsByDoctor(String doctorID) {
    Vector<String> result = new Vector<String>();
    Statement stmt = null;
    ResultSet resultset = null;
    // Load database driver
    openConnection();
    //define query
    String query = "SELECT Patient_Id FROM Patient WHERE doctor_id = "+doctorID+"";
    try {
        //get query results
        if (debug) {System.out.println("Query: "+query);}
        stmt = con.createStatement();
        resultset = stmt.executeQuery(query);

        //if results, grab the max
        while (resultset.next()) {
            result.add(resultset.getString("Patient_Id"));
        }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
// close db connection
closeConnection();
if (debug) {System.out.println("GetPatientByDoc("+doctorID+"): "+result.toString()); }
return result;
}

public int getRuleIdByName(String rulename) {
    int result = 0;
    Statement stmt = null;
    ResultSet resultset = null;
    // Load database driver
    openConnection();

    // define query
    String query = "SELECT RuleNum FROM RulesList WHERE RuleName = '" + rulename.replace("'","''" ) +"';"
    try {
        // get query results
        if (debug) {System.out.println("Query: "+query);}
        stmt = con.createStatement();
        resultset = stmt.executeQuery(query);

        // if results, grab the max
        if (resultset.next()) {
            result = resultset.getInt(0);
        }
    }
catch (Exception e) {
    e.printStackTrace();
}

// close db connection
closeConnection();

return result;
}

public boolean doctorExistsByUsername(String username) {
    boolean result = false;
    Statement stmt = null;
    ResultSet resultset = null;
    // Load database driver
    openConnection();

    // define query
    String query = "SELECT DocID From DoctorLogin WHERE Username = '"" + username.replace("'", "''") + "'");
    try {
        // get query results
        if (debug) {System.out.println("Query: "+query);}  
        stmt = con.createStatement();
        resultset = stmt.executeQuery(query);

        // if results, the doctor exists already
        if (resultset.next()) {
            result = true;
        } else {
            result = false;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return result;
}
catch (Exception e) {
    e.printStackTrace();
}

//close db connection
closeConnection();

return result;
}

public void serveNewUserRequest(NewUser nu) {
    Statement stmt = null;
    // Load database driver
    openConnection();

    //define query
    String query = "INSERT INTO DoctorLogin (Firstname, Lastname, Username, Password) VALUES ('" + nu.getFirstName().replace("'","''") + ",'" + nu.getLastName().replace("'","''") + ",'" + nu.getUsername().replace("'","''") + ",'" + MD5('"+nu.getPassword()+"isadoc'"))";
    try {
        //get query results
        if (debug) {System.out.println("Query: "+query);}
        stmt = con.createStatement();
        stmt.executeUpdate(query);
    }
    catch (Exception e) {

e.printStackTrace();

// close db connection
closeConnection();

// Ugly mess of like five database queries and crap
public FDASubmissionResponse serveFDASubmissionRequest(FDASubmissionRequest request) {
    FDASubmissionResponse response = new FDASubmissionResponse();
    Statement stmt = null;
    ResultSet resultset = null;

    response.setAdrID("Unknown");
    response.setPatientID("Unknown");
    response.setPatientDOB("Unknown");
    response.setPatientSex("Unknown");
    response.setPatientWeight("Unknown");
    response.setPatientDOD("Unknown");
    response.setDocName("Unknown");
    response.setDocAddress("Unknown");
    response.setDocEmail("Unknown");
    response.setDocPhone("Unknown");
    response.setDocOccupation("Unknown");
    response.setReactionDate("Unknown");
    response.setReportDate("Unknown");
    response.setReactionDesc("Unknown");
    response.setRelevantTests("Unknown");
response.setOtherHistory("Unknown");
response.setDrugName("Unknown");
response.setDrugStrength("Unknown");
response.setDrugManufact("Unknown");
response.setDrugDose("Unknown");
response.setDrugFreq("Unknown");
response.setDrugRoute("Unknown");
response.setDrugStartDate("Unknown");
response.setDrugEndDate("Unknown");
response.setDrugReasonForUse("Unknown");
response.setDrugDechallenge(false);
response.setDrugRechallenge(false);
response.setDrugLotNum("Unknown");
response.setDrugExpirationDate("Unknown");

// Load database driver
openConnection();

//define query
String query = "SELECT * FROM ADRs WHERE ADR_Id = " + request.getAdrID();
try {
    //get query results
    if (debug) {System.out.println("Query: "+query);} 
    stmt = con.createStatement();
    resultset = stmt.executeQuery(query);

    if(resultset.next())
{  
    response.setPatientID(resultset.getString("Patient_Id"));  
    response.setDrugName(resultset.getString("Drugname"));  
    response.setReactionDesc(resultset.getString("Reaction"));  
}

