A Submission and Adjudication System to Track Training and Enhance Performance

G. A. Martin¹, C. D. Gouge², and A. Orooji³
¹Institute for Simulation and Training, University of Central Florida, Orlando, FL, USA
²Oviedo, FL, USA
³Department of Computer Science, University of Central Florida, Orlando, FL, USA

Abstract—Programming competitions continue to grow in number. Whether open to high school, college or the general public, the number of organizations sponsoring them continues to increase. In addition, competitive learning is increasingly used in courses and other activities to foster improved focus on improvement. Such competitions require management tools and many exist. However, we also wish to track individual and team performance across such competitions. We have developed a vision for a suite of tools to provide such a support system. This paper discusses the second such tool: a system for submitting an entry in a real-time competition and having it judged. We review the database structure and how it supports our vision for tracking performance. We then review our implementation including how it was built on top of the Joomla! content management system. Finally, some lessons learned are offered and some thoughts on our next tool are given.

Keywords: Internet Delivery and Applications, Web Interfaces to Databases, Web-based Management Tool, Web-based Training

1 Introduction

Programming competitions have increased in number with new ones appearing regularly. While still the largest and most prestigious, the Association for Computing Machinery's International Collegiate Programming Contest (ICPC) is now joined by Google's CodeJam, TopCoder's TopCoder Open, Facebook Hacker Cup and others. Furthermore, so called “competitive learning” is used as an educational tool now [1][2]. Teachers and professors use these competitions within their classes to drive performance. Indeed, the ICPC now holds a Competitive Learning Institute Workshop each year at its World Finals competition. During this event, speakers discuss how they have brought competitive learning to some area and review systems built to support such purposes.

Our university has an increasingly active student competitive programming team. The team competes in various competitions with the focus being the ICPC regional competitions and, usually, in the ICPC World Finals. However, team members have represented the university in the IEEE SoutheastCon software competition and at Mercer University’s Spring Programming Contest. They have also participated on an individual basis in Google, TopCoder, Facebook and CodeForces competitions.

The team also holds an annual high school programming competition on our university’s main campus. Each year, area high schools come and participate in an ICPC-like competition. While mostly just a fun activity, it also has a background goal of recruiting the best students to our university.

2 Vision

We have built a number of support tools over the years to support the team and its training [3][4][5]. These have ranged from tracking team designation to tracking training problems to allowing program submission. Each tool has succeeded very well in its individual goals. However, the end result was a disjointed set of data across a series of very different tools.

Given these issues, we have created a vision for training and tracking performance of individuals across their development. Our vision has two goals. First, we wish to track performance of members of our competitive programming team; second, we wish to track performance of individuals in classes. Regarding the latter, we teach a class on “problem solving techniques” where students get to apply common computer science algorithms to solving problems. In includes a lecture component and a lab component with the latter run much like a programming contest.

Despite the two goals, the way they fit into our vision is the same. We have planned around the idea of individual and team training performance, coupled with team formation. A number of tools are planned to support this vision. Each member's performance will be tracked including competitions before they qualify for the team (whether in a class such as the problem solving course or as a participant in our high school programming competition) and throughout their career on the team itself. In addition, a team’s performance will also be tracked (including the individuals making up that team) to further support future analysis.

The first of these tools, a competition registration system, was the focus of this paper [6]. We now focus on a
system for submitting entries in a competition and having them judged. This system defines competitions as a set of problems posed. Participants then “solve” the problem and submit their solution to a judge; the judge reviews the submission and replies with one of a fixed number of responses (each of which may be categorized as either a “correct” or “incorrect” response). The system tracks performance through the number of submissions/responses and the time of those submissions. A “scoreboard” is provided for viewing during the competition.

In addition to the participant views, the system also stores all data and allows for later analysis. For example, the submissions can be re-displayed and even offered to other participants as a learning tool (e.g. see what others did). A “total score” is also tracked across multiple competitions. Other aspects of the data are also possible and are only limited by the data that is available.

When creating the submission and adjudication system, we had a number of goals in mind:

- The involvement of teams in each competition should be tracked over time.
- It should support real competitions, team practices and classroom use.
- It should track performance at each competition.

To address these goals, teams must have a single record that is used competition-to-competition. This allows correlating their data together. The people making up the team are also tracked within the team formation so we can detect trends over time as well.

