ASEAM

Developer's Manual

Andreas Bauer, baueran@cs.anu.edu.au

The Australian National University
Department of Computer Science
Canberra, ACT 0200

April 17, 2000

Contents

List of Figures

1 Introduction

ASEAM stands for *ANU Student Enrollment And Marks System* used within the Faculty of Engineering and Information Technology. This manual describes the internal structure of ASEAM, the employed programming languages and modules (API). It also outlines its security model and the possibility of future extensions to the system.

2 Application Structure

2.1 Java Servlets

The ASEAM business logic is realized with server-side Java applications called Servlets. Servlets are an established technique for implementing web based applications which are portable and highly flexible due to the modern concepts of the Java programming language [HC98].

Generally the clients need merely a web browser in order to access such applications. Java or JavaScript does not have to be supported (nor any other plug-ins). All incoming client requests (the common HTTP commands `GET`, `PUT` or `POST`) are being translated by the Servlet and get forwarded to its subsystems (see also the ASEAM Overview and Specification). Servlets usually provide methods for each possible HTTP command that comes in. Therfore, most commonly the methods `doPut()`, `doGet()`, etc. have to be overriden. They are responsible for the actual translation and for passing on the client information such that other methods can process the request. A dynamically generated HTTP-response is then sent back to the client which usually contains the resulting web page.

Unlike common CGI applications, Servlets can be considered as 'processes' which initialize the first time an incoming request is received and unload after a certain time out or a manual shut down. That is, Servlets have to be 'threadsafe' to be able to deal with many users accessing simultaniously[1]. Java provides an advanced implementation of thread-handling methods which make it easy to control multiple client requests.

2.2 Session Tracking

There are two common ways to track information about clients who access a Servlet: session tracking via URL-rewriting or via Cookies. Since Apache does not support URL-rewriting by default (and not without a lot more effort in configuring and implementing it), ASEAM uses choice No. 2: the Cookies. Such a Cookie file contains not only a serial number (session ID) to distinguish the user, but also the user's name, the status within the Department of Computer Science (in general an integer value indicating 'staff' or 'student') and an `HtmlBrowser` object which can be considered as the user's individual display object. Internally the Servlet stores this information in a `UserSessionProfile` object.

A unique `UserSessionProfile` object (and therefore Cookie file) is created as soon as a new user tries to establish contact to the ASEAM system. It is destroyed either by time out or once the user exits the system manually.

[1]As a matter of fact, a Servlet does not necessarily have to be multi-threaded. But then one of the big advantages of using Java would be lost. It is possible to only handle each user at the time and queue all other users until this specific process has finished - on the expense of system performance.

Everytime the Servlet invokes one of its methods it checks the current session state and then processes the user's request.

2.3 Database Connectivity

ASEAM uses the PostgreSQL database engine as persistent data storage subsystem. Its interface to the database itself is the Java Database Bridge Driver (JDBC) provided by the PostgreSQL distribution. The driver itself is a thread-safe Java archive which makes it impossible for two clients to share a single connection at the same time. Servlets may use this advantage and employ only one connection for all interacting users. With such an approach the overall performance would suffer tremendously as only one client request could be processed at a time. However, this approach theoretically works.

More efficient and elegant is the creation of an object pool which contains objects the methods can borrow and return. ASEAM uses the `DbConnectionPool` class (which extends a generic `ObjectPool` class) to handle multiple database connections simultaneously [Dav98].

Algorithm 1 Concept of the ASEAM object pool

```
public abstract class ObjectPool {
  private long expiration Time;
  private Hashtable locked;
  private Hashtable unlocked;

  abstract Object create();
  abstract boolean validate(Object currentObject);
  abstract void expire(Object currentObject);
  synchronized Object checkOut() {
    ...
  }
  synchronized void checkIn(Object currentObject) {
    ...
  }
}

public class DbConnectionPool extends ObjectPool {
  ...
}
```

Whenever a client evokes a database query, the according method borrows a free connection, performs the query, receives the result and finally closes and returns the connection back to the pool. That way resources are only held as shortly as possible.