}  
  
catch (Exception e) {
    e.printStackTrace();
  }

  
  
closeConnection();  
  return response;

}

public void addADRRecord(ADRRecord r) {
  Statement stmt = null;
  ResultSet resultset = null;
  int recordid = 0;

  // Load database driver
  openConnection();

  // see if this record is already in the table
  String query = "SELECT ADR_Id FROM ADRs WHERE Patient_Id = "+r.getPatientID()  
  +" AND Drugname = "+r.getDrug().replace("", "'")  
  +" AND Reaction = "+r.getReaction().replace("", "'")+");
  try {
    // get query results
if (debug) {System.out.println("Query: "+query);}
stmt = con.createStatement();
resultset = stmt.executeQuery(query);

//if successful, load this adrreport's recordid
if (resultset.next()) {
    recordid = resultset.getInt("ADR_Id");
}
}
catch (Exception e) {
e.printStackTrace();
}

//define insert/update query, based on whether we found a matching record
if (recordid == 0) {
    //insert new record
    query = "INSERT INTO ADRs (Patient_Id, Drugname, Reaction, Score) VALUES ('"" +r.getPatientID()+"', '""+r.getDrug().replace("'", "''")+"', '"" +r.getReaction().replace("'", "''")+"', '""+r.getTotalScore()+"")";
} else {
    //just update the score
    query = "UPDATE ADRs SET Score = '""+r.getTotalScore()+"' WHERE ADR_Id = '""+recordid+"'");
}

try {
//get query results
if (debug) {System.out.println("QUERY: "+query);}
stmt = con.createStatement();
stmt.executeUpdate(query);
catch (Exception e) {
    e.printStackTrace();
}

//close db connection
closeConnection();

public Rule serveAddRuleRequest(AddRule ar) {
    Statement stmt = null;
    ResultSet resultset = null;
    boolean intable = false;
    String rulenum = "0";
    Rule result;

    // Load database driver
    openConnection();

    String query = "SELECT RuleNum FROM RulesList WHERE RuleNum='"+ar.getRuleID()+"'";
    //see if this record is already in the table
    if (ar.getRuleID().equals("0") == false) {
        try {
            //get query results
            if (debug) {System.out.println("Query: "+query);} 
            stmt = con.createStatement();
            resultset = stmt.executeQuery(query);

            //if successful, load this adrreport's recordid

if (resultset.next()) {
    intable=true;
}
}
catch (Exception e) {
    e.printStackTrace();
}

//build the descriptions string
String descriptions = "";
Vector<String> descVect = new Vector<String>(ar.getVectorTestDescriptions());
for (int i =0; i<descVect.size(); i++) {
    if (i != 0) {
        descriptions = descriptions+"\r\n";
    }
    descriptions = descriptions+descVect.elementAt(i);
}

descriptions = descriptions.replace("'","''").replace("\r\n","\r\n\n");
String code = ar.getCode().replace("'","''").replace("\r\n","\r\n\n");

//define insert/update query, based on whether we found a matching record
if (!intable) {
    //insert new record
    query = "INSERT INTO RulesList (Code, RuleName, RuleDetails, Active) VALUES ('" +code+"', '"+ar.getName().replace("'","''")+"', '"'+descriptions +"', "+ar.getActive()+")";
} else {
    //just update the score
    query = "UPDATE RulesList SET Code = '"+code +"', RuleName = '""
try {
    //get query results
    System.out.println("QUERY: "+query);
    stmt = con.createStatement();
    stmt.executeUpdate(query);
}

try {
    //get query results
    System.out.println("Query: "+query);
    stmt = con.createStatement();
    resultset = stmt.executeQuery(query);
    //if successful, load this adrreport's recordid
    if (resultset.next()) {
        rulenum = resultset.getString(0);
    }
}
catch (Exception e) {
}
e.printStackTrace();
}

//close db connection
closeConnection();