We, previously, chose the Joomla! content management system (CMS) to handle our web sites [7]. We run web sites for the team and for the high school competition as well as an internal site to support the university team's training. Accordingly, we desired that the submission system support both the high school needs, the team’s training needs and the use in our “Problem Solving Techniques” course. Therefore, in order to provide a seamless experience, we developed our approach within the Joomla! framework so that our tools could run directly within our web sites.

3 Design

The Joomla! CMS supports three types of extensions: components, modules and plug-ins. Modules are small units that add content to a web site. For example, a module might display the next five scheduled events for a department. Plug-ins are invisible extensions that provide additional functionality to the web site (such as a new authentication scheme). Components are the most complex type of extension and use the full model-view-controller paradigm to provide an enhanced add-on to the web site.

In order to provide a submission and adjudication extension, we developed it as a component. This allows us the necessary flexibility and power to create a back-end administrative capability as well as front-end user interfaces. An important aspect of any database driven application is the database design itself.

Figure 1 shows the database design we developed. In this design, contests refer to conceptual groupings (such as 2011 ICPC World Finals, or 2011 High School Tournament) while competitions are specific rounds of a contest (for example, the practice session vs. the actual competition itself). Competitions include data about a specific round including the name, start time and end time, as well as information about when what teams are participating. Teams and judges are stored in their own tables and are assigned to a specific contest. While both are stored by username, teams may have multiple usernames specified, allowing a team to use the login information of any of its members. The teams are also connected to our registration system as well [6].

We then store problems posed in the competition in its own table. Each problem has a name, a short name (or filename) and an optional time-limit. For programming
contests, the time limit specifies how long a team’s submission may run on the testing data; however, this can also be used as a submission time limit for other types of competitions (e.g., an essay writing contest). Fields for storing judge testing data as well as a description of each problem (such as in a PDF file) are also provided. Currently, each problem is assigned to a specific competition although we are considering reversing this relationship (let the competition choose the problems from a problem pool as opposed to assigning a problem to a single competition).

Similar to problems, categories are specified to allow teams to ask clarification questions about a problem. Each contains a name and a short name (filename). By separating categories from problems in concept, the system allows broad categories (such as general contest questions, or possibly input-specific questions) and specific questions (such as one category for each problem). While this may cause a slight amount of additional set-up time, the flexibility of decoupling the categories from the problems allow for many uses.

Up to this point, the database table design has focused on the set-up and creation of a competition. However, tables are also necessary for the actual execution of a competition. The system described here includes tables for the submissions themselves, a judgment of a submission from the judges, questions posed in one of the categories, an answer to a question from the judges, and the score.

Obviously, the submission includes which team is submitting it, what problem is being attempted and then the actual submission itself. However, submissions also include fields for tracking the judging (status and judge assignment) as well as a timestamp submission. In addition, a field to track “re-judging” is included to track when judging errors have necessitated a re-check of submissions.

Judgments include the judge’s response, including which judge completed the review. In addition, time is recorded in order for post-analysis of judging speed. Questions and answers follow a similar scheme.

Finally, the score table tracks which team has submitted or solved which problem. It includes data such as number of submissions and timestamps when the problem is solved. In addition, it supports a “public view” that supports hidden scores towards the end of a competition for suspense reasons.

Note that each database table has a “published” field. This is a de facto Joomla! standard that represents whether a given table row should be considered in database queries. Table rows can easily be toggled to “unpublish” its database, meaning that it should not be considered in database queries. By maintaining this standard, it provides the submission and adjudication system the ability to hide data as needed or desired. For example, if a team fails to show for a competition, their table row can be unpublished from the score as opposed to deleted.

4 Implementation

Joomla! is built within PHP [8] and, typically, using MySQL as a database infrastructure [9]. Each added component is linked through a naming convention that includes the component name as well as the function of the component itself. For example, our system is named SPARTA (Submission of Programs for Adjudication and Response with Tournament Administration). Therefore, for a list of submissions within the system we have a controller named SpartaControllerSubmission, a model named SpartaModelSubmission, and a view named SpartaViewSubmission.

The controller is responsible for coordinating the tasks operating on data including editing or triggering the view to display. The model represents data and interacts with the database system (e.g. MySQL), performing queries and updates as necessary. The view handles rendering of the data itself and, in Joomla!, handles producing HTML for display in the web browser. Joomla! provides an administrative interface (referred to as the “back-end”) in addition to the website view itself (referred to as the “front-end”). In order to create a component, the system typically provides both interfaces. The back-end supports administrative functions.

