2006

# Managing the ACM programming contest

Navin Bhaskar

# Project Report: Contest Management Application

## Committee

Prof. Axel-Tobias Schreiner, Chair

Prof. Rajendra K. Raj, Reader

Prof. Paul Tymann, Observer

Navin Bhaskar
Department of Computer Science
Rochester Institute of Technology
Rochester, New York
E-Mail: nxb8951@cs.rit.edu
November 22, 2005
Document Version: Final

Table of contents

## Abstract

The Association for Computer Machinery (ACM) conducts an international collegiate programming contest held on an annual basis. Teams compete to solve multiple questions within the allotted six hour duration of the contest. A panel of judges grades the solutions online. The team that answers the maximum questions correctly wins the contest.

This application will assist in the process of selection of winners of the programming contest. It will make the process of submission of solutions, and grading of the answers convenient. The ACM application is a Java based and web-enabled application and therefore platform independent.

## 1. Introduction

This project report describes the design, function, implementation and deployment of a contest management application for conducting programming contests. The first part of this document is primarily dedicated to product architecture, design and implementation. The latter sections compare and contrast it with currently available solutions and technology options. Two software products have been evaluated, PC$^{2}$ [2] and Mooshak[1]. With minor configuration changes, this contest management application can be reused to conduct other programming contests held along similar lines. System prerequisites, installation, configuration and deployment details are also elaborated.

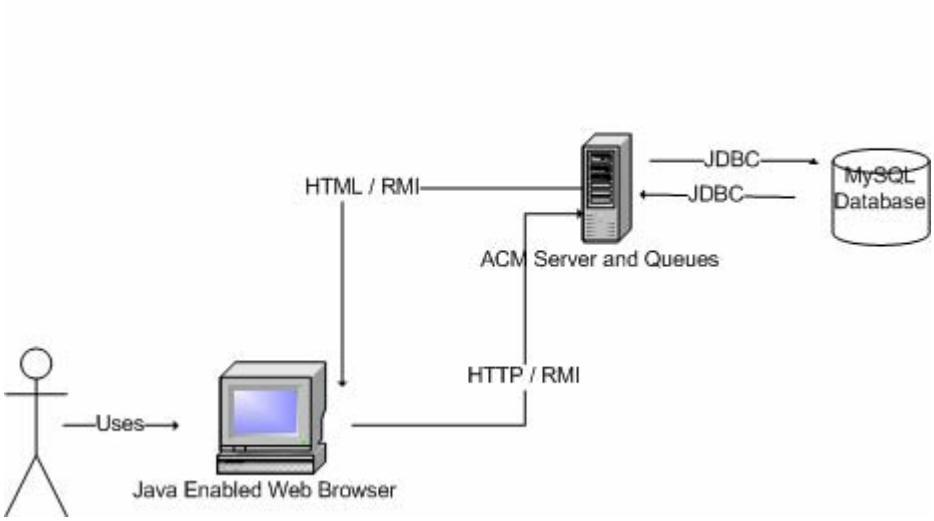
## 2. Objectives

This product is primarily meant to assist in conducting the annually held ACM programming contest. With minimum configuration changes, it should be reusable for contests held along similar lines. It should not rely on proprietary libraries, use of which would limit the customization and/or enhancement in the future. The code is modularly organized and implementations have been wrapped using Java language interfaces to modify the implementation without breaking the application. This product is also compared with 2 available products namely Mooshak[1] and PC$^{2}$[2]. The first product is based on legacy CGI technology; the second requires separate installation. Implemented in Java using distributed and object-oriented technologies, this contest management software requires minimum configuration and installation steps.

## 3. Initial Hypothesis

Unlike legacy CGI based contest software employing process switching; a thread-based design would provide performance benefits. Though modeled for primarily assisting in conducting the ACM programming contest, this product would be generic enough to permit application reuse for other contests. Being rich in thread and network protocol API's, Java technology could be used to develop and demonstrate a robust, modular and web based contest management application without operating system, platform or shell (command line interpreter) dependencies. Applet-Servlet communication and RMI technology could be leveraged to implement a near-real time scoreboard. Modular coding, packaging and technologies could be utilized to extend the product in the future.

## 4. Architecture




The ACM contest management application is a 3-tier web-application with the middle tier being the application server and the final tier being a relational database (storage tier) to log contest activity. The web application is comprised of many web pages accessed using a web browser. Users have to be authenticated and authorized by the application server to access the web application. Authentication is achieved by password comparison. Authorization is verified by comparing the role assigned to the user with the role required to access the resource. This security mechanism prevents teams from accessing web pages allowed to judges. The server application is a Java Servlet implementing the *Remote* interface. A single Servlet services all client requests. The ACM server is responsible for logging the contest activity, storing uploaded team submissions for subsequent grade assignment by a judge and refreshing client scoreboards with the latest standings. Contest activity is maintained as records in a database table. JDBC API's allow the ACM server (or simply server) to access and update the database tables. The choice of a relational database makes the task of storage and retrieval of data convenient. Additionally, creation of reports in the future is easy. Only one client can be used from a machine at a time because of resource constraints.

Teams use a web form to upload solutions from their machines. HTTP POST mechanism is used to upload the zip archive containing the team submission to the ACM server. Subsequently, a judge downloads the solution and assigns a grade after matching the output with the correct solution. Solution submission by a team and grade assignment by a judge causes the standings to change. An automatically refreshing scoreboard allows the participants to view the latest standings. Scoreboards are also capable of sorting the standings automatically; the leading team is always displayed at the top. A team that solves the maximum number of problems correctly is adjudged the winner of the programming contest. In case of a tie, minimum cumulative time is used to determine the winner.

The scoreboard is implemented as an AWT Applet implementing the *Remote* interface. The ACM server exposes *Remote* methods to enable participant scoreboards to register

with it. Scoreboards self register with the server during Applet initialization. Self registration involves providing its *Remote* reference to the server. As soon as the standings change, the server updates each registered scoreboard with the latest standings (server initiated RMI method invocation).Visually, the judge and team scoreboard appear to be identical. However, the scoreboard available to a judge is context sensitive. It allows a judge to download team solutions and assign grades with right-click of a mouse. Once the submission has been downloaded by a judge, the process of archive extraction and output matching is manually performed. Unlike judges and teams; an administrative user is responsible for assisting the judges with the contest and not considered as a participant.

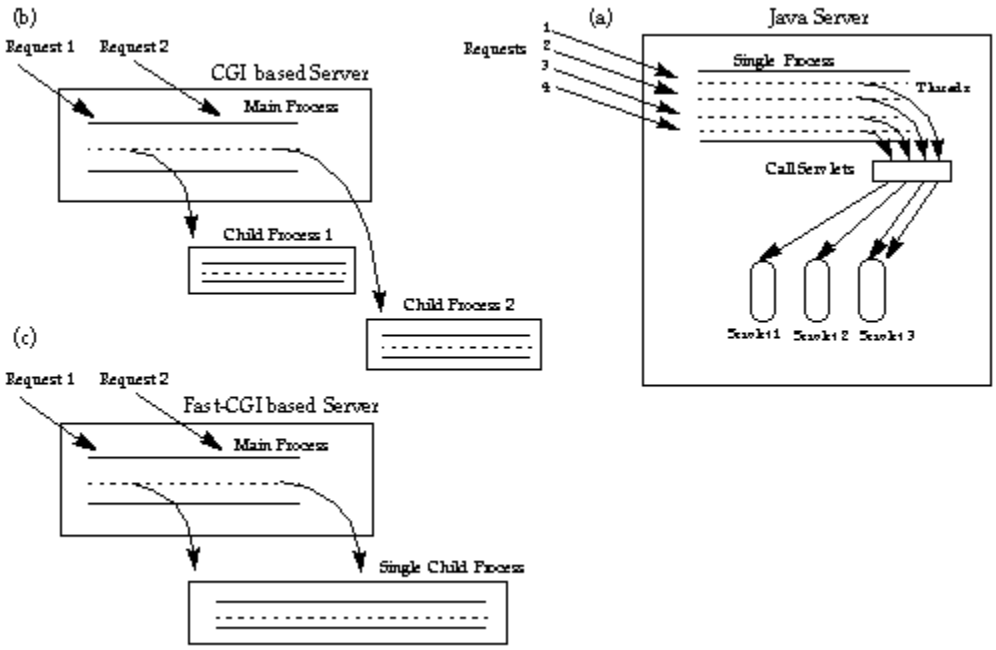
Teams can seek clarifications from judges for ambiguous questions. Clarifications are forwarded to the judge by the ACM server as an e-mail using Java Mail API involving mail protocols like SMTP and POP3/IMAP. Secure versions of mail protocols are also supported. File submission by the team and grade assignment activities are logged in the database during the duration of the contest. Clarification requests are not logged in the database. Though this implementation will be deployed on Apache Jakarta Project's Tomcat Application Server, it can be ported to any J2EE compliant application server.

# 5 Technology Overview

## 5.1 Java Servlet technology

A Java class is said to be a Servlet if it sub-classes (or extends) a basic Servlet implementation (*GenericServlet*) or a HTTP (*HTTPServlet*) specific implementation. Servlets are regular classes allowing the use of all Java programming language syntax, API's, language features and constructs. Unlike a Java application that is executed from the command line (as a separate JVM/process), Servlets execute within the application server's JVM process.

Unlike a legacy CGI server that spawns a new child process per-request to generate dynamic content [sections (b) and (c) in the figure below], a Servlet services client requests on separate threads within the server's (JVM) process. Each client request results in a call to the Servlet's *service* [section (a) of the figure below] method. The overhead of a thread context switch on the underlying operating system is substantially less than a process context switch.




(Source: http://java.sun.com/products/Servlet/whitepaper.html)

Servlets are loaded, instantiated and initialized by the application server's Servlet container (to be referred henceforth as container). The main advantage of using Servlet technology is that the container manages the protocol. Servlets permit the use of instance variables; however they need to be explicitly synchronized because of concurrent (multi-threaded) access.

A Servlet Container (container) is responsible for loading and managing Servlets. Web requests are intercepted by the container and routed to the requested Servlet based on information available in the deployment descriptor. A deployment descriptor is a XML configuration file containing Servlet information. If the Servlet is listed (in the deployment descriptor files) but not loaded in the memory, the container attempts to load the Servlet class from the file system using a chain of class loaders. If the container determines that this is the first request for the Servlet, then its (Servlets) *init* method is invoked.

For each request from the web browser, the Servlet parses the input stream and creates new instances of *ServletRequest* and *ServletResponse* objects. The *ServletRequest* instance contains client HTTP parameter, encoding and content information. The *ServletResponse* contains cookie, status code, and header information encapsulating HTTP response provided in response to a client request. The container invokes the *service* method and passes both of them as parameters. The container invokes the *service* method of the Servlet for each request.

Conversely, when the server is being shut down or attempting to free memory, the container calls the Servlet's *destroy* method before the Servlet class is unloaded from the

memory. A catastrophic failure like the process being killed prevents the invocation of this method. The *destroy* method is an opportunity for the unloading Servlet to clean up/close resources by itself.