RuleRequest rr = new RuleRequest();
rr.setRuleID(rulenum);
return serveRuleRequest(rr);
}

private void openConnection() {
    try {
        try {
            Class.forName(dbDriver);
        }
        catch (Exception e) {
            System.out.println("Failed to open JDBC driver" + e);
        }
        con = DriverManager.getConnection(dbURL, dbLogin, dbPass);
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}

private void closeConnection() {
}
try {
    con.close();
} catch (Exception e) {
    e.printStackTrace();
}
Program: HospitalOntology.java
Description: Defines the actions, objects, and relationships that WSIG needs in order to create a properly formatted SOAP Document Template Description (DTD), and validate incoming and outgoing SOAP requests.
Programmer: Andrew Rueckert

package edu.gonzaga.cpen.hospitalagents;

import jade.content.onto.BasicOntology;
import jade.content.onto.Ontology;
import jade.content.onto.OntologyException;
import jade.content.schema.AgentActionSchema;
import jade.content.schema.ConceptSchema;
import jade.content.schema.TermSchema;
import jade.content.schema.PrimitiveSchema;
import jade.content.schema.ObjectSchema;

public class HospitalOntology extends Ontology {
    //Define instance to return upon request
    private final static Ontology theInstance = new HospitalOntology();

    public final static Ontology getInstance() {
        return theInstance;
    }

    private HospitalOntology() {
        //call Ontology's constructor
        super("hospital-ontology", BasicOntology.getInstance());
    }
}
//define variables to use later
AgentActionSchema as;
ConceptSchema cs;

try {
    //Default (necessary) schemas
    //concepts are things, which can contain other concepts as well as primitives
    add(new ConceptSchema("agentInfo"), AgentInfo.class);
    //actions are things that can be requested. They have input parameters that
    //can be primitives or concepts, and results which can be primitives or
    //concepts
    add(new AgentActionSchema("GetAgentInfo"), GetAgentInfo.class);

    //define a concept, as well as it's composition
    cs = (ConceptSchema) getSchema("agentInfo");
    cs.add("agentAid", (TermSchema) getSchema(BasicOntology.AID));
    cs.add("startDate", (PrimitiveSchema) getSchema(BasicOntology.DATE));

    //define an action, as well as it's return value
    as = (AgentActionSchema) getSchema("GetAgentInfo");
    as.setResult((ConceptSchema) getSchema("AgentInfo"));

    //User-defined schemas
    add(new AgentActionSchema("ADRDetailRequest"), ADRDetailRequest.class);
    add(new ConceptSchema("ADRList"), ADRList.class);
    add(new AgentActionSchema("ADRListRequest"), ADRListRequest.class);
    add(new ConceptSchema("ADRRecord"), ADRRecord.class);
    add(new AgentActionSchema("AddRule"), AddRule.class);
add(new ConceptSchema("Doctor"), Doctor.class);
add(new AgentActionSchema("FDASubmissionRequest"), FDASubmissionRequest.class);
add(new ConceptSchema("FDASubmissionResponse"), FDASubmissionResponse.class);
add(new AgentActionSchema("Login"), Login.class);
add(new AgentActionSchema("NewUser"), NewUser.class);
add(new ConceptSchema("Patient"), Patient.class);
add(new ConceptSchema("Reaction"), Reaction.class);
add(new ConceptSchema("Rule"), Rule.class);
add(new ConceptSchema("RuleList"), RuleList.class);
add(new AgentActionSchema("RuleListRequest"), RuleListRequest.class);
add(new AgentActionSchema("RuleRequest"), RuleRequest.class);
add(new ConceptSchema("RuleResult"), RuleResult.class);

as = (AgentActionSchema) getSchema("ADRDetailRequest");
as.add("adrID", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.setResult((ConceptSchema) getSchema("ADRRecord"));

cs = (ConceptSchema) getSchema("ADRList");
cs.add("ADRs", (ConceptSchema) getSchema("ADRRecord"), 0, ObjectSchema.UNLIMITED);

as = (AgentActionSchema) getSchema("ADRListRequest");
as.add("doctorID", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.setResult((ConceptSchema) getSchema("ADRList"));

cs = (ConceptSchema) getSchema("ADRRecord");
cs.add("patientID", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("adrID", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("drug", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("reaction", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("totalScore", (PrimitiveSchema) getSchema(BasicOntology.FLOAT));
cs.add("correlation", (PrimitiveSchema) getSchema(BasicOntology.FLOAT));
cs.add("scoreList",
    (ConceptSchema) getSchema("RuleResult"), 0, ObjectSchema.UNLIMITED);

as = (AgentActionSchema) getSchema("AddRule");
as.add("name", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.add("code", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.add("active", (PrimitiveSchema) getSchema(BasicOntology.BOOLEAN));
as.add("ruleID", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.add("testDescriptions",
    (PrimitiveSchema) getSchema(BasicOntology.STRING), 0, ObjectSchema.UNLIMITED);
as.setResult((ConceptSchema) getSchema("Rule"));

