Grading Ecosystem – Spoj0 Integration



Contents

[Introduction 3](#_Toc368060129)

[Spoj0 Design 3](#_Toc368060130)

[Data Model 3](#_Toc368060131)

[High Level Design 3](#_Toc368060132)

[Extensions 3](#_Toc368060133)

# Introduction

The purpose of this document is to outline the integration of the grading ecosystem with the spoj0 grader. Currently Spoj0 supports Java and C/C++ as programming languages being judged. All problems are being graded in ACM style meaning that there is a single result output based on all of the test data (accepted/wrong solution, runtime error during a test execution, time limit during a test execution).

# Spoj0 Design

## 

## Data Model

Spoj0 stores problem sets in a directory tree with the following structure:

sets (top level folder storing contest data)

<set-code> (problem set folder)

[set-info.conf -- for automatic import of a set]

<problem-letter> (problem folder)

test.in (input data)

test.ans (the correct answer)

[solution-<something>.{c,cpp,java}]

[problem-info.conf]

[checker]

However the actual problems are imported inside a MySQL database – the above structure is used only to store sets on the file system along the associated test cases.

## High Level Design

The overall design of the system is pretty simplistic. The entire user interface is several Perl files in the **web** folder of the project. During installation an alias (/spoj0) is added to the apache web server to point to the directory. The following diagram illustrates the high-level architecture of spoj0:



The files from the **/home/spoj0/web/** directory are referenced by Apache Web server. Once a solution to a problem (run) is submitted then it is inserted in MySQL with a status of **‘waiting’**. The **spoj0-daemon.pl** script is started as a UNIX daemon process (using the **start-stop-daemon** utility). It regularly polls the databases for submissions with a status of ‘**waiting’** , marks it as **‘judging’** and executes the **spoj0-grade.pl** script to perform the actual grading of the submission. Тhen it writes the result of the run and logs additional information to the spoj0 logs (**/home/spoj0run/grade.log**; **/home/spoj0run/grade.err**). The standard output stream from **spoj0-grade.pl** is the submission result and the standard error stream contains any errors. The **spoj0-control.pl** script is used to perform the following:

* start - starts the spoj0 deamon (needs root)
* start-here - starts the spoj0-daemon here (not as a daemon)
* stop - stops the spoj0 deamon (needs root)
* kill - kills the deamon if it has blocked by some reason... use with care (needs root)
* rejudge-problem <problem\_id> - marks all submits on the given problem for redjudge except the accepted ones
* rejudge-problem-all <problem\_id> - marks all submits on the given problem for redjudge

including the accepted ones

* rejudge-run <run\_id> - marks the given run for redjudge
* sync-news <set\_code> - synchronizes the news, by adding all new news form the 'news' directory to the database
* import-set <set\_code> - Imports given set into the system. Note that the set should already be in the 'sets' directory
* sync-set <set\_code> - Imports given set or updates its information if already present. Note that the set should already be in the 'sets' directory
* submit <problem\_id> <user\_id> <source\_file> <language> [<about>] - submits given solution

# Extensions

In order to be able to integrate the grading ecosystem with spoj0 we have to provide external interfaces for dealing with the various features. Since we may treat the different items in spoj0 as resources we will provide an overview of the RESTful web services that will extend the features of spoj0. There are number of frameworks that can be used to implement RESTful web services including Mojolicious [1], Catalyst [2], Dancer [3], Dancer2[4], CGI::Application [5] and Jifty [6]. We can also directly use Perl to provide the RESTful web services but for the purpose of clarity, simplicity and maintainability we will be using Dancer. In short the extensions are as follows:

* The **users** table in the spoj0 database is extended with the **admin** column in order to add support for administrative users;
* A new **services.pl** script that contains the implementation of the Dancer RESTful web services that expose the functionality of the system – the web services are described in the user documentation;
* Customized installation and update scripts – located in the Installer project. The installation script adds the mod\_proxy to the apache configuration in order to be able to proxy Dancer from inside Apache (Dancer runs by default on port 3000) along with the rest of spoj0. Since Apache runs by default with the **www-data** user – permissions are also changed and the **www-data** users is added to the spoj0 group in order to able to execute scripts and manage the file system from the Dancer web services (however the web services script is run with a sudo and hence this step is obsolete). The modified update script might be used during development in the form of a deployment scripts for spoj0 – manual changes might be made to script in order to adopt a custom development environment for the system. Original installation and update scripts are left intact until changes are merged into the original SVN repo of spoj0;
* Changes to the spoj0.pm common module and other minor enhancements.

# References

[1] Mojolicious Perl Framework  
<http://mojolicio.us/>

[2] Catalyst Perl MVC framework  
<http://www.catalystframework.org/>

[3] Dancer Perl web framework  
<http://perldancer.org/>

[4] Dancer2 Perl web framework  
<http://search.cpan.org/~sukria/Dancer2-0.10/lib/Dancer2.pm>

[5] CGI::Application MVC framework  
<http://www.cgi-app.org/>

[6] Jifty  
<http://jifty.org/>