### Servlet lifetime

From the time a Servlet is loaded by the Servlet container to the time it is ready to be garbage collected, three methods are invoked. Therefore, these methods are called *lifetime* methods. Sub classes can override these methods to provide additional functionality.

- Method: *init* - Once the Servlet class is loaded by the Servlet container (‘s class loader), it is guaranteed to invoke the *init* method on it. Default implementation (provided by the super class) can be extended to provide application specific initialization steps. A Servlet can throw *ServletException* during initialization. It can service requests only if the *init* method has returned normally.

- Method: *service* - The container is responsible for routing each client request to the Servlet’s *service* method. Each *service* method executes concurrently in a separate thread within the server process unless the Servlet implements the *SingleThreadModel* interface. For such a reason, all instance members need to be explicitly synchronized.

- Method: *destroy* - This is a container call-back method invoked prior to a server shutdown or a Servlet unloading. It is an opportunity for the Servlet to return and close important resources like files, sockets and database connections. It is reciprocal to the *init* method. Once a Servlet has been destroyed, it can be garbage collected by the JVM.

The *init* and *destroy* methods are called once; the bulk of the Servlet’s logic is within the service method.

## 5.2 Java RMI technology

RMI technology allows method invocations between Java objects residing on disparate hosts (*Remote* objects). The *caller* and the *callee* can potentially reside on different machines. The *callee* has to exist before the *caller* can invoke methods on it.

Syntactically, a RMI method invocation appears similar to an in-JVM call (*caller* and *callee* executing in the same JVM). For such an invocation to happen the *callee* or *Remote* object must implement a marker (*Remote*) interface and all distributed methods must throw *RemoteException.* The *callee* then registers itself with a naming facility in order to be looked-up by *callers*. Sun Microsystems provides an executable (*rmiregistry*) implementing a basic naming service. The caller/client looks up the remote object at naming service based on a key. If the lookup is successful, the client/caller has a reference to the remote object on which it can invoke distributed methods. The reference obtained as a result of the successful lookup is a local surrogate or stub of the *Remote* object implementation. RMI technology uses a combination of marshalling (with the use

of serialization) and socket technologies to pass the parameters from the *caller*, execute the method (on the *Remote* object) and return a value (or *Exception*) to the *caller*.

## 5.3 Java Applet technology

Java Applets are client-side executing Java code. They execute within the web browser. They are transported along with the web page usually in the form of binary jar/zip files. Applet's can make socket connections to the server that sent it to the client. Applet lifetime is closely related to the actions performed by the user accessing the web page (containing the Applet). Applet lifetime methods are summarized below:-

- Method: *init* – One time initialization of the Applet.
- Method: *start* – Page (re)visit or Applet (re)load.
- Method: *stop* – When the control leaves the page containing the Applet or user closes the browser window.
- Method: *destroy* – Release of resources requested before being unloaded from the memory by the browser. It is a reciprocal method to init. It is also an opportunity for an Applet to clean up resources after use.

## 5.4 Java Server Pages technology

It is used to dynamically generate HTML content based on embedded Java code between HTML tags. Interestingly, a Java Server (JSP) Page compiles into a HTTP Servlet. The additional benefit is that the application server's web container is responsible for compiling the JSP file containing the source code to a Servlet. All underlying Servlet API's are therefore accessible from a JSP page. JSP simplifies programming by allowing the use of implicit objects. Majority of the implicit objects are classes directly related to Servlets. It is also possible to programmatically invoke other Servlets from within the JSP code.

### 5.4.1 Implicit Objects

JSP extends Servlet technology by providing implicit objects for ease of use. They include but are not limited to

- *request - javax.Servlet.ServletRequest*
- *response - javax.Servlet.ServletResponse*
- *session - javax.Servlet.http.HttpSession*
- *application - javax.Servlet.ServletContext*

## 5.5 Application Server

The importance of the application server in all the technologies cannot be understated. All the technologies have to be supported by the application server. It contains the Servlet Container responsible for loading the Servlets. Application server also provides additional facilities like database connection pooling, naming lookup facility etc. Though

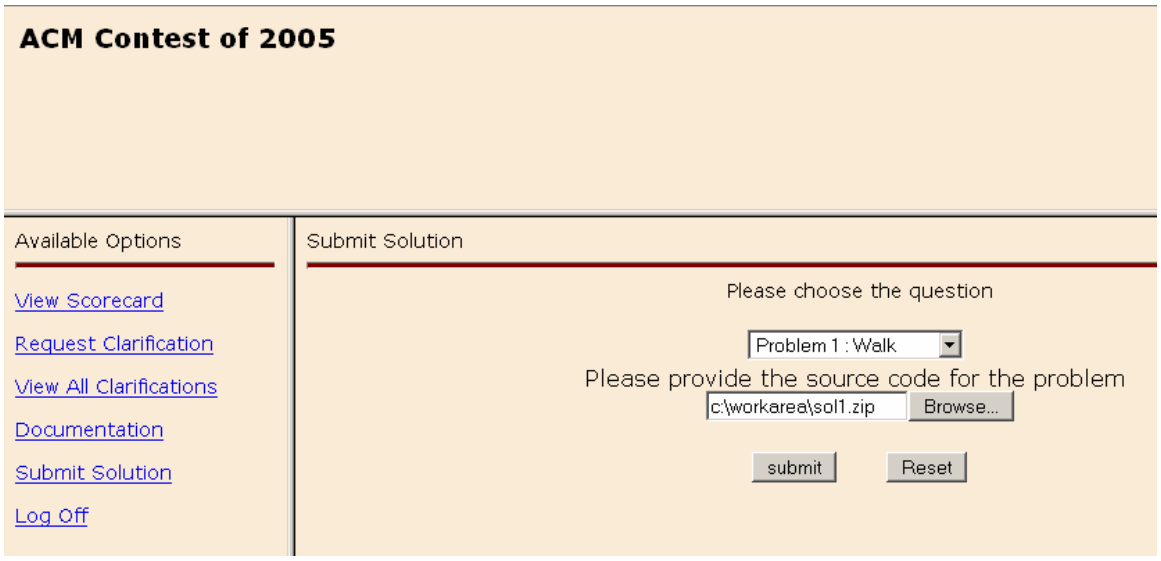
this implementation is deployed on Tomcat Application Server (henceforth referred to as Tomcat), it can be deployed on any J2EE compliant application server.

# 6. Application: User Interface

Being a protected web application, users will be challenged to provide a valid user ID and password to access the contest management software. Even after the user has successfully logged in, the application server (hosting the web application) verifies if the user has the required role before serving each web page. An administrator is responsible for user setup. User setup is a two step process, the first step involves the adding a user (team or a judge) to the list of application server users. Tomcat maintains user information in a tomcat-users.xml file. The second step involves maintenance of additional contest specific information. This step is required since the application server does not maintain additional attributes required for a product user. The details maintained for teams include name of the institution, user ID to access the application and password to access the e-mails. Similarly, the administrator also updates information relating to a judge including e-mail address, room number and contact phone number. This information is separately maintained in product specific userplus.xml file in XML format. This file exclusively maintains information for users participating in the contest. Like all the users participating in the programming contest, an administrator also has to be an existing application server user. Once the user setup is complete the administrator has updates the start time, end time and contest title. Once the application is redeployed with the contest specific configuration, teams and judges can start using the web application. Sometime prior to application redeployment, the teams are provided with a hard copy of the contest problems. Teams log into the application before uploading the submissions. Similarly judges log in to answer clarifications and assign grades. The different options based on the roles available to a judge and team is described in detail.

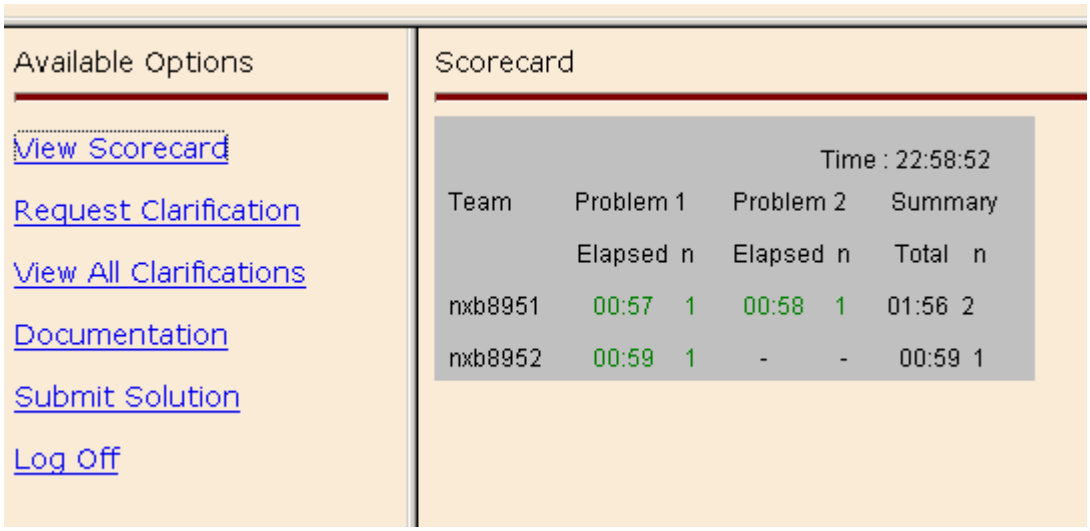
## 6.1 Options: Team

a) Submit solution – During the duration of the contest, teams upload/submit source code and build files in the form of a single zip archive. A web form allows the team to upload the team submission containing this archive. A copy of this submitted file is downloaded by the judge to assign grades. The file upload web page provides a drop-down menu to select the problem for which the solution is being uploaded. Once the team confirms it by pressing the “Submit” button, the ACM server makes a copy of the uploaded solution. The correct output based on the problem will also be inserted in the archive being maintained by the server. All scoreboard Applets will also be notified indicating (by incrementing the count of number of submission and elapsed time) after receipt of this submission by the server.