2.4 Dynamic Elements

User interaction is based upon dynamically generated web pages. In general the Servlet builds these, depending on the state and status of the corresponding user. Their content is sent to the user's web browser in an `HttpServletResponse` object. Such a response may contain database results or static web pages loaded from the local filesystem.

The figure on page 7 illustrates user interaction as well as the application scheme of ASEAM as a whole.

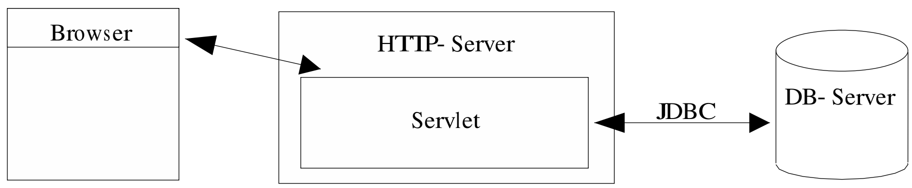

Figure 1: Application Scheme

2.5 Error Handling

The ASEAM system distinguishes between several kinds of errors but handles them as homogeneously to the user as possible by sending detailed error messages to the web browser. In general the following kinds of errors are handled:

- General exception errors
- SQL exception errors in particular
- User input errors (such as sending wrong parameters to the Servlet, etc.)

The Java Virtual Machine itself usually takes care of the general exception handling (as it would happen for example if you divide something by zero) by passing errors on to other methods or catching and processing them. This provides stability and flexibility to the ASEAM system in case of unexpected failure.

ASEAM handles SQL exceptions particularly different though. Those errors usually occur while the Servlet is communicating with the underlying database architecture. For example if a whole lot of users tried to establish a database connection at the same time it might occur that the system runs out of resources and causes the database driver to throw an SQL exception. This case, and all other similar situations, are handled within the corresponding Servlet methods themselves: the methods construct an error message containing the description of the exception and send this information back to the browser's display window. That is, SQL exceptions are not passed on or processed within the Servlet in a way that the user is not able to see what went wrong with his or her request.

User input errors on the other hand, are usually no *real* errors. Whenever a user tries to evoke a method which is temporarily unavailable to this particular user, the ASEAM system creates a message to explain the situation. For example, if a student tried to unenroll from a last year's unit the system will respond with an error message to prevent this operation. To control which operations are valid, ASEAM uses two central methods: `doPut()` and `doPost()`. Every time a user evokes a Servlet function either one of the two methods gets called first. They act as a kind of 'central filter' keeping track of what is allowed and what is not. Most future aspects of access restriction can therefore be implemented in these two methods without putting additional code in any other methods at all.