cs = (ConceptSchema) getSchema("Doctor");
cs.add("username", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("firstName", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("lastName", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("docID", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("docAddress",
    (PrimitiveSchema) getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);
cs.add("docEmail",
    (PrimitiveSchema) getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);
cs.add("docPhone",
    (PrimitiveSchema) getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);
cs.add("docOccupation",
    (PrimitiveSchema) getSchema(BasicOntology.STRING), ObjectSchema.OPTIONAL);
cs.add("failure", (PrimitiveSchema) getSchema(BasicOntology.BOOLEAN));
as = (AgentActionSchema) getSchema("FDASubmissionRequest");
as.setResult((ConceptSchema) getSchema("FDASubmissionResponse"));
cs.add("drugEndDate", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("drugReasonForUse", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("drugDechallenge", (PrimitiveSchema) getSchema(BasicOntology.BOOLEAN));
cs.add("drugRechallenge", (PrimitiveSchema) getSchema(BasicOntology.BOOLEAN));
cs.add("drugLotNum", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("drugExpirationDate", (PrimitiveSchema) getSchema(BasicOntology.STRING));

as = (AgentActionSchema) getSchema("Login");
as.add("username", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.add("password", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.setResult((ConceptSchema) getSchema("Doctor"));

as = (AgentActionSchema) getSchema("NewUser");
as.add("username", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.add("firstName", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.add("lastName", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.add("password", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.add("docAddress", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.add("docEmail", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.add("docPhone", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.add("docOccupation", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.add("failure", (PrimitiveSchema) getSchema(BasicOntology.BOOLEAN));
as.setResult((ConceptSchema) getSchema("Doctor"));

cs = (ConceptSchema) getSchema("Patient");
cs.add("patientID", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("patientDOB", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("patientSex", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("patientWeight", (PrimitiveSchema) getSchema(BasicOntology.STRING));
// HospitalOntology.java

package edu.gonzaga.cpen.hospitalagents;

import org.apache.jena.rdf.model.ModelMaker;

public class HospitalOntology {

    private static final ModelMaker modelMaker = ModelMaker.getSingleton();

    private static final String HOSPITAL_ONTOLGY_NAMESPACE = "http://example.com/hospital ontology#";

    private static final String PURCHASED_ONTOLGY_NAMESPACE = "http://example.com/purchased ontology#";

    private static final String PATIENT_ONTOLGY_NAMESPACE = "http://example.com/patient ontology#";

    private static final String DRUG_ONTOLGY_NAMESPACE = "http://example.com/drug ontology#";

    private static final String RULE_ONTOLGY_NAMESPACE = "http://example.com/rule ontology#";

    public static void main(String[] args) {
        // Create a new model
        Model model = modelMaker.createModel();

        // Add a new patient
        Person patient = new Person();
        patient.setName("John Doe");
        patient.setAge(30);
        patient.setGender("M");
        model.add(patient);"
cs = (ConceptSchema) getSchema("RuleList");
cs.add("rules", (ConceptSchema) getSchema("Rule"), 0, ObjectSchema.UNLIMITED);

as = (AgentActionSchema) getSchema("RuleListRequest");
as.setResult((ConceptSchema) getSchema("RuleList"));

as = (AgentActionSchema) getSchema("RuleRequest");
as.add("ruleID", (PrimitiveSchema) getSchema(BasicOntology.STRING));
as.setResult((ConceptSchema) getSchema("Rule"));

cs = (ConceptSchema) getSchema("RuleResult");
cs.add("rulename", (PrimitiveSchema) getSchema(BasicOntology.STRING));
cs.add("score", (PrimitiveSchema) getSchema(BasicOntology.FLOAT));

} catch (OntologyException oe) {
    oe.printStackTrace();
}
catch (Exception e) {
    e.printStackTrace();
}
package edu.gonzaga.cpen.hospitalagents;

import jade.util.leap.ArrayList;
import jade.util.leap.List;
import javax.xml.soap.*;
import java.util.Iterator;
import jade.content.Concept;
import java.util.Vector;
import jade.content.AgentAction;

public class ADRRecord implements SOAPSerializable, Concept, Cloneable {
    private String patientID;
    private String adrID;
    private String drug;
    private String reaction;
    private double totalScore;
    private double correlation;
    private ArrayList scoreList; // <RuleResult>

    // constructor
    public ADRRecord() {
        scoreList = new ArrayList();
    }
}
//generic setter/getter functions for primitives
//Concepts must have full setter/getter sets
public void setPatientID(String s) { patientID = s; }
public String getPatientID() { return patientID; }
public void setAdrID(String s) { adrID = s; }
public String getAdrID() { return adrID; }
public void setDrug(String s) { drug = s; }
public String getDrug() { return drug; }
public void setReaction(String s) { reaction = s; }
public String getReaction() { return reaction; }
public void setTotalScore(double d) { totalScore = d; }
public double getTotalScore() { return totalScore; }
public void setCorrelation(double d) { correlation = d; }
public double getCorrelation() { return correlation; }