(Please note: The right and the bottom part of the above screen shot has been trimmed for formatting)

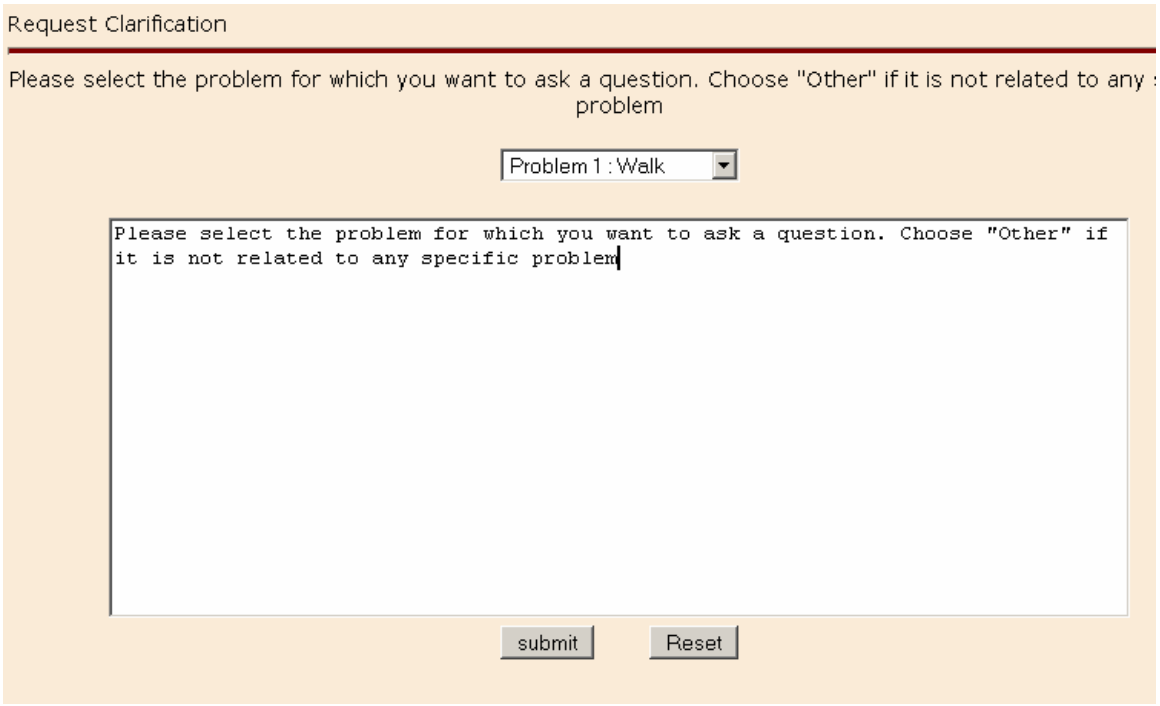
b) View Scorecard – This functionality allows the teams to view the latest standings. The scoreboard is a matrix of problems and teams. Teams are arranged horizontally and problems vertically. As soon as the teams make a submission or the judges mark the uploaded solution to be correct, the scoreboard gets refreshed automatically.



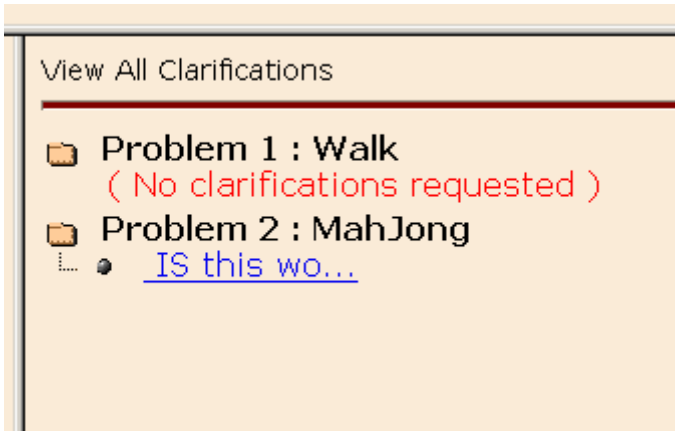

(Please note: The right and the bottom part of the above screen shot has been trimmed for formatting)

c) Request a clarification – This functionality allows the teams to request for a clarification. Teams can use this facility if they feel there is any confusion or ambiguity in the problems. The provided text is sent to the ACM server using HTTP POST mechanism. The ACM server forwards the clarification to the

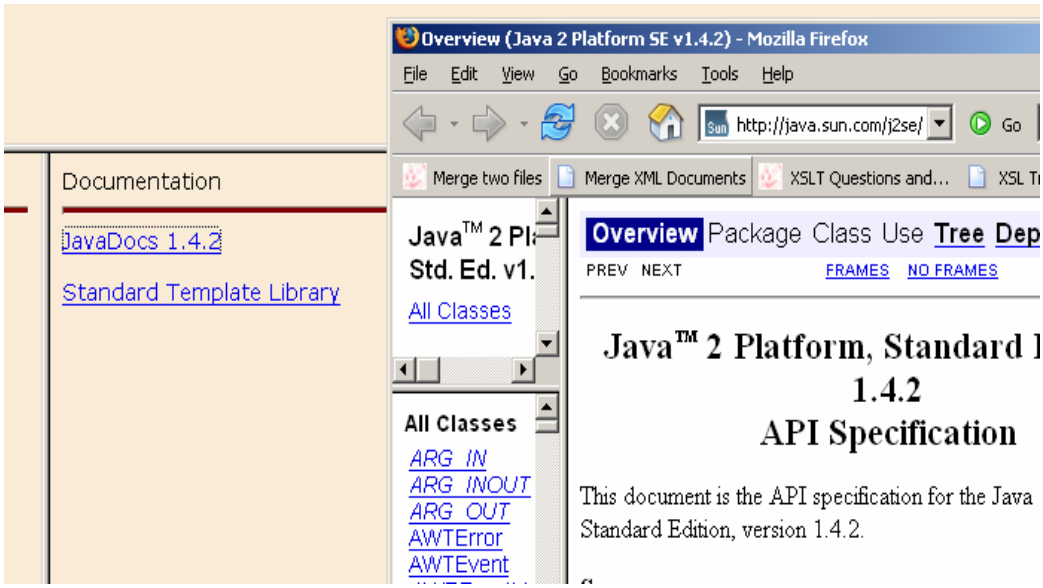
judge as an e-mail using Java Mail API's. The judge can use any mail client to reply to the team's clarification. The judge has to manually forward the e-mail to a group account so that the information is available to the benefit all the teams.




d) View all clarifications – This facility allows the teams to centrally view all the clarifications. Clarifications are sorted by problems. Clicking on a clarification allows the team to view the entire exchange of communication between the team and the judge in detail.

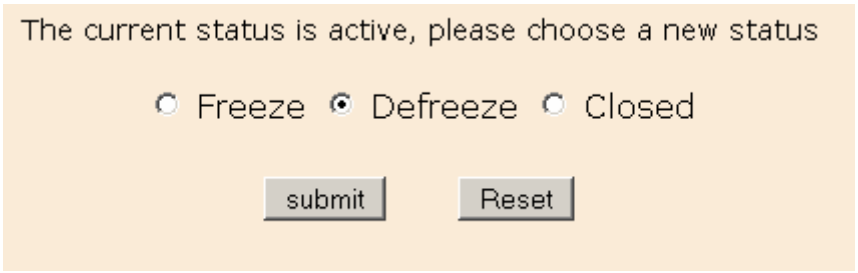



e) View API documentation – This facility allows the teams to refer to web resources to solve the problems. The URL information is maintained in the Docs.xml file and is configured before the beginning of the contest.



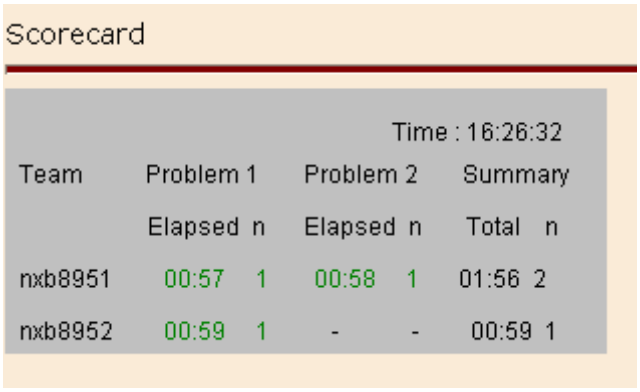

## 6.2 Options: Administrator

a) Freezing the scoreboard – This facility allows an administrator to freeze the scoreboard until the time; the judges have graded all the solutions and chosen the winner. Teams can continue to upload the solutions but the scoreboard will not get refreshed. The defreeze option will return the application back to the active state.

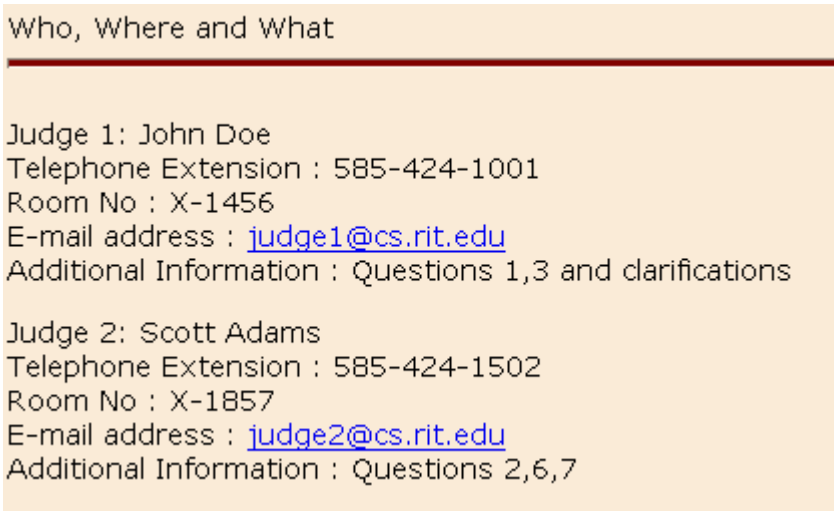



## 6.3 Options: Judge

a) View / Grade solution – As mentioned before, the team and judge scoreboards appear to be the same. However judge scoreboards are context sensitive. They allow operations using the right-click of the mouse (pointing device). Judges can download team submissions and also assign grades.




b) Look Up judge information – This is an informative page containing information about the location and contact details of other judges.




# 7. Application: Functioning

The container implements the *ServletContext* interface. This interface provides methods to acquire a *RequestDispatcher* for the current *ServletContext*. The *RequestDispatcher* can be used to *include* or *forward* a request to a target Servlet for further processing. The two methods differ in target Servlet's ability to modify the request header and response body. Once the *include* method is invoked on the ACM Servlet, the *service* method gets invoked. This mechanism can be used to centrally route all the contest activity through the ACM Servlet. This helps the server to assign consistent timestamps to each team submission, grade assignment and provide timely scoreboard updates.

## 7.1 File upload/submission and logging

This product accepts team submissions in the form of a zip archive. Teams can submit any number and type of files within the zip archive necessary for the compilation and execution of the program. As mentioned before teams have to select the problem for which the solution is being uploaded from a drop-down menu. When the team submits the zip archive the following sequence of events occurs:-

1) The contents of the web form including the zip file are submitted using HTTP POST mechanism.

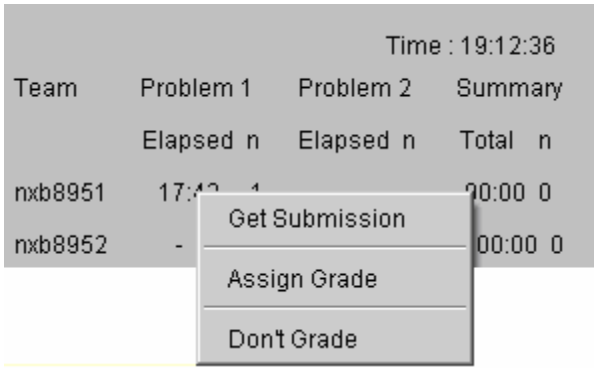
2) Before the form is submitted, a numeric value (or action code) is set in *request* object to be read by the ACM Servlet's *service* method. Team name and problem number of the file being uploaded are also set.
3) The *RequestDispatcher* reference for the ACM Servlet (target Servlet) is acquired and JSP file's request and response reference are passed as parameters to the include method. Calling the *include* method on the Servlet using the *RequestDispatcher* allows the control to be passed to the ACM Servlets *service* method.
4) The numeric value (or action code) in the request allows the server to handle grade assignment and file upload differently. Based on the action code specified in the request (specified before the file upload), the ACM Servlet delegates the task to the correct method of the *ServiceProvider*. In case of a team submission, it parses the multi-form data and stores a copy of the file submitted by the team. A new record is also inserted into the ACTIVITY table to acknowledge the receipt of the team's submission.
5) The Servlet's *service* method updates a flag to indicate that team standings have changed. Based on this flag, the "*notifier*" determines that the standings have changed and invokes the *enqueueScore* method on each scoreboard. The scoreboards are refreshed to have the latest scores.