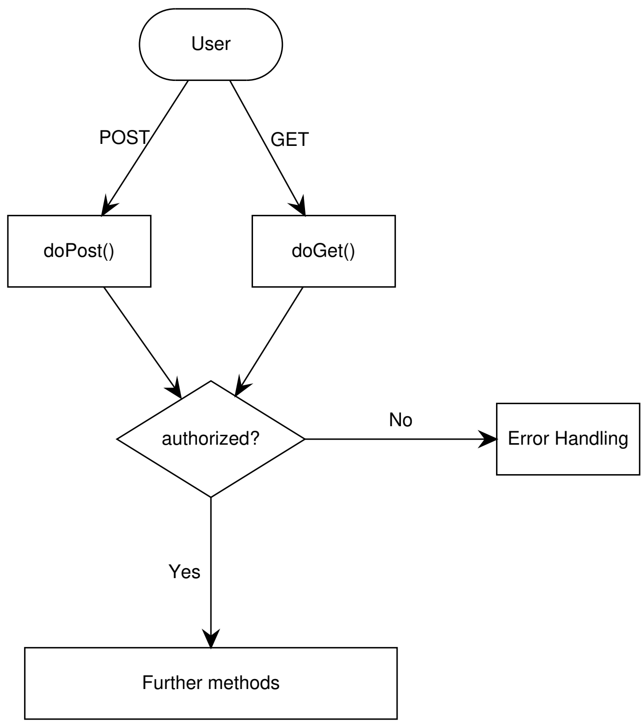

Figure 2: Access restriction scheme within the ASEAM class files

Algorithm 2 Access control realized in the `doGet()` method

```
public void doGet (HttpServletRequest request,
                   HttpServletResponse response)
                   throws ServletException, IOException {

   // Get the user's current profile as a session object
   HttpSession session = request.getSession(true);
   UserSessionProfile userSession =
       (UserSessionProfile)session.getValue(session.getId());

   // Set output properties
   response.setContentType("text/html");
   HtmlBrowser localBrowser =
      new HtmlBrowser(response.getWriter());

   // If a Session is currently running and active, grant
   // access. Otherwise return error message: user has to
   // authenticate first.
   //
   if ( !session.isNew() && userSession != null ) {
      userSession.setBrowser(localBrowser);

      if ( (templateFileName =
         request.getParameter("template")) != null ) {

         // Check whether the user requested a template HTML
         // file. Check which template files have to be
         // loaded and if the user is authenticated to
         // perform operation
         //
         if ( templateFileName.compareTo("students") == 0 &&
              userSession.status() == STUDENT )

              // Show main view for students
              showStudentsMainView(request, response,
                                   session, userSession);
      ...
}
```

3 Security

To protect ASEAM from unrestricted user access as well as from various spying attacks a comprehensive security model is implemented. The main tasks of the model are:

- user authentication
- data encryption
- access control/restriction

User authentication depends on a valid username/password combination. The idea was to tie this to the campus NIS+ database [NIS00], but for reasons discussed in the project overview [ANU01], ASEAM uses merely the standard mod_auth implementation of the Apache web server. mod_auth accesses an encrypted password file on the local filesystem to compare the user's input values. By default only the `docs/` and the `pics/` directories as well as the main `index.html` file are accessible for unauthorized users. All further communication, including Servlet requests, demand authorization.

This authorization operation is encrypted to prevent others from tracking user passwords. ASEAM uses the HTTPS protocol to send only encrypted information over the network. That is, HTTP over SSL (an additional transport protocol on top of TCP/IP providing communication authentication, privacy and integrity through strong cryptography techniques) and thus a secure HTTP variant. SSL is short for "Secure Socket Layer" [E00]. Similar to ASEAM's

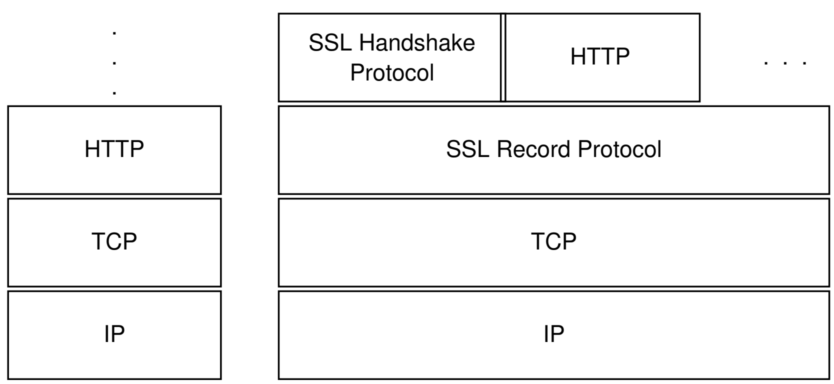

Figure 3: Transport protocols without SSL *(left)*, SSL protocol stack *(right)*

technique of handling runtime errors, it restricts access also to parts of the web server itself, such as directories and the Servlet engine itself. A specific IP number and a specific username is necessary to execute the Servlets in this environment. That way it is not possible to upload and execute unauthorized code. The fact that `inetd` is entirely switched off makes it quite difficult anyway.