//setter for a List. setScoreList is overloaded to handing JADE lists
public void setScoreList(jade.util.leap.List l) {
    scoreList.clear();
    for (int i=0; i<l.size(); i++) {
        scoreList.add(l.get(i));
    }
}

//... OR Java Core lists
public void setScoreList(java.util.List l) {
    scoreList.fromList(l);
}
// getter function for a list. You can get a JADE list
public jade.util.leap.List getScoreList() { return scoreList; }

// ... OR a Java Core list
public java.util.List getVectorScoreList() { return scoreList.toList(); }

// This function takes an XML string and fills in the Concept variables
// from it. It recursively calls other Concepts unmarshal functions,
// if necessary
public void unmarshal(String xml) {
    XMLParser xp = new XMLParser();
    String temp;
    Vector<String> tempvect = new Vector<String>();

    patientID = xp.getTag(xml, "patientID");
    adrID = xp.getTag(xml, "adrID");
    drug = xp.getTag(xml, "drug");
    reaction = xp.getTag(xml, "reaction");
    totalScore = Double.parseDouble(xp.getTag(xml, "totalScore"));
    correlation = Double.parseDouble(xp.getTag(xml, "correlation"));
    scoreList.clear();
    temp = xp.getTag(xml, "scoreList");
    tempvect = xp.getList(temp, "RuleResult");
for( String s : tempvect) {
    RuleResult r = new RuleResult();
    r.unmarshal(s);
    scoreList.add(r);
}

//This function takes the concepts current contents, and builds a SOAPElement
//based off of it. It then attaches the element as a child to the parent, passed
//as a parameter
public void marshal(SOAPElement parent) throws javax.xml.soap.SOAPException {
    SOAPElement ADRRecordEle = parent.addChildElement("ADRRecord");

    SOAPElement patientIDEle = ADRRecordEle.addChildElement("patientID");
    patientIDEle.setAttribute("xsi:type", "xsd:string");
    patientIDEle.addTextNode(patientID);

    SOAPElement adrIDEle = ADRRecordEle.addChildElement("adrID");
    adrIDEle.setAttribute("xsi:type", "xsd:string");
    adrIDEle.addTextNode(adrID);

    SOAPElement drugEle = ADRRecordEle.addChildElement("drug");
    drugEle.setAttribute("xsi:type", "xsd:string");
    drugEle.addTextNode(drug);

    SOAPElement reactionEle = ADRRecordEle.addChildElement("reaction");
    reactionEle.setAttribute("xsi:type", "xsd:string");
    reactionEle.addTextNode(reaction);
SOAPElement totalScoreEle = ADRRecordEle.addChildElement("totalScore");
totalScoreEle.setAttribute("xsi:type", "xsd:string");
totalScoreEle.addTextNode(Double.toString(totalScore));

SOAPElement correlationEle = ADRRecordEle.addChildElement("correlation");
correlationEle.setAttribute("xsi:type", "xsd:string");
correlationEle.addTextNode(Double.toString(correlation));

SOAPElement scoreListEle = ADRRecordEle.addChildElement("scoreList");
for (int i=0; i<scoreList.size(); i++) {
    RuleResult element = (RuleResult)scoreList.get(i);
    element.marshal(scoreListEle);
}

//This function is very similar to marshal(), but is intended for when the
//ADRRecord is a child of the SOAP Messages's body, rather than a child element
//of another SOAP Element
public void bodyMarshal(SOAPBody body, SOAPEnvelope env) throws javax.xml.soap.SOAPException {
    SOAPElement ADRRecordEle = body.addBodyElement(env.createName("urn:ADRRecord"));

    SOAPElement patientIDEle = ADRRecordEle.addChildElement("patientID");
    patientIDEle.setAttribute("xsi:type", "xsd:string");
    patientIDEle.addTextNode(patientID);

    SOAPElement adrIDEle = ADRRecordEle.addChildElement("adrID");
    adrIDEle.setAttribute("xsi:type", "xsd:string");
adrIDEle.addTextNode(adrID);

SOAPElement drugEle = ADRRecordEle.addChildElement("drug");
drugEle.setAttribute("xsi:type", "xsd:string");
drugEle.addTextNode(drug);

SOAPElement reactionEle = ADRRecordEle.addChildElement("reaction");
reactionEle.setAttribute("xsi:type", "xsd:string");
reactionEle.addTextNode(reaction);