## 7.2 Obtaining the scoreboard implementation

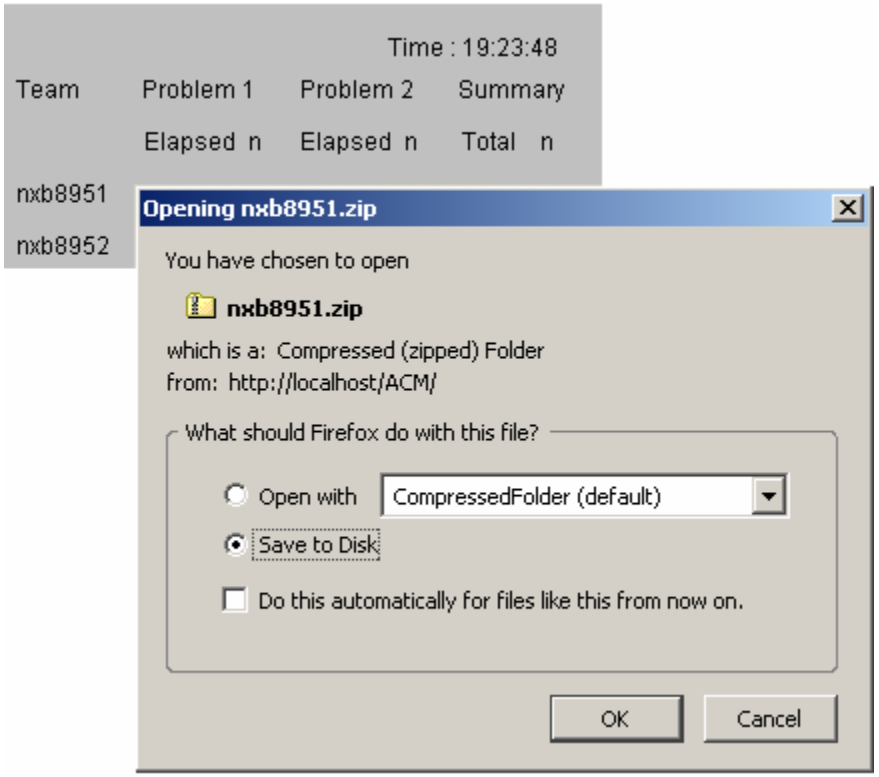
The class files containing the team and judge Applets are packaged into separate zip files at placed at the root of the web application. This location is called the Applet codebase and is a URL relative to the root of the web application. When the team or judge clicks on the web page containing the Applet, the zip file is automatically downloaded from location specified by the *codebase* attribute. Apart from containing the class files required for painting the scoreboard (and related data structures), the archive also contains surrogate or stubs classes for the ACM server. The scoreboard uses it to invoke remote methods on it.

## 7.3 Downloading latest team submissions

Judges have a facility to download the team's submissions directly from the scoreboard. The scoreboard Applet available to a judge is context-sensitive, which means that the judge needs to right-click on a mouse and choose the "Get Submission" option to download the zip archive containing the solution. The menu is implemented as a *PopupMenu*. The event generated as a result of the mouse click is handled. The actual download is implemented by redirecting the judge to a web page which independently downloads the latest submission from the file system based on the logs from the ACTIVITY table. File download feature is achieved by setting the header as "Content-Disposition" and writing the contents of the file to the Response stream.




**Judge Scoreboard [downloading a team's latest submission]**




## 7.4 Scoreboard refresh and computation

During scoreboard initialization, the Applet registers itself with the ACM server. During this self-registration process, it provides its *Remote* object reference to the ACM server. This allows the ACM Server to maintain a collection of scoreboard references in a hash table. The scoreboard Applet's being *Remote* objects themselves can be invoked from the ACM server. This mechanism is used by the ACM Server to implement client-callbacks. When the ACM Servlet is initialized, it spawns a "notifier" thread. Periodically, this thread determines if the standings have changed. Once the standings have changed, the notifier thread reads all the records from the ACTIVITY table and creates and populates the *Score* data structure for each problem and team. An array of this data structure containing the standings for the problems solved by team is transported to the scoreboard

Applet via RMI. This information is used by the Applet to recreate the scoreboard with the latest standings.

## Implementation: Scoreboard

The scoreboard is implemented as an Applet implementing *Remote* interface. Visually the scoreboard visible to the judges and teams appear to be the same. A scoreboard available to a judge has the capability to download the latest team submissions and assign grades. Each row of the scoreboard (excluding the headers) is a *Panel* implementing the *Comparable* interface. Teams are arranged in the descending order based on the value in the summary column (last row in each row).

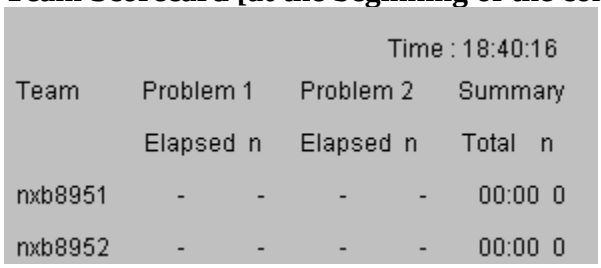
## Computation and Logic

Once the standings have changed the ACM server calls the *enqueueScore* method on the scoreboard Applet using its *Remote* reference. The server invokes the *enqueueScore* method and provides the *Score* array as parameters. Each *Score* data structure represents the standing for a problem and team. It is rendered automatically.

## Scoreboard auto-refresh

The scoreboard is computed dynamically. Whenever a team makes a submission or judge assigns a grade, the scoreboards are refreshed. The scoreboard is a matrix made up of teams arranged horizontally and problems vertically.

**Team Scorecard [at the beginning of the contest]**



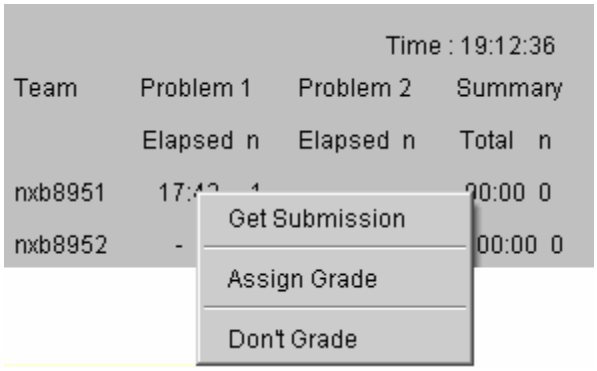
Time : 18:40:16

| Team | Problem 1 | | Problem 2 | | Summary | |
|---|---|---|---|---|---|---|
| | Elapsed | n | Elapsed | n | Total | n |
| nxb8951 | - | - | - | - | 00:00 | 0 |
| nxb8952 | - | - | - | - | 00:00 | 0 |

**Team Scoreboard [after uploading a solution]**

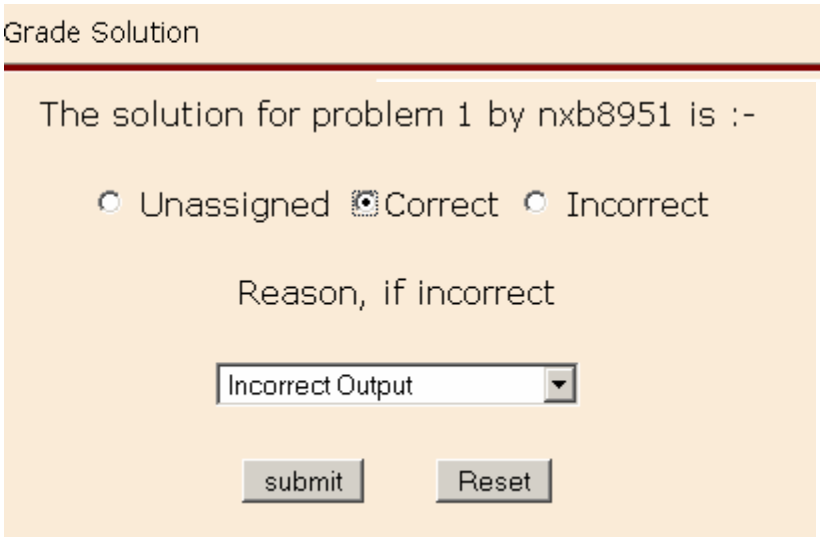
As soon as the teams upload the solution for a given problem, the scoreboard refreshes itself. If the problem is being attempted again the count on the problem column is incremented. The elapsed time also changed. The elapsed time is the difference in time when the submission was made to the time the contest started and is displayed in hour-minute-second precision format. The *Scoreboard* available to a judge appears visually similar to the teams; however the major difference being that it is context-sensitive. A judge is displayed a pop-up menu when the judge right clicks over team standings. The options available to a judge include grade assignment and downloading of team solutions.

**Judge Scoreboard [Assigning the grade]**




Once the latest submissions have been downloaded, the judge has to externally compile the source filed using the build scripts provided by the teams. Output matching with the correct solution has to be done manually by the judge before grade assignment. The correct/required output for a problem is automatically inserted into the archive by the ACM server before the solution containing the team's solution is saved in the server file system. A judge compares the output of the team's submission with the correct solution prior to grade assignment. The "Assign Grade" option redirects the judge to a web page with radio buttons to assist the judge in assigning a grade. If the judge is to mark the solution as incorrect, a detailed reason is also provided in the form of a drop-down menu.

**Judge Scoreboard [marks them as correct]**




As soon as the judge marks the solution as correct, all scoreboards are automatically updated with the latest standings.

Time : 19:17:27

| Team | Problem 1 | | Problem 2 | | Summary | |
|---|---|---|---|---|---|---|
| | Elapsed | n | Elapsed | n | Total | n |
| nxb8951 | 17:43 | 1 | - | - | 17:43 | 1 |
| nxb8952 | - | - | - | - | 00:00 | 0 |

## Scoreboard Sorting

The judge and team scoreboards get sorted automatically at run time. Teams leading the contest are arranged at the top of the row. Sorting is exclusively performed based on the summary column. For a team, this column displays the cumulative time for correct submissions and the total number of correct submissions. At the end of the contest, the winning team would have solved the maximum number of correct solutions in minimum cumulative time.

Time : 19:29:35

| Team | Problem 1 | | Problem 2 | | Summary | |
|---|---|---|---|---|---|---|
| | Elapsed | n | Elapsed | n | Total | n |
| nxb8951 | 17:43 | 1 | - | - | 17:43 | 1 |
| nxb8952 | - | - | 18:19 | 1 | 18:19 | 1 |

The scoreboard sorts itself on the basis of total number of submissions followed by least cumulative time in that order. Correct submissions are marked in green color.