4 Database Model

ASEAM's persistent data storage layer is a PostgreSQL database [SQL00]. It combines two major systems from the old DCS database infrastructure: ATLES and TYCHO [ANU00]. According to the Project Overview and Specification [ANU01], the database has to track student, staff and unit information as well as related sub items such as marks and assignments. For this purpose, ASEAM uses the following tables and fields (see E-R-Diagram on page 15):

Student

The table "Student" stores information about students. Its fields are:

- id - the student's id number (primary key)
- username - the student's username (usually the s $\circ$ <id>)
- password - the student's password (currently an empty field)
- stud_name - the student's full name
- initial - the students' initials
- lastname - the student's lastname
- forename - the student's firstname
- sex - the student's sex
- account - the student's account information, if different from username
- intensity - the intensity, either F(ulltime) or P(art time)
- degree - the degree the student is working towards to
- comments - additional comments

Unit

The table "Unit" stores information about units. Its fields are:

- unitid - the unit's internal id number (primary key)
- limittut - the unit's limit on the maximum number of tutorials a student is allowed to sign up for
- limitprac - the unit's limit on the maximum number of practicals a student is allowed to sign up for
- descr - the unit's description
- linkedunitid - the id of the unit which is linked to the existing one
- semester - the semester in which the unit is offered
- year - the year in which the unit is offered

Activity

The table Activity stores information about the unit's activities. Its fields are:

- actid - the activity's id (primary key)
- descr - the activity's description
- maxregd - the maximum number of students allowed to participate in the activity
- location - the activity's location
- type - the activity's type (Lecture, Practical, Tutorial)
- day - the activity's day
- time - the activity's time
- comments - additional comments for this activity
- year - the year in which this activity is offered (currently this field is not used)

Assessment

The table "Assessment" stores information about the unit's assessments and assignments. Its fields are:

- aid - the assessment id (primary key)
- totalmark - the maximum number of points that can be achieved in the assignment
- finalpercent - indicates how strong the assignment accounts towards the final grade
- descr - the assignment's description

Mark

The table "Mark" stores information about student marks. Its fields are:

- studentid - the id of the student who the mark belongs to (primary key)
- aid - the id of the assignment the mark belongs to (primary key)
- amark - the mark itself
- sub_date - the date when the mark was submitted

Staff

The table "Staff" stores information about DCS staff. Its fields are:

- staffid - the id of the staff (primary key)
- username - the username of the staff
- name - the full name of the staff member
- dept - the name of the department the staff member works for

ModifyAllowed

The table "ModifyAllowed" acts as flag indicating whether students are allowed to change sensitive values or not. Its fields are:

- allowed - 1 if the students are allowed to modify information, 0 otherwise

AllowedUser

The table "AllowedUser" stores information of users who are allowed to access the ASEAM system in supervisor mode (in general only available for DCS staff). Its fields are:

- username - the username of the privileged user
- usergroup - the user's group (only for future use, may be left blank)

UnitEnrollment

The table "UnitEnrollment" represents the many-to-many relationship between "Student" and "Unit". Its fields are:

- unitid - the id of the unit (primary key)
- studentid - the id of the student (primary key)

StudentsActivity

The table "StudentsActivity" represents the many-to-many relationship between "Student" and "Activity". Its fields are:

- studentid - the id of the student (primary key)
- actid - the id of the activity (primary key)

UnitsActivities

The table "UnitsActivities" represents the many-to-many relationship between "Unit" and "Activity". Its fields are:

- unitid - the id of the unit (primary key)
- actid - the id of the activity (primary key)

UnitsAssessments

The table "UnitsAssessments" represents the many-to-many relationship between "Unit" and "Assessment". Its fields are:

- unitid - the id of the unit (primary key)
- aid - the id of the assignment (primary key)

ActivityStaff

The table "ActivityStaff" represents the many-to-many relationship between "Staff" and "Activity". Its fields are:

- staffid - the id of the staff member (primary key)
- actid - the id of the activity (primary key)

The PostgreSQL engine allows to update and even insert fields into the database during run time [SQL00]. The main source of information is imported from existing DCS database systems [ANU00] via Perl scripts (which can be found on `http://www.in.tum.de/~baueran/aseam/aseami.pl`). This separate database import (and later also export) module is generally called only once before the beginning of each semester to set up the system. Students and staff create the necessary links, and therefore entries in the many-to-many relationship tables, by using the ASEAM system.