SOAPElement totalScoreEle = ADRRecordEle.addChildElement("totalScore");
totalScoreEle.setAttribute("xsi:type", "xsd:string");
totalScoreEle.addTextNode(Double.toString(totalScore));

SOAPElement correlationEle = ADRRecordEle.addChildElement("correlation");
correlationEle.setAttribute("xsi:type", "xsd:string");
correlationEle.addTextNode(Double.toString(correlation));

SOAPElement scoreListEle = ADRRecordEle.addChildElement("scoreList");
for (int i=0; i<scoreList.size(); i++) {
    RuleResult element = (RuleResult)scoreList.get(i);
    element.marshal(scoreListEle);
}

//This is a generic clone function. It creates a copy of this object,
//whose member variables are copies of this object's member variables
//(recursively, if necessary)
@Override
protected ADRRecord clone() {
    ADRRecord retval = new ADRRecord();
    retval.patientID = patientID;
    retval.adrID = adrID;
    retval.drug = drug;
    retval.reaction = reaction;
    retval.totalScore = totalScore;
    retval.correlation = correlation;
    retval.scoreList.fromList(scoreList.toList());
    return retval;
}
/*
 * Program: XMLParser.java
 * Description: Can do very basic XML parsing. Determines presence of well-formed
 * tags, returns contents of first well-formed tag, or returns a list of the
 * contents of all well-formed tags of a certain type.
 * Programmer: Andrew Rueckert
 */

package edu.gonzaga.cpen.hospitalagents;
import java.util.TreeSet;
import java.util.Vector;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

/**
 * @author srueckert
 */

public class XMLParser {
    //Used for debugging output
    public boolean debug;
    public void XMLParser() {
        debug = true;
    }

    //returns TRUE if it finds a start/end pair of tags with tagname inbetween
    //start and end chars.
    public boolean findTag(String input, String tagname, int start, int end) {
        boolean foundstart=false, foundend=false;
        }
// regular-expression pattern to find the opening tag. Must start with '<',
// optionally have 'urn:' before the tagname, and may not contain any '/'
// before the closing '>
Pattern openpattern = Pattern.compile("<(urn:)?"+tagname+"[^/]*?>");
Matcher openmatcher = openpattern.matcher(input);
while (openmatcher.find()) {
    foundstart=true;
}

// regular expression to find the closing tag
// must start with '</', followed by the tagname, and may not contain
// any '<' or '>' characters before the closing '>
Pattern closepattern = Pattern.compile("</(urn:)?"+tagname+"[^<>]*?>");
Matcher closematcher = closepattern.matcher(input);
while (closematcher.find() && foundstart) {
    foundend=true;
}
return (foundstart&&foundend);

// get tag returns the contents of the first well-formed tag of "tagname" that
// it finds.
public String getTag(String input, String tagname) {
    return getTag(input, tagname, 0, input.length());
}

public String getTag(String input, String tagname, int startpos) {
    return getTag(input, tagname, startpos, input.length());
}

public String getTag(String input, String tagname, int start, int end) {
    // treesets are ordered lists
    TreeSet<Tag> locations = new TreeSet<Tag>();
//build an ordered list of opening and closing tags
Pattern openpattern = Pattern.compile("<\(urn:\)?\+tagname\="[\>]\+\">\+\"\+\";");
Matcher openmatcher = openpattern.matcher(input);
while (openmatcher.find()) {
    int s = openmatcher.start();
    int e = openmatcher.end();
    if (s < start) {
        continue;
    }
    if (e > end) {
        break;
    }
    Tag t = new Tag(s, e, true);
    if (debug) {System.out.println("Open :" + t.toString());}
    locations.add(t);
}

Pattern closepattern = Pattern.compile("</\(urn:\)?\+tagname\="[\>]\+\">\+\"\+\";");
Matcher closematcher = closepattern.matcher(input);
while (closematcher.find()) {
    int s = closematcher.start();
    int e = closematcher.end();
    if (s < start) {
        continue;
    }
    if (e > end) {
        break;
    }
    Tag t = new Tag(s, e, false);
    locations.add(t);
}
if (debug) {System.out.println("Close :"+t.toString());}
locations.add(t);
}

// finds any "empty" tags. Empty tags look like <tagname/> this, and do
// not have a "closing" tag. They count as both an opening and closing tag,
// and will not have any contents.
Pattern emptypattern = Pattern.compile("<(urn:)?"+tagname+"[^<>]*?/>");
Matcher emptymatcher = emptypattern.matcher(input);
while (emptymatcher.find()) {
    int s = emptymatcher.start();
    int e = emptymatcher.end();
    if (s < start) {
        continue;
    }
    if (e > end) {
        break;
    }
    Tag t = new Tag(s, e, true);
    if (debug) {System.out.println("Empty :"+t.toString());}
    locations.add(t);
    locations.add(new Tag(e, e, false));
}
if (debug) {System.out.println("TagList: "+locations.toString());}

int depth = 0;
int startloc = 0;
int curloc = 0;