Time : 20:04:05

| Team | Problem 1 | | Problem 2 | | Summary | |
|---|---|---|---|---|---|---|
| | Elapsed | n | Elapsed | n | Total | n |
| nxb8951 | 00:57 | 1 | 00:58 | 1 | 01:56 | 2 |
| nxb8952 | 00:59 | 1 | - | - | 00:59 | 1 |

## Safeguards against thread freezing

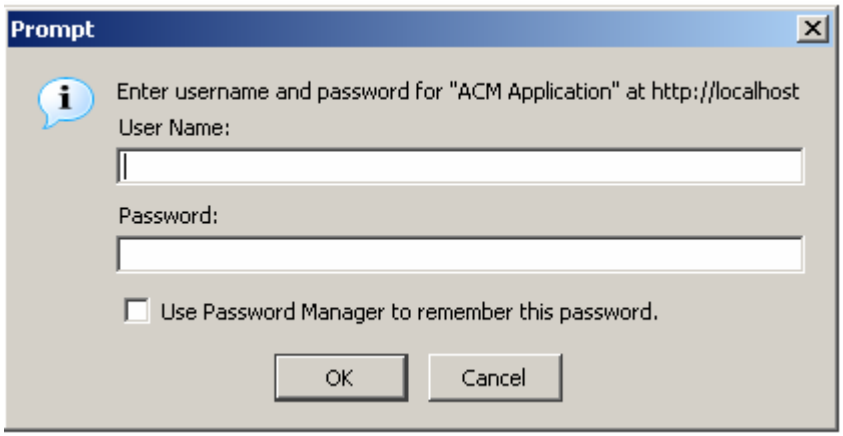
The ACM Server invokes the *Remote enqueueScore* method of the scoreboard whenever there is a change in standings. This method internally calls the *refresh* method of the scoreboard. The *refresh* method obtains the *SystemQueue* reference and calls the *invokeLater* method. This method requires a target reference whose *run* method will be invoked. The scoreboards implement the *run* method themselves; therefore the "*this*" reference is passed as a parameter to this method. This method in-turn calls the *run* method and requests the event-dispatching threads to execute the steps required to refresh the scoreboard. The *invokeLater* method returns immediately since the code in the invoked *run* method is executed concurrently.

## 7.5 Clarification mechanism

Teams can seek clarifications from judges for ambiguous questions. A judge replies to these questions. Clarifications are classified on the basis of problem numbers. Teams provide the text of the clarification in the text area provided. The text is submitted to the ACM server using HTTP POST mechanism. This submitted text is programmatically converted to an e-mail and transported to the judge in the form of an e-mail. Java Mail API supports SMTP protocol. The e-mail is addressed to a single judge in the contest. The judge replies to the e-mail using any e-mail client of choice and forwards the e-mail to a group account so that this information is shared by all participating teams. This e-mail is available in the "View All Clarifications" page. This page displays all contest related e-mails. E-mails are displayed by using pattern matching mechanism (on the subject data element) to selecetively display contest related e-mails. POP and IMAP protocols are supported by Java Mail to obtain e-mails from the mail server.

## 7.6 Application security

Only bona-fide users having the required roles can access the ACM application. Three roles have been defined for this product. From an application perspective, any user assigned an "*acmTeam*" role is a participating team. Similarly "*acmJudge*" and "*acmAdmin*" roles also exist for judge and administrative users respectively. The ACM application authorizes and authenticates the user before the page is displayed to the user. This security feature prevents unauthorized access to web pages accessible to users having different roles (even if the URL to access the web page is known to a potentially malicious user). An administrator is responsible for creating the contest users and assigning them appropriate roles. With respect to Tomcat, the administrator has to add users first to the Tomcat tomcat-users.xml file prior to adding the contest participants. This is necessary because the resources are protected using system user information.

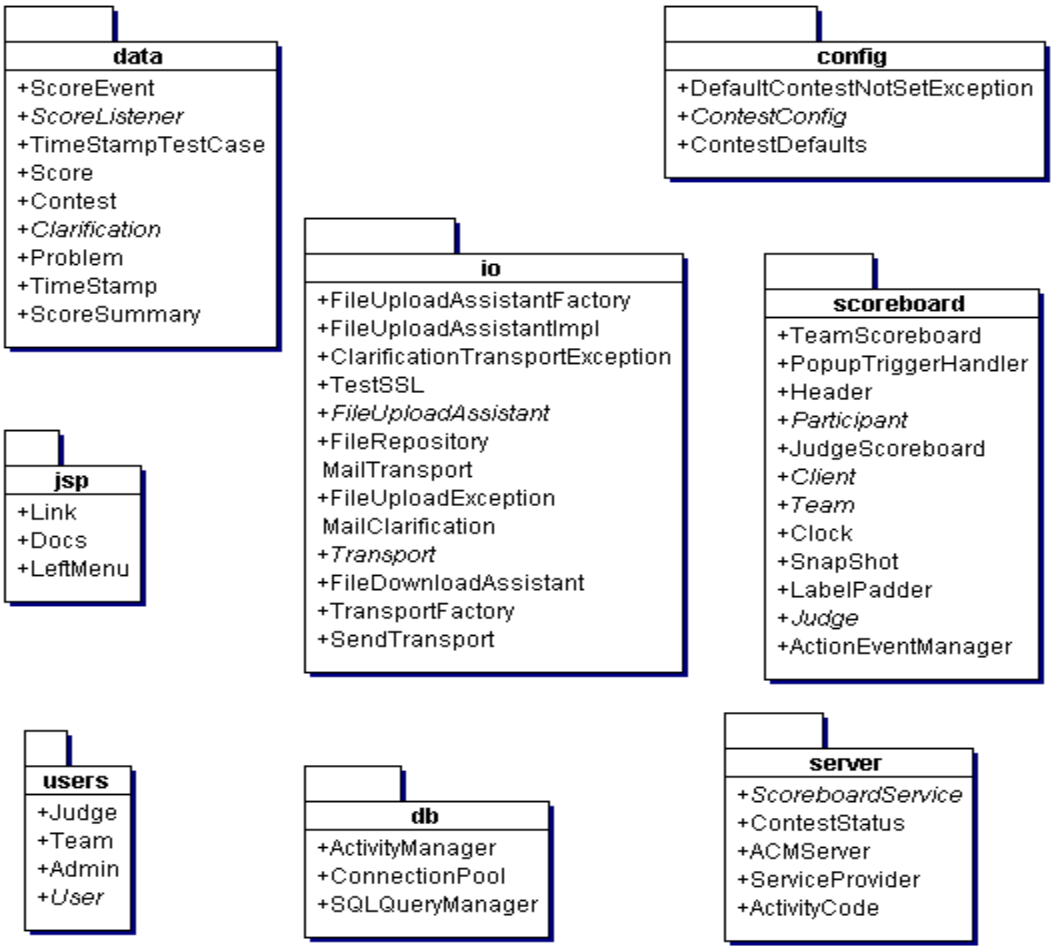



In case participants attempt to access a web page normally unavailable to them, they will receive an error message informing that the user's credentials do not permit access to the requested resource. Each web page in the application is protected by roles. The page-wise protection is specified in the web.xml web application deployment descriptor file.

## 8. Application: Packages

The ACM application is packaged into a single Web Application Resource or WAR file. Packaged within the WAR file are the JSP files, style sheets, images and Java classes that comprise the application. The *Application Deployment Guide Document* describes the WAR file and its content in complete detail.

Some of the most important classes and their use will be described in detail.
(Please note that this is not a comprehensive list of all the classes constituting this application. Some have been omitted for the sake of clarity.)

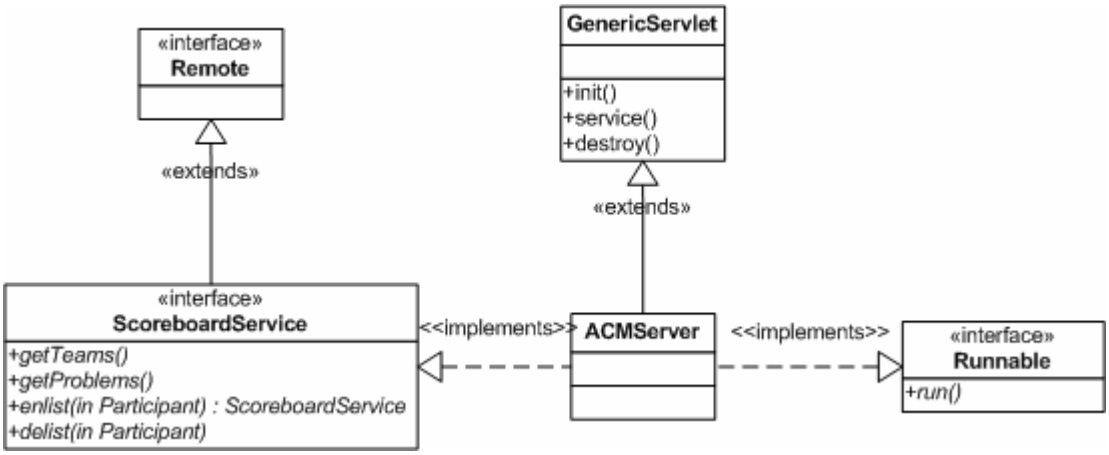



### 8.1 Package: server

### Class: ACMServer

This class is a Servlet class. The *init* method creates the subdirectories for storing the team submissions, subdirectories containing the judge's data file to compare during grading. It also spawns a "*notifier*" thread that updates the clients with the updated

standings, in case of a change. The service method accepts team submissions and delegates it. It services judge's grading requests and administrator requests to freeze and unfreeze the scoreboard.




The ACM Server sub classes a *GenericServlet* and provides implementations to *Runnable* and *Remote* interfaces. The *init* method is inherited from the *GenericServlet* super class. The ACM Server overrides this implementation by doing the following activities in the order given below. The ACM Server creates the directory structure required for the storing team submissions. This is an optimization to avoid directory creation at run-time. It also creates the directory structure for storing the correct output. The directories created will be same as the number of problems for the contest. The text files in these directories will be automatically inserted when the team submits a given problem. This will assist the judges in comparing the output from the team's submission with the correct solution before grades are assigned. The *init* method also starts a "*notifier*". This thread is responsible for updating the client scoreboards with the latest standings.

All contest specific requests are routed through the ACM Servlet's service method. Therefore, this method is organized on the basis of activity codes. This is done by invoking the *include* method on the ACM Servlet's *RequestDispatcher*. Before the include method is invoked, a numeric activity code is set to the request object. The service method understands the steps to be performed based on this activity code. Syntactically this method is broken down into many cases for a switch statement. The *destroy* method stops the "*notifier*" thread from running.

## Class: ServiceProvider

The ACM Server delegates the task of storing the team submission as an archive in the correct directory and updates the logs (ACTIVITY table) appropriately with the available information. Each team submission is handled by this class.

## Interface: ScoreboardService

The implementation has to override *enlist* and *delist* methods. Only enlisted clients receive standing change notifications. The onus of delisting is on the clients themselves. It happens automatically in the *destroy* method of the Applet. Recall that scoreboards are

implemented as Applets. This interface also provides methods to provide information about participating teams and contest problems.