Figure 2 shows the back-end for the component we built. The default view is a control panel that allows the administrator to access all other elements of the component back-end. Tables can be edited or new rows added. In addition, filters are available within each table view to easily select a subset of the table rows. For example, the list of teams can be filtered to those in a specific competition.
Figure 3 shows the sub-panel for the set-up section of the component back-end. Here, the administrator can create contests and competitions, defines teams and judges, and enter problems and categories for specific competitions. For example, Figure 4 shows the corresponding section for viewing (or editing) actual submitted data.

While having the back-end available for data editing is beneficial, developing it also provides a method for exercising the database design before tackling the front-end and its look-and-feel and usability questions. This also allows the database design to be improved as various problems were found. Exploring issues before the front-end was completed saved a large amount of wasted effort.

For example, Figure 5 shows the interface for a team submitting their solution to a problem. The team selects the problem they wish to submit and then attaches their file. The submission is then placed in the submission table, which acts as the queue for the judges.

Once this step is completed, the judge would see the submission appear in a queue view (Figure 6 shows the view with an empty queue). The judge then claims the next submission (an option allows the judge to select a specific submission, if desired).

Once a submission is claimed, the judge is automatically forwarded to the response view. The judge reaches a conclusion as to the response to provide and enters it in this response view. The team is then notified of the judge response.

A similar approach is used for questions and answers. The team uses a text editor built into the view to enter a question and specifies a category. This is then forwarded to the judges. A judge claims the question from the queue; once claimed, an answer editor is presented and the judge submits the answer, which is returned to the team.

Both teams and judges have access to the score (shown in Figure 7). Each team is listed by problems solved with ties broken by a notion of penalty points (currently, we use the approach common in programming competitions that provide a time and accuracy penalty). Each problem is then listed.
with the number of submissions and the time solved (if solved).

A number of interesting performance measures can be tracked given the data. Currently, we have focused on two although have plans for many more. Figure 8 shows one such measure. This display presents the total number of problems solved within a competition (in this case, one entire year of our team’s practices), including the tie-breaking penalty points. In addition, we report the number of problems worked on (submitted) but never solved as well as the total number of problems posed to the teams across all competitions within the contest.

Once team members are entered, the advisor can then enter teams for the competition. Figure 8 shows this interface. The form displayed is dynamically created based upon the maximum number of team members allowed per team. However, this is a maximum and only a single team member must be selected per team.

In addition, we have also provided a view where all the submissions of the teams can be viewed (including whether the submission was judged correct or not). This is only provided after each competition and allows each team (or team member) to review the work of other teams. This works extremely well for self-learning.

The data collected provides much more potential for analysis. For example, we desire to categorize each problem into a type. For example, in a programming competition, a problem might require dynamic programming as a solution technique. By tracking performance across all the problems that might require such a technique, we can begin to see the strengths of teams (and perhaps individuals) become visible.

5 Discussion

During this development, we found that database design was very important and should be given due consideration. While we largely had our database design in place, we did make a few alterations during development. In addition, we are still considering having competitions contain a list of problems rather than a problem being assigned to a single competition.

The SAPRTA system has been in use for just a short while, but we have found that it is easy-to-use and works well in achieving its goals. It allows for team performance to be
tracked and we will increasingly find the data useful as our team progresses.

6 Conclusions

The system described in this paper fulfills the second step in our training performance vision. Previously, we built a registration capability to track team performance. Here, we have a system that allows a competition to be created with problems posed. A team can submit solutions/entries addressing the problems and a judge can review them, providing a response and score. In addition, the performance of teams across all competitions and contests can be tracked and we can detect trends of performance over time.

In the future we desire to extend the SPARTA system further. We wish to create an automatic judging capability to allow competitions to be “extended.” This will allow teams (and team members) to continue to submit solutions to posed problems after the official competition has ended. The focus here is on training and learning, and having such “extended” time allows for that process to continue.

In addition to extending SPARTA itself, we also have a few more tools in our vision. First, we hope to create a problem tracking database. This will collect problems and information about them such as type and difficulty. This will allow us to track performance trends as alluded to above, but also allows us to build problem sets based around specific topics, difficulty levels, etc. In addition, SPARTA can connect into that new database and that will provide us additional performance tracking capabilities.

7 References