// start at 0 depth. You will immediately hit an "open" tag, bumping you
for (Tag t : locations) {
    if (t.open) {
        depth++;
        if (startloc == 0) {
            startloc = t.end;
        }
    } else {
        depth--;
        curloc = t.start;
    }
    if (depth==0) {
        break;
    }
}

    return input.substring(startloc, curloc);
}

public Vector<String> getList(String input, String tag) {
    return getList(input, tag, 0, input.length());
}

public Vector<String> getList(String input, String tag, int start) {
    return getList(input, tag, start, input.length());
}

public Vector<String> getList(String input, String tagname, int start, int end) {
    TreeSet<Tag> locations = new TreeSet<Tag>();
    Vector<String> retlist = new Vector<String>();
//build an ordered list of opening and closing tags
Pattern openpattern = Pattern.compile("<(urn:)?"+tagname+"[^/<]*?>");
Matcher openmatcher = openpattern.matcher(input);
while (openmatcher.find()) {
    int s = openmatcher.start();
    int e = openmatcher.end();
    if (s < start) {
        continue;
    }
    if (e > end) {
        break;
    }
    Tag t = new Tag(s, e, true);
    if (debug) {System.out.println("Open :"+t.toString());}
    locations.add(t);
}
Pattern closepattern = Pattern.compile("</(urn:)?"+tagname+"[^<>]*?>");
Matcher closematcher = closepattern.matcher(input);
while (closematcher.find()) {
    int s = closematcher.start();
    int e = closematcher.end();
    if (s < start) {
        continue;
    }
    if (e > end) {
        break;
    }
    Tag t = new Tag(s, e, false);
if (debug) {System.out.println("Close :" + t.toString());}
locations.add(t);
Pattern emptypattern = Pattern.compile("<(urn:)?" + tagname + "[^<>]*?/>");
Matcher emptymatcher = emptypattern.matcher(input);
while (emptymatcher.find()) {
    int s = emptymatcher.start();
    int e = emptymatcher.end();
    if (s < start) {
        continue;
    }
    if (e > end) {
        break;
    }
    Tag t = new Tag(s, e, true);
    if (debug) {System.out.println("Empty :" + t.toString());}
    locations.add(t);
    locations.add(new Tag(e, e, false));
}
if (debug) {System.out.println("Taglist: " + locations.toString());}
int depth = 0;
int startloc = 0;
int curloc = 0;
//same as before, except keep a list of all top-level open/close
for (Tag t : locations) {
    if (debug) {System.out.println("Taglist: " + locations.toString());}
    if (t.open) {
        depth++;
    }
    if (startloc == 0) {
        int startloc = t.start();
    }
    if (t.open) {
        int startloc = t.start();
    }
    if (curloc == t.start()) {
        depth--;
    }
}
//Tags have a start location at the '<' character, an end location at the
// '>' character, and are either opening tags (<tag>) or closing tags
// </tag>)
//They can be compared by their opening position, so they can be sorted
//(like in a treelist)
class Tag implements Comparable {
    public int start;
    public int end;
    public boolean open;

    public Tag(int start, int end, boolean open) {
        this.start = start;
        this.end = end;
        this.open = open;
    }
}
public String toString() {
    return "{start: "+start+", end: "+end+", open: "+(open)?"open}:"close}"};
}

public int compareTo(Object o) {
    if (o instanceof Tag) {
        Tag otag = (Tag)o;
        if (start < otag.start) {
            return -1;
        } else if (start > otag.start) {
            return 1;
        } else {
            return 0;
        }
    } else {
        return 0;
    }
}
For VOLUNTARY reporting of adverse events and product problems

A. Patient information
1. Patient identifier
2. Age at time of event:
   or Date of birth:

3. Sex
   □ female
   □ male

4. Weight
   lbs
   kgs

B. Adverse event or product problem
1. ☐ Adverse event
   or ☐ Product problem (e.g., defects/malfunctions)

2. Outcomes attributed to adverse event
   (check all that apply)
   □ death
   □ life-threatening
   □ hospitalization - initial or prolonged