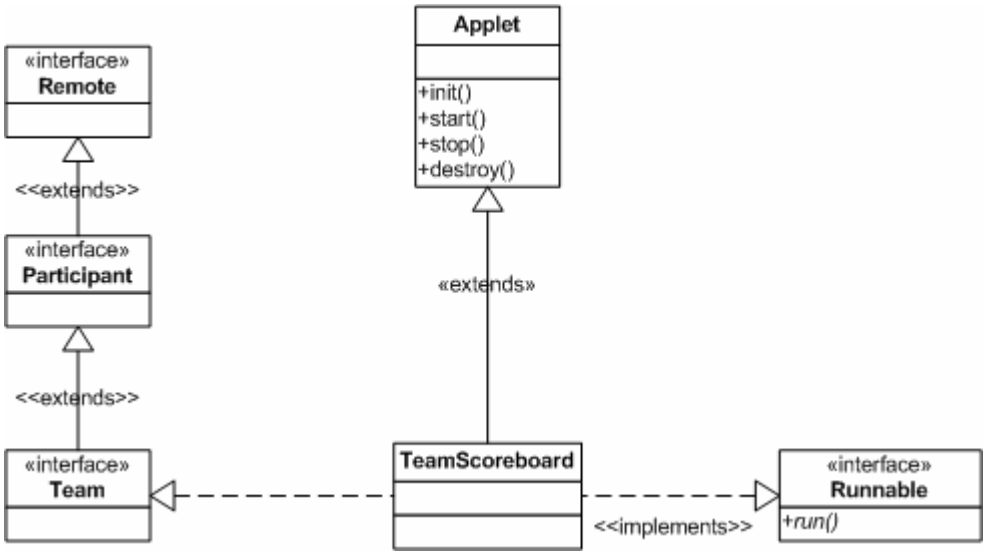
## 8.2 Package: scoreboard

### System Event Queue and AWT Event Dispatch Thread

GUI classes from the AWT library push events (in response to events like Button press, mouse over etc.) into the System Event Queue. An AWT Event thread periodically pops an event from the System Event Queue and delegates it to the appropriate Event handler associated with that GUI component. The AWT Event Dispatch Thread is important for this application because the ACM server notifies a standing change by calling the *enqueueScore* method on the scoreboard which in turn calls the *invokeLater* method. This method queues code execution with the AWT Event Queue. It is a mechanism to request a non event dispatching thread to delegate the code (to be executed) to the event dispatching thread .It is the only way of notifying the GUI of a non GUI event without locking itself. When the Event thread is free it will execute the code (in the run method) that was requested by the *invokeLater* method.

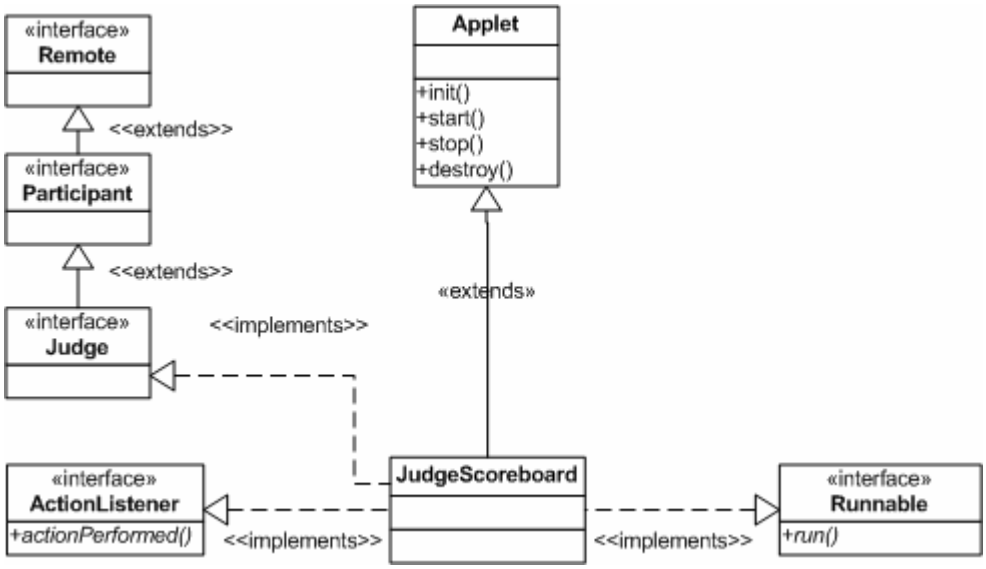
### Class: TeamScoreboard

*TeamScoreboard* is an AWT GUI Applet implementing the *Judge* interface. When a team logs into the contest management system, the Applet is sent to the client from the "codebase" along with the page HTML contents. When the client Applet code is executed within the browser, the client provides its reference with the ACM server. Being a *Remote* implementation, it can be invoked as a distributed call. This mechanism is utilized by the ACM Server to queue messages with the client scoreboard.




### Class: JudgeScoreboard

*JudgeScoreboard* is an AWT GUI Applet. It also implements the *Judge* interface. When a judge logs into the contest management system, the Applet is sent to the client from the "codebase" along with the page HTML contents. When the client Applet code is executed within the browser, the client provides its reference with the ACM server. Being a *Remote* implementation, it can be invoked as a distributed call. This mechanism is utilized by the ACM Server to queue messages with the client scoreboard.




The *JudgeScoreboard* receives all the notifications of score change, a team scoreboard receives. A *JudgeScoreboard* additionally allows a judge to download the team submissions from the server and assign grades.

## 8.3 Package: io

### Class: FileUploadAssistantImpl

A factory implementation provides this class to the caller (ACM Servlet). Teams upload their team submissions to the server. The files are uploaded over HTTP using FORM POST mechanism (as a multi-part message). This class reads the input stream, reconstructs the uploaded files and stores it under the appropriate locations under the server file system. It uses Apache commons library to provide this functionality. The files are stored as a zip archive under the directory structure. Before the archive is stored in the file system, the judge's data file to compare the teams output is also inserted into the archive. This file will assist the judge in performing output comparison prior to grade assignment.

### Class: MailTransport

*TransportFactory* class provides an implementation for transport of messages. Present implementation uses a SMTP and IMAP protocols for clarifications transport. Each

message is mapped to a *Clarification* data type. The *MailTransport* class provides implementation for two methods namely send and receive. The *send* method sends a single clarification using SMTP protocol as the medium of transport to the judges. Before the clarification from the team is sent to the SMTP server, the e-mail's subject is modified with an appropriate string for easy lookup later. The receive method receives all the messages from the users mailbox. It excludes messages from the team's inbox, irrelevant to the contest. The *receive* method filters the e-mail messages based on a string pattern in the e-mail's subject. Recall that this pattern was introduced by the *send* method.

## 8.4 Package: db

### Class: ActivityManager

During the course of the contest, if the scores or standing change the ACM server notifies all the clients of the score change. The task of obtaining the latest scores from the underlying database is performed by this class. It queries the ACTIVITY table and obtains the details for teams that have submitted a solution at any time or have been graded. As soon as the standings change, the "*notifier*" thread invokes the *getStandings* method on the *ActivityManager* class which returns an array of *Score* objects. This array is used to update the scoreboards with the latest standings. To reduce the size of payload (on the array being sent to the scoreboard from the server), the *getStandings* method excludes *Score* objects for problems which the teams are yet to attempt/solve.

### Class: SQLQueryManager

This is a placeholder class consolidating all the SQL statements issued to the database for execution in a central location. SQL statements are stored in the form of strings. This centralization permits code change in case there are any non-standard query requirements from the underlying database.

### Class: ConnectionPool

The current implementation wraps the *DataSource* class. However, the class leaves enough room for any special database or application specific requirements if necessary.

## 8.5 Package: data

### Class: Score

This data structure represents standing per problem per team. Each *Score* class contains the problem number, team name; time elapsed and a *Boolean* flag to indicate if this problem has been graded by a judge. Since this class is sent across the wire it extends the marker *Serializable* interface. As soon as the standings change, the ACM server updates all registered *Scoreboard*'s with updated scores in the form of an array of *Score*'s. The *Scoreboard* iterates through the array and reconstructs the scoreboard.

### Class: Contest

This data structure contains information about the contest itself including details like start time, anticipated end time, status of the contest, e-mail address to forward team clarifications and locale information. Locale information from this class is used to achieve application internationalization. During contest start, this class contains information obtained from the userplus.xml file. Once this information is written to the database, information is retrieved from the database.

## 8.6 Package: config

### Class: ContestDefaults

This is a class implementing all static methods. It provides methods to obtain information about the participating teams, problems, judges, locale and the contest. This class provides the caller's [JSP pages and the Servlet] with meta-data information like list of problems, teams, contest details and status. For optimization, this information is read once during server startup and requires a refresh if any underlying configuration files or database has changed.

## 8.7 Package: jsp

### Class: LeftMenu

This class maintains hyperlink information used to display the web application's left menu. Information required by this class is loaded once during server startup from the LeftMenu.xml configuration file. This XML file is made up of three separate sections (for each role). Each section in the configurable file lists the URL's accessible to a user having a particular role. This mechanism allows addition or removal of hyperlinks/options on the basis of roles without any change in application code.

## 8.8 Package: users

### Class: Team

Each instance of this class represents a single participating team. Product specific user information including e-mail password and name of representing institution is available to the product from this class. Since this information does not change during the duration of the contest, this information is loaded from the userplus.xml file during server startup. *Scoreboard* Applet queries the ACM Servlet for list of *Team*'s to display the scoreboard.

### Class: Judge

Each instance of this class represents a single judge. Contact (name, e-mail address) details and location (room number) related information is available to the product using this class. For performance optimization, this information is loaded from the userplus.xml file once during server startup and maintained until server shutdown. Information available in this class is specifically used to display the "Who, where and what" page only accessible to judges.

# 9. Application: Configuration files

The ACM application can be customized if required with ease. The application relies extensively on information available from configuration files. Some of the files described below are related to the contest and need to be modified with each new contest. The rest need not be touched under normal circumstances, unless there is a compelling reason to modify or customize the application itself.

## 9.1 Internationalization files: MessagesBundle_xx_XX.properties

ACM application supports internationalization, i.e., the application displays the textual content (verbiage and the error messages) based on the region or locale. To support this feature, all the web pages use *MessageBundle* class to obtain locale specific text from property files. File names containing locale specific information have to follow ISO codes for specifying language (ISO 639) and country (ISO 3166). Information maintained in each file is separate for unique combination of a language and country. Based on the locale specified in the appropriate file is read and the entire application appears to be in that language. Support for new languages not provided by default is as simple as creating an additional set of information. It does not require source code recompilation.