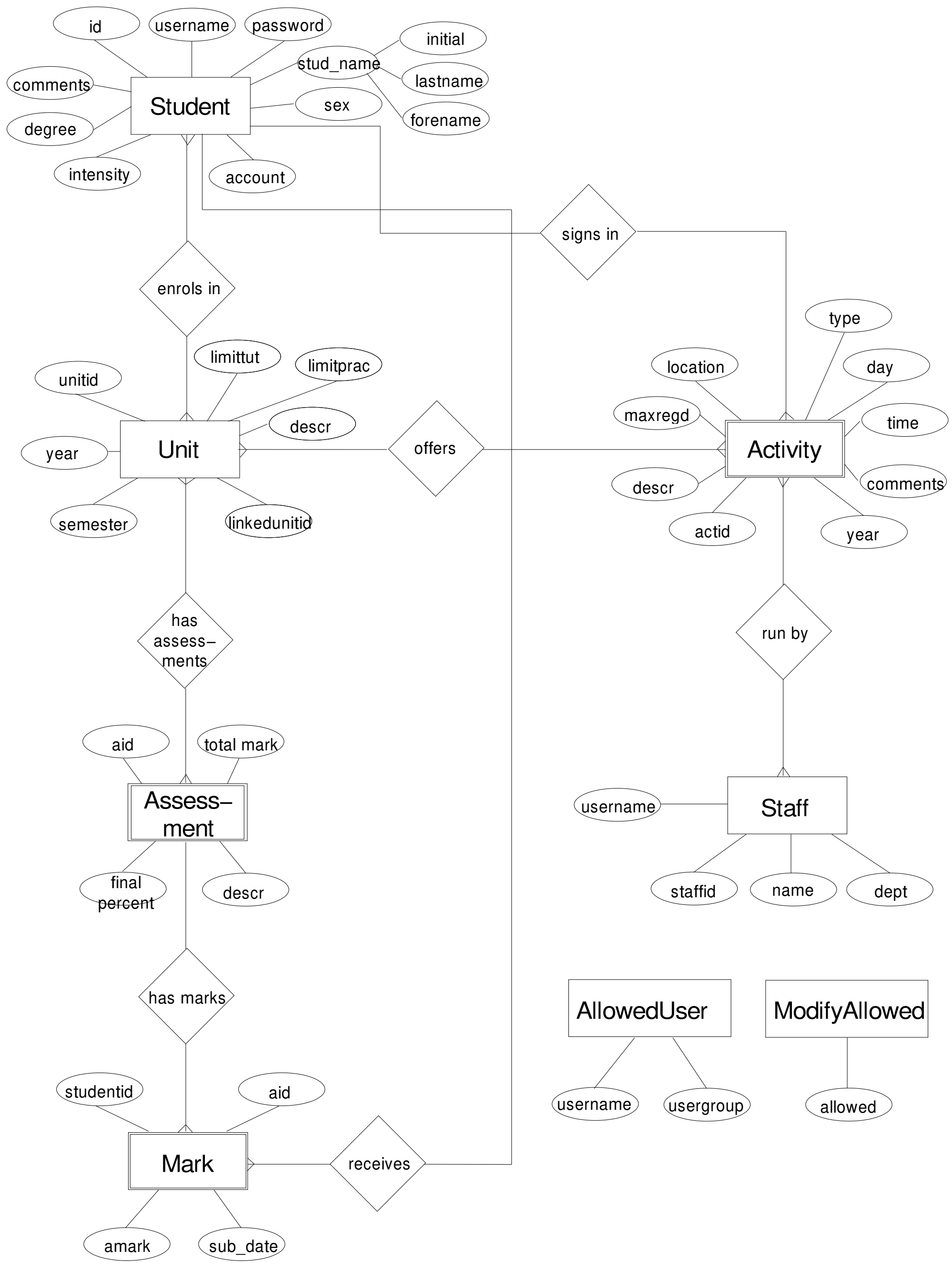

Figure 4: Entity-Relationship Diagram of ASEAM

5 Java Servlet Classes (API)

Instead of outlining the entire API of the ASEAM application within this static document, an easy to navigate description can be found at these URLs: `http://aseam.anu.edu.au/docs/api/index.html` or `http://www.in.tum.de/~baueran/aseam/api/index.html`. These files should be compatible with the commonly used Javadoc [Sun00] standard and are readable by any web browser. However, they should be considered part of this document.