3. Date of event
   (mo/day/yr)

4. Date of this report
   (mo/day/yr)

5. Describe event or problem

C. Suspect medication(s)
1. Name (give labeled strength & mfr/lbl, if known)
   #1
   #2

2. Dose, frequency & route used
   #1
   #2

3. Therapy dates (if unknown, give duration)
   #1
   #2

4. Diagnosis for use (indication)
   #1
   #2

5. Event abated after use stopped or dose reduced
   #1
   #2

6. Lot # (if known)
   #1
   #2

7. Exp. date (if known)
   #1
   #2

8. Event reappeared after reintroduction
   #1
   #2

9. NDC # (for product problems only)
   -
   -

10. Concomitant medical products and therapy dates (exclude treatment of event)

D. Suspect medical device
1. Brand name

2. Type of device

3. Manufacturer name & address

4. Operator of device
   □ health professional
   □ lay user/patient
   □ other:

5. Expiration date
   (mo/day/yr)

6. model #

7. catalog #

8. serial #

9. lot #

10. Other relevant history, including preexisting medical conditions (e.g., allergies, race, pregnancy, smoking and alcohol use, hepatic/renal dysfunction, etc.)

E. Reporter (see confidentiality section on back)
1. Name & address
2. Phone #

3. Occupation

4. Also reported to
   □ manufacturer
   □ user facility
   □ distributor

5. If you do NOT want your identity disclosed to the manufacturer, place an “X” in this box.

Submission of a report does not constitute an admission that medical personnel or the product caused or contributed to the event.
ADVICE ABOUT VOLUNTARY REPORTING

Report adverse experiences with:
• medications (drugs or biologics)
• medical devices (including in-vitro diagnostics)
• special nutritional products (dietary supplements, medical foods, infant formulas)
• cosmetics
• medication errors

Report product problems – quality, performance or safety concerns such as:
• suspected contamination
• questionable stability
• defective components
• poor packaging or labeling
• therapeutic failures

Report SERIOUS adverse events. An event is serious when the patient outcome is:
• death
• life-threatening (real risk of dying)
• hospitalization (initial or prolonged)
• disability (significant, persistent or permanent)
• congenital anomaly
• required intervention to prevent permanent impairment or damage

Report even if:
• you’re not certain the product caused the event
• you don’t have all the details

How to report:
• just fill in the sections that apply to your report
• use section C for all products except medical devices
• attach additional blank pages if needed
• use a separate form for each patient
• report either to FDA or the manufacturer (or both)

Confidentiality: The patient’s identity is held in strict confidence by FDA and protected to the fullest extent of the law. FDA will not disclose the reporter’s identity in response to a request from the public, pursuant to the Freedom of Information Act. The reporter’s identity, including the identity of a self-reporter, may be shared with the manufacturer unless requested otherwise.

If your report involves a serious adverse event with a device and it occurred in a facility outside a doctor’s office, that facility may be legally required to report to FDA and/or the manufacturer. Please notify the person in that facility who would handle such reporting.

Important numbers:
• 1-800-FDA-0178 to FAX report
• 1-800-FDA-1088 to report by phone or for more information
• 1-800-822-7967 for a VAERS form for vaccines

To Report via the Internet:
https://www.accessdata.fda.gov/scripts/medwatch/

Confidentiality:
The patient’s identity is held in strict confidence by FDA and protected to the fullest extent of the law. FDA will not disclose the reporter’s identity in response to a request from the public, pursuant to the Freedom of Information Act. The reporter’s identity, including the identity of a self-reporter, may be shared with the manufacturer unless requested otherwise.

The public reporting burden for this collection of information has been estimated to average 30 minutes per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information.

Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to:

DHHS Reports Clearance Office
Paperwork Reduction Project (0910-0291)
Hubert H. Humphrey Building, Room 531-H
200 Independence Avenue, S.W.
Washington, DC 20201

Please DO NOT RETURN this form to this address.

U.S. DEPARTMENT OF HEALTH AND HUMAN SERVICES
Public Health Service • Food and Drug Administration

The public reporting burden for this collection of information has been estimated to average 30 minutes per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information.

Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to:

DHHS Reports Clearance Office
Paperwork Reduction Project (0910-0291)
Hubert H. Humphrey Building, Room 531-H
200 Independence Avenue, S.W.
Washington, DC 20201

Please DO NOT RETURN this form to this address.

U.S. DEPARTMENT OF HEALTH AND HUMAN SERVICES
Public Health Service • Food and Drug Administration

The public reporting burden for this collection of information has been estimated to average 30 minutes per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information.

Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to:

DHHS Reports Clearance Office
Paperwork Reduction Project (0910-0291)
Hubert H. Humphrey Building, Room 531-H
200 Independence Avenue, S.W.
Washington, DC 20201

Please DO NOT RETURN this form to this address.

U.S. DEPARTMENT OF HEALTH AND HUMAN SERVICES
Public Health Service • Food and Drug Administration