For e.g.:-
A code snippet of an internationalized section of the application code would appear as follows:-

```
ResourceBundle messages =
      ResourceBundle.getBundle("MessagesBundle",defaultLocale);
messages.getString("UserName");
```

Depending on the locale set one of the following region specific descriptions would be displayed.

| Country/Language | US/English | Germany/German | France/ French |
|---|---|---|---|
| MessageBundle_xx_XX .properties | *en_US* | *de_DE* | *fr_FR* |
| Sample name-value pairs | [logon.jsp]<br>*UserName = User Name*<br>*Password = Password* | [logon.jsp]<br>*UserName = Benutzer-Name*<br>*Password = Kennwort* | [logon.jsp]<br>*UserName = Nom D'Utilisateur*<br>*Password = Mot de passe* |

## 9.2 Application menu options file: LeftMenu.xml

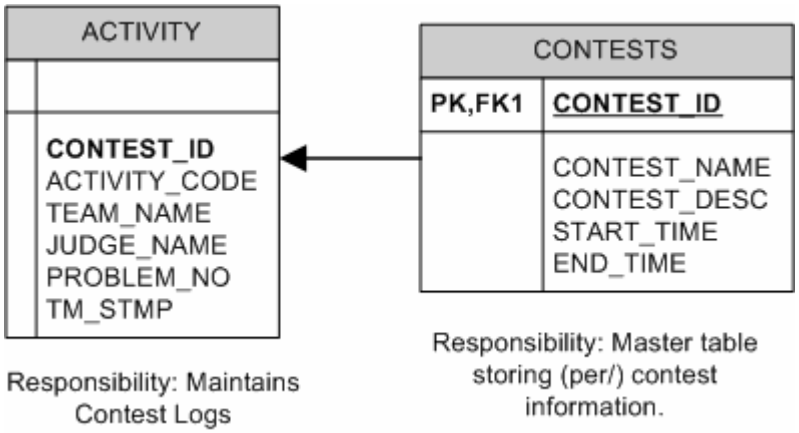
Depending on the user's role, a user is provided different options. Therefore the options available to the team, judge and the administrator are different. Judges are allowed to download individual team submissions. It logically follows that teams should not be allowed to access this feature. This information is maintained in a configurable XML based file names LeftMenu.xml. The actual protection and role to resource mapping is done in the web.xml file.

## 9.3 Contest user details file: userplus.xml

This file is in addition to any application server specific requirements (For e.g. tomcat-users.xml file in case of Tomcat application server). This file maintains details about teams registered for the contest. Before the commencement of a contest, a new section has to be included to this file.

# 10. Logging and server crash recovery

The ACM application records important events during the duration of the contest using activity values. Therefore solution submission and grading activity is recorded along with the timestamp. Each action results in a record in the ACTIVITY table. Important events in the lifetime of a contest like solution submission and grading have been classified by the use of unique activity codes. Clarification requests have been excluded from the logging activity. After a crash, the server reads the records for the contest from the ACTIVITY table and automatically resurrects the scoreboard.




## 10.1 Activity Codes

The ACM Server logs contest information in the ACTIVITY table. Actions performed by a team and a judge are differentiated on the basis of activity codes. These activity codes assist the server to render the latest scoreboard during crash recovery. In event of a change in standing, the ACM server's "*notifier*" thread iterates through the table and reads the records in the ACTIVITY table having activity codes (5 and 8). It creates and array of *Score*'s and updates all the registered scoreboards with the latest standings. The main activity codes are listed.

| Sr. No | Activity | Activity Code |
|---|---|---|
| 1. | Judge determines that the submitted solution is correct. | 5 (ActivityCode.GRADE_CORRECT) |
| 2. | Incorrect solution. | 6 (ActivityCode.GRADE_INCORRECT) |
| 3. | Team uploaded/submitted the solution to the ACM server | 8 (ActivityCode.SUBMIT_SOLUTION) |

# 11. Security

Security is given importance and is implemented at various levels:-

1. Security has been given importance right from the choice of technology and programming language. Java is a type safe language. Java Applet technology via its sand box model only allows limited access to the file system for a code executing from the web browser.
2. Clarifications use mail technologies like IMAP and SMTP. This application is also secure socket ready. Medium of transport for clarifications are over secure channels.
3. Client-side Applet code containing the scoreboard functionality is packaged into separate archives. Even if the file names are known, the user has to have the relevant credentials (via roles) to access it.
4. Client side policy restricts the work an Applet can do within the Applet sand box model.
5. All web pages are protected by the corresponding roles. Therefore, teams cannot access resources available to a judge or an administrator even if the URL's were known somehow.

# 12. Choice of technology

## 12.1 Microsoft based solution

Microsoft provides Active X controls. Active X controls are self-registering Component Object Model (COM) components (reusable code implementing IUnknown interface). Apart from implementing the mandatory IUnknown interface, an ActiveX control can be implement user-defined interfaces to receiving notifications or callbacks from a server. It can also invoke web services to query the latest standings. A web service is a platform neutral way of invoking method or operation. The input parameters, types, return values, and methods/operations are defined in a standard way via the use of WSDL. The WSDL also has location (host and the port details) of the service. Based on the WSDL, the server/callee provides implementation to the web service and the caller uses it to generate the client so as to invoke the web service. The calling code is platform specific. Active X controls could use Web Services to query for the latest scores from a service endpoint. However, the biggest disadvantage of using Microsoft Active X based solution is that it is keyed into M.S. Internet Explorer browser and Windows technology. Moreover, popular browsers like Netscape and Mozilla are not capable of executing client side Active X controls as-is.

## 12.2 Java based solution

Over the years Java technology has matured to provide seamless support for networking, and security. Java compiler's and run time are available for most operating systems today. Recall that Java does require platform specific compiler and run-time. Java is supported by almost all modern browsers including Microsoft Internet Explorer, Netscape and Mozilla. Java is therefore the logical choice for this application. Deploying the Applet as a Java plug-in is a portable way of deploying the Applet.

## Choice of programming language

1. Ease of programming. Java language does not provide pointer operations. When compared with C and C++, that is a huge advantage for a novice programmer.
2. All modern browsers support Sun's JVM. Therefore Java API's and its extensions can be used across browsers. Microsoft's VM (MSJVM) capable of executing proprietary version of Java is being phased out.
3. Java compiler generates a byte code as target. This class files containing the byte code (virtual machine instructions) can be executed on any platform providing a run time environment.
4. Java provides API's for accessing XML files with ease.

No platform or O.S requirements like Microsoft's Active X. This application does not impose any platform or O.S specific feature like Microsoft's Active X technology. This technology is not supported by most of the non-Microsoft/Windows browsers. Therefore this technology was not used for the contest management application.

# 13.1 Other contest management systems

Two contest management products are available, $PC^2$ and Mooshak.

## 13.1 $PC^2$

Developed by California State University, Sacramento, $PC^2$ is a Java based contest management application based on the client-server model. Server and client parts are stand-alone Java applications. Accounts have been categorized as administrators, judges, teams and scoreboard respectively. There is a separate application ("module") corresponding to each account category. Teams use their separate module to submit solutions.

### 13.1.1 Product organization

$PC^2$ is composed of multiple programs or modules. Team module is a tabbed stand alone Java application. Having one team application per machine is recommended but not a necessity. The only requirement being that each team module of a given type be executing on a different directory in the file system. The team module assists the team in submitting the solution, requesting clarifications from a judge and viewing the status of a prior submission or run. Teams can also perform a test run of the solution using their own input file before submitting it for grading with the judges. A Judge module is also a tabbed stand-alone Java application executed separately. A judge uses this module to verify the solutions and update the standings. If the teams close their module in between, they can view the clarifications at a later time. They are persisted. The scoreboard module periodically fetches the latest standings and stores the standings in the local file system as HTML files. More than one version of the scoreboard is provided. Only one scoreboard module is needed to view the latest standings. An administrator can hyperlink the HTML files displaying the standings from a web server to let the contestants view their latest standings. The delay in refreshing the scoreboard is configurable. The administrator is responsible for starting/stopping the contest and the contest clock. Additionally, creation of problems, teams and configuration of *Validators* are also responsibilities of an

administrator. The administrator can view all the prior team runs and current team status (logged in /clarification requested/run made etc.) using the module.

## Comparison

Unlike this product, Team standings in $PC^2$ are not updated automatically. Instead, latest standings are fetched when the team requests for an updated scoreboard.

## Additional features supported by $PC^2$

- $PC^2$ provides an optional facility to grade the submission via the use of *Validators*. *Validators* are implemented akin to UNIX 'diff' command. Additionally it allows a rudimentary comparison white spaces option, one at a time.
- Facility for teams to perform test runs before the actual submission. Code compilation and execution on local machine using the participating team's own data file (provided alongside).

## Disadvantages of $PC^2$

- $PC^2$ requires client-side Java runtime installation and the module containing the application needs to be available on the machine being run.
- Separate applications for teams, judges, administrator and scoreboard.
- Scoreboard is not real-time. In $PC^2$, a separate "board" module updates the scoreboard periodically (based on a configuration parameter). Hyperlinks to this location in the file system (or to a separate location where the files are copied) have to be used to view the scoreboard.

## 13.2 Mooshak

Mooshak is based on two-tiered web model. It is implemented using legacy CGI technology and deployed on open source Apache web server. CGI technology relies on inter-process communication between the web server and the process generating dynamic content for each request using CGI protocol. Tcl scripts deployed under the *cgi-bin* directory generate the HTML content eventually rendered by the web browser.

### 13.2.1 Product organization

Mooshak provides different set of web interfaces (or views) depending on the permission level of the user attempting to access the application. It is very similar in function to the all the other software solutions. Mooshak requires any browser with basic support for Javascript. Three main types of views are available namely contestant, judge and an administrator. A contestant view allows a team to upload submissions and request for clarifications. Teams can also view clarifications requested by other teams using a web interface. A judge view allows judges to answer clarifications, grade submissions and manage printouts. Grading is performed automatically the application, so a judge is not required to grade each submission; however the option of revaluating the submission manually is also provided. An administrative access is required to configure or prepare the software for the contest. An administrator is also responsible for maintaining

problems, adding teams, maintaining language (compile-time and runtime) settings for automatic grading.

### Additional Features

- Automatic grading – Automatic grading is a two step process involving close analysis of source code and program output. Static analysis includes verification of source code compilation and size. If the static test fails, Mooshak halts further analysis. Dynamic analysis involves verification of output, presentation errors and run time errors as a result of program execution. Depending on the results from the analysis a numeric severity value for the team submission is assigned. A correct solution has the least severity.
- Printout management – Mooshak provides commands using which teams can take printouts of the source code and separately work on the solutions.

### Disadvantages of Mooshak

- Mooshak is built using legacy CGI technology. Each web request results in the creation of a new server side process to cater to the request and generate HTML content.

### Advantages of this application over $PC^2$ and Mooshak

- User experience is based on the roles assigned/available to the user. This product does not require separate client installation.
- Scoreboard application is integrated into the web application. No special user/application is needed to view the scoreboard.
- Scoreboard is automatically updated as soon the standings change.
- This product is a 3-tier web application using modern object oriented language.
- This product is scalable. The database and the application server can be scaled up. Scaling up applications that use file for maintaining the logs can be difficult.
- Servlet technology services client requests on separate threads rather than spawning child processes in the case of CGI based applications. It is less CPU intensive.

## 14. Future enhancements

1) This application relies on Java RMI technology to notify client (scoreboard Applet's) of score changes. Since some organizations disallow these ports, a version of this solution that is capable of tunneling requests through a proxy server over HTTP would be beneficial.
2) E-mail password (to read the users inbox) is maintained in clear-text. For added security, it could be maintained in an encrypted format.