6 Expandability

The ASEAM application can be expanded with new features. Even though there is no 100% consistent concept of how to implement new methods, these guidelines should be considered:

- All user functionality has to be implemented in the HTML code first (either by implementing a new hyperlink or a new button).
- The method which is evoked by pressing the newly implemented button, link, etc. should be called `showStaffXYZ` or `showStudentXYZ` depending on whether this method is meant to be used by staff or students. `XYZ` should briefly describe what the function does. For example `showStaffViewToEnterMarks`.
- Newly implemented methods have to show up either in the `doPost()` or the `doGet()` method.
- The `showUser` functions usually produce HTML output, providing further input elements to the user. In general these input forms have a confirm button evoking other methods.
- The methods which perform the newly provided functionality are then called `handleStaffXYZ` or `handleStaffXYZ`.

The following scheme illustrates roughly how ASEAM distinguishes and processes requests.

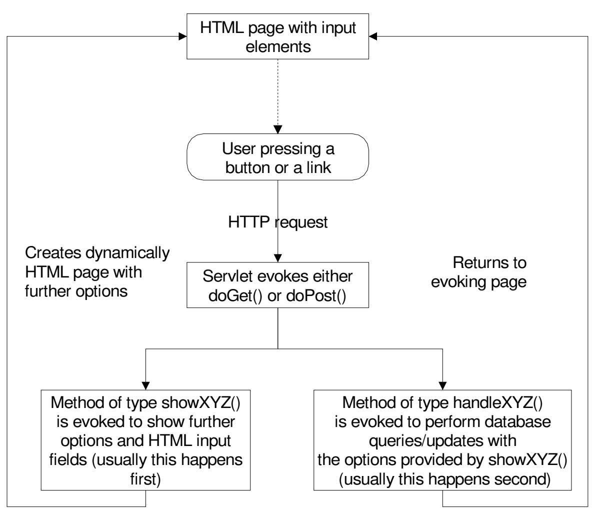

Figure 5: Model of operation

7 Source Code

The Java source code of ASEAM can be found on `http://www.in.tum.de/~baueran/aseam/src/`. Read the API documentation for further details (page 16).

8 TODO

At this stage the ASEAM source code is pretty much a fast 'hack' due to a lack of time and especially man power. However, it works fine and does what it should do, but one should spend more time trying to rewrite major parts of it - just for the sake of readability and integrity.

Here is a list of things that should be implemented to make it more open to other programs and more easy to maintain for future contributors. The list also includes things to do with the web server for example, not just the source code.

- Implement XML to structure information exchange
- Encapsulate database tables and objects to hide the nasty SQL details from other programmers
- Remove all HTML tags from the source code (should be easy once the first two points are taken care of)
- Implement LDAP support
- Write method to display a weekly, coloured time table for students (may be a simple HTML table)
- Write export function for marks and other assignment related information such that a spreadsheet program like Microsoft Excel or Gnumerik can directly import the relevant data

References

[ANU00] Australian National University. Administrative Information Systems. http://tux.anu.edu.au/Services/Ais/index.html, 2000

[ANU01] Australian National University. ASEAM Overview and Specification. http://aseam.anu.edu.au/docs/index.html, 2000

[CBS96] Thomas M. Connolly, Carolyn E. Begg, Anne D. Strachan (1996): *Database systems: a practical approach to design, Implementation, and management*, Addison-Wesley

[Dav98] Thomas Davis (1998): *Build your own object pool.* JavaWorld Magazine, June 1998

[E00] Ralf Engelschall. modSSL - *The Apache Interface to OpenSSL.* http://www.modssl.org/index.html, 2000

[HC98] Jason Hunter, William Crawford (1998): *Java Servlet Programming*, O'Reilley

[SQL00] PostgreSQL. Documentation. http://www.postgresql.org/docs/index.html, 2000

[NIS00] Solaris NIS+ Frequently Asked Questions And Answers. http://www.eng.auburn.edu/users/rayh/solaris/NIS+_FAQ.html, 2000

[Sun00] Sun. Java Documentation. http://java.sun.com/docs/index.html, 2000