## Conclusion

A combination of technologies including Java Servlet, RMI and JDBC technologies can be used to design and build a modular and efficient web based content management application. A web based product-design removes the need for separate application

installation, typical of other contest management systems. Applet-Servlet communication can be used to implement an automatically refreshing scoreboard. Scores are updated via callbacks as soon as the standings change. This mechanism results in fewer messages between scoreboard (on the client-side) and server when compared with periodic page refresh. Networking protocols like TCP/IP, HTTP and SMTP have been used to implement clarification mechanism. Java networking and mail API's provide implementation for the above protocols. Servlet technology avoids the overhead of spawning many processes to service client request.
Even with Java's "*Write once run anywhere*" [4] guarantee, Java Applet code can encounter web browser portability issues. Java plug-in technology could be utilized to minimize its effects. Newer technologies using web services could also be employed to implement an automatically refreshing scoreboard.

## Appendix: Web Application Deployment

The ACM application is deployed as a WAR file (web archive) on the application server.

Steps to run and access the ACM application

- Admin: Download and copy the policy and key store files under $HOME directory for each user logging in.
- Admin: Modify the userplus.xml file to include team, problem and judge information for the current contest.
- Admin: Start the MySQL database ensuring that acmdb instance/database is created.
- Admin: Start the RMI registry service.
- Admin: Start Tomcat application server.

All: ACM application ready for use by all participants.

## Software product requirements

The following software components are required for the ACM application to function.

Application server, database and Ant script requirements are only applicable for the host running the contest application. Client requirements are minimal and most of the popular web browsers qualify as-is.

**Application server: Apache Tomcat**
On Windows platform this amounts to downloading the archive (zip format) and extracting it in a path that is recognized by Windows OS, or explicitly adding the installed package to the PATH environment variable. Tomcat application server if not already existing has to be downloaded and extracted.

**Database: MySQL**
Contest information is eventually maintained in a database. Storage and retrieval of data is relatively easy compared to text file manipulation. SQL queries can also be easily used to generate meaningful reports. This implementation uses MySQL database, however any

database allowing programmatic access via JDBC protocol and supporting ANSI SQL statements can be used.

**Runtime and Compilation: Java**
Java runtime and compiler is required to be installed on the application server hosting the application server.

**Automation scripts: Apache Ant**
This requirement is only for the host running the application server. Ant needs to be used only for compilation of the code from source files and deployment on the application server.

**Client: Java enabled web browser**
Any Sun's Java enabled web browser should suffice as the client. Most popular browsers qualify as-is. Policy file modification might be required to permit socket communication between the scoreboard Applet and Servlet.

**External Java Library Requirements**
ACM application relies on external libraries for database connections, file upload support over HTTP, mail services and unit testing.

| 1. | Library | Description |
|---|---|---|
| 2. | mysql-connector-java-x.jar | JDBC drivers for MySQL database. |
| 3. | activation.jar | JavaBeans TM Activation Framework |
| 4. | mailapi.jar | JavaMail API core classes. |
| 5. | pop3.jar | Provider for POP3 protocol. |
| 6. | smtp.jar | Provider for SMTP protocol. |
| 7. | imap.jar | Provider for IMAP protocol. |
| 8. | commons-fileupload-1.0.jar | Libraries allowing form based file upload using HTTP. |
| 9. | junit.jar | Libraries for JUnit test suite. Only required for unit testing purposes. |
| 10. | ant-jmeter.jar | Libraries for JMeter required by Ant. |
| 11. | catalina-ant.jar | Library containing Apache's implementation of tasks for deployment and un-deployment. This has to be placed in the "lib" directory where a new archive is being made. |

File numbers 1 through 8 inclusive of both are to be placed under $CATALINA _HOME/common/lib directory (of Tomcat application server). Putting the archives at this location allows all the web applications deployed on this server to access the libraries.

File numbers 9 and 10 are only required for unit testing using Ant scripts, so it is placed under $ANT HOME/lib directory. These libraries are not required by the application server.

## Building and Deployment process

Building of the web archive containing the implementation and the deployment onto the application server is achieved using the Ant utility. One build file is sufficient to automate all the tasks.

**Ant Utility**

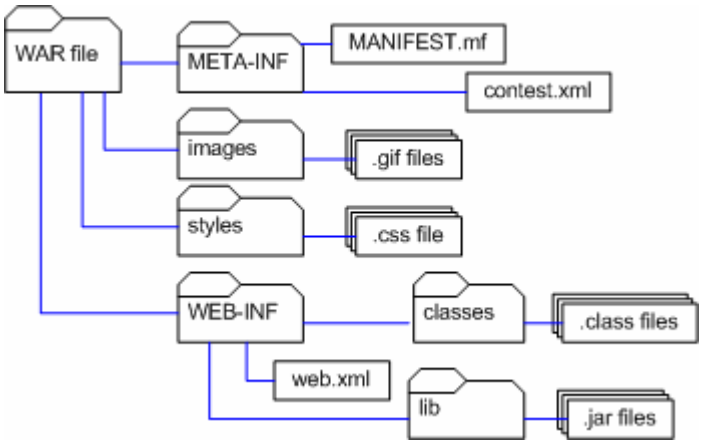
Ant is a Java based build utility without the limitations and problems associated with makefile's. Ant interprets the task information from a XML file (named build.xml by default). It can be used to compile the sources, package the contents into a web archive file and also deploy it. The build.xml file defines the various Ant tasks. The relevant Ant tasks for this application deployment are as follows:-

| Sr. No. | Ant task (Case sensitive) | Description. |
|---|---|---|
| | all | Do the pre-requisite tasks in a given order. Default task for this build file. |
| | clean | Remove the entire work directory. |
| | compile | Compile source files and generate RMI stubs if required. |
| | install | Install the latest version on the application server. |
| | dist | Make a deployable distribution. |
| | javadoc | Generate JavaDoc format source code documentation for all the compiled classes. |
| | list | List installed applications on Servlet container |
| | prepare | Create directories for building the archive. |
| | reload | Reload application on Servlet container |
| | remove | Remove application from the Servlet container |
| | rmic | Generate stubs for *Remote* types. |
| | archive | Make zip file archives containing the scoreboard Applet. |
| | junit | Perform unit test cases using JUnit. |
| | jmeter | Perform stress testing using JMeter. |

## Contents of the web archive file

The Ant utility is used to compile the source files, include the web pages and resources (CSS files and images). The contents are packaged into a single Web Archive or WAR

file. The WAR file is actually a JAR archive confirming to a particular directory structure. The structure and contents of the file are as follows:-




The WAR file is placed in the "webapps" directory of the application server from where the web application can be deployed or removed. Depending on the application server implementation, the WAR file contents might be extracted under this directory during deployment.

| Path | Files/Directories/Packages | Description/Use/Comments. |
|---|---|---|
| / | *.jsp | JSP source files for the web application. |
| /WEB-INF/classes | MessagesBundle_xx_XX. properties | Look up file for internationalization. |
| /WEB-INF/classes | acm/* (The packages are elaborated later in this document under a separate section.) | Java class libraries/packages containing implementation for the web application. |
| /WEB-INF/lib | External jar archive files. | Not being used. |
| /WEB-INF | web.xml | |
| /styles | contest.css | Style sheet for the entire application. |
| /images | *.gif | Images in GIF format used in the web pages. |
| /META-INF | MANIFEST.MF | Manifest file. |
| /META-INF | context.xml | XML file defining context, naming, session and data sources. |

## Source file organization

ACM web application is comprised of Java source files, JSP web pages, images, style sheets, properties files and XML configuration files. Java sources are included in the archive in compiled form. JSP files and other resources like images and style-sheets are also included as-is. They are resolved relative to the WAR file root.

**Java packages**

The contents of the package have been described below:-

1. *config:* Contains classes and interfaces that represent the category of users that can access the ACM application.
2. *data:* Contains data structures that are used across other packages.
3. *db:* Contains classes that will persist the data from the ACM Server to and from the permanent storage (database).
4. *io:* Contains classes that are responsible for the implementation of the underlying transport layer for clarifications.
5. *jsp:* Contains classes that will assist in rendering JSP pages.
6. *scoreboard:* Contains client scoreboard Applet classes and interfaces.
7. *server:* Contains ACM Server interfaces and implementation classes.
8. *users:* Provides classes and interfaces that represent the category of users that can access the ACM application.

**Web page related resources**

The following resources assist in the look-and-feel of the contest application.

- *styles:* This directory contains the style sheets required for the web pages. For the sake of simplicity a single CSS file can change the style for the entire application. The only style sheet file is:-
    - contest.css
- *images:* This directory contains the images required to support the web pages. The following images are present under the images directory:-
    - dir.gif
    - dir_open.gif
    - file.gif
    - icon_bar.gif
    - icon_blank.gif
    - icon_folder_open.gif
    - icon_folder_open_topic.gif
    - red_alert.gif

**Configuration files**

The ACM application maintains important configuration information in XML and properties files. Information for internationalization is maintained in locale-specific properties file. They are as follows:-

1. contest.xml - Main application configuration file.
2. LeftMenu.xml – Configuration file containing HTML link information for all users.
3. userplus.xml – Configuration file containing team and judge. This file needs to be provided/updated in addition to application server specific file containing user information [like tomcat-users.xml]

4. MessagesBundlexxXX.properites - Internationalization related information file. Needs to be modified only if additional languages or locales need to be supported.
5. web.xml - ACM application deployment descriptor file.
6. tomcat-users.xml – Tomcat server configuration file containing user account information [user name, password and role information].
7. server.xml - Tomcat configuration file.
8. .java.policy - Java Policy file, generated by policy tool.
9. .keystore – Key store containing trusted certificate (of the mail server), optional. Required only if secure connection is used.

Files numbers 1 to 4 are product specific files. Tomcat application server requires file numbers 5, 6 and 7. File numbers 8 and 9 are required by Java at run-time.

## References


1. Jos´e Paulo Leal, Fernando Silva: "*Managing programming contests with Mooshak"*. Universidade do Porto http://www.ncc.up.pt/mooshak/
2. California State University at Sacramento: "*Programming Contest Control System (PC$^2$)"*, USA http://www.ecs.csus.edu/pc2/
3. Grady Booch, James Rumbaugh, Ivar Jacobson: "*The Unified Modeling Language User Guide"*. U.S.A: Addison Wesley, 1st ed., 1998
4. Sun Microsystems: "*The Java Servlet API White Paper*" http://java.sun.com/products/Servlet/whitepaper.html
5. Sun Microsystems: "*Java – Internationalization*" http://java.sun.com/docs/books/tutorial/i18n/intro/index.html
6. IETF: "*Form-based File Upload in HTML (RFC 1867)*" http://www.ietf.org/rfc/rfc1867.txt
7. Navin Bhaskar: "*Contest Management Application - Deployment Guide*" http://www.cs.rit.edu/~nxb8951/
8. Navin Bhaskar: "*Contest Management application – Project Proposal*" http://www.cs.rit.edu/~nxb8951/
9. Sun Microsystems: "*Java Remote Method Invocation specifications*" http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmiTOC.html