# Laboratorio virtual basado en experiencias de concursos de programación

**Área de**
**Investigación, Ciencias y Tecnologías**

Vladimir Costas Jáuregui ,
vladimir@memi.umss.edu.bo
Marcelo Flores Soliz
marcelo@memi.umss.edu.bo
Jorge Orellana Araoz
jorellana@memi.umss.edu.bo

*Resumen*

Los Laboratorios Virtuales son entornos donde el estudiante puede experimentar con actividades inherentes al área de estudio. Estos laboratorios, en el caso de programación, requieren entornos seguros de ejecución así como un balanceo de recursos para completar la ejecución y evaluación de todas las ejecuciones (programas enviados) solicitadas (las que previamente deben ser compiladas para algunos lenguajes de programación)

El proyecto pretende aplicar el uso de los jueces de concursos de programación, sus entornos de ejecución, y evaluación por casos, como herramienta de aprendizaje en entornos virtuales de la programación de computadoras y adquirir experiencia en la construcción de laboratorios virtuales que pueden apoyar en las TIC educativas.

Este laboratorio virtual hace uso de evaluación basada en casos, que permite contrastar el resultado esperado con el obtenido por la solución del estudiante.

Para la construcción del laboratorio virtual se analiza y estudia las características pedagógicas básicas que se requieren para aprender programación basándose en proyectos de programación

El Laboratorio Virtual podrá ser enlazado a un sistema e-Learning por medio del servicio de enlace a herramientas externas de evaluación como el que provee Moodle.

*Palabras Clave***:** Enseñanza de programación, entornos de aprendizaje, evaluación basada en casos

*Abstract*

Virtual Labs are environments where students can experiment with activities inherent to the study area. These laboratories, in the case of programming, requires execution sandbox and resource balancing in order to complete a program execution and evaluation of all requested executions (which previously must be compiled for some programming languages)

The project aims to apply the use of programming contest judges, their execution environments and assessment cases, as a learning tool in virtual computer programming environments to get experience in building virtual laboratories that can be supported by ICT tools for education.

This virtual laboratory uses case-based assessment, which allows us to compare the expected outcome with the solution obtained by the student.

In order to build the virtual laboratory, we analyze and study the basic pedagogical features that are required to learn programming based on programming projects.
The Virtual Laboratory may be linked to an e-Learning system through service linking to external assessment tools such as LTI for Moodle.

*Keywords*: computer programming teaching, learning environments, based assessment cases

## 1 Introducción

Dentro el ámbito de la enseñanza de la programación, se han realizado muchas exploraciones de distinta naturaleza, algunas dirigidas a la necesidad de incrementar la calidad de las experiencias de los estudiantes de programación y otros dirigidas a lidiar con los aspectos tecnológicos propios de esta disciplina.

El trabajo descrito a continuación tiene la motivación de intentar incrementar la satisfacción y promover resultados positivos en la enseñanza de la programación en un medio con adversidades evidentes como la gran cantidad de estudiantes que se inscriben a cursos iniciales de programación y la falta de acceso e implementación de laboratorios de experimentación de prácticas de programación físicos.

## 2. Antecedentes

### 2.1 Concursos de Programación

Los estudiantes de la carrera de Informática de la Universidad Mayor de San Simón participan en la International Collegiate Programming Contest (ICPC) promocionado por la Association for Computing Machinery (ACM) desde el año 2007. A partir del año siguiente, los docentes que sostienenlas actividades estudiantiles en este tipo de eventos, inician competencias locales preparatorias haciendo uso del Sistema de Competencias de Programación utilizado en el ACM-ICPC: PC^2 (Programming Contest Control System)

PC^2 es una aplicación de tipo Cliente-Servidor, donde el servidor se encuentra en un equipo central que consolida una base datos con los resultados enviados, tanto por jueces como equipos competidores. Los clientes son aplicaciones que ejecutan los jueces y equipos competidores.

Cada equipo tiene acceso al sistema mediante su aplicación cliente y puede enviar su solución particular de cualquiera de los problemas de la competencia. La solución consiste en un archivo de programa fuente en el lenguaje previsto para la competencia; posterior al

envío, reciben una notificación del resultado de su respuesta que consiste en seis respuestas posibles:

- Aceptado, el programa enviado ha cumplido satisfactoriamente con todos los casos de entrada
- Respuesta Equivocada, el programa enviado ha fallado con alguna entrada de los casos
- Tiempo Límite Excedido, el programa enviado no ha terminado de consumir todos los casos de entrada y producir respuestas a los mismos en el tiempo límite establecido para el problema
- Error en Formato de salida, el programa enviado resuelve todos los casos de prueba correctamente pero no presenta la salida en el formato esperado
- Error en Tiempo de Ejecución, el programa enviado inesperadamente se ha detenido y ha dejado de ejecutarse
- Error en Tiempo de Compilación, el programa fuente enviado no pudo ser compilado

Adicionalmente los equipos pueden requerir aclaraciones adicionales a los jueces de la competencia, estas aclaraciones se limitan a la redacción de los problemas y posibles errores encontrados en el texto del problema que se provee a los competidores.

Cada juez tiene acceso al sistema y puede evaluar la respuesta enviada por un equipo competidor. Durante la evaluación, el juez no conoce la identidad del remitente de la solución, el juez puede obtener el código fuente, compilarlo y ejecutarlo con cada caso de prueba previsto en el banco de pruebas para el problema. Luego el juez emite su calificación.

PC^2 y otros sistemas de competencias cuentan con un procedimiento de Auto-Juzgar. Auto-Juzgar es un programa especial que hace uso de un sandbox[1], en el que se realiza la compilación y ejecución de las soluciones de los equipos, así mismo controla el tiempo de ejecución, uso de memoria y validación de los resultados aplicando los casos de entrada, contrastando con las respuestas correctas para cada entrada almacenada en el servidor.

Los jueces y equipos deben contar con computadoras con las mismas características, ya que la evaluación de la ejecución debe ser homogénea. Existen también recomendaciones, como

1Ambiente de ejecución de programas que protege el sistema operativo y otros recursos de posibles accesos no autorizados y con capacidad de causar situaciones inesperadas

la de validar la compilación, ejecución y casos de entrada con juez humano y no solamente confiar en la respuesta de la opción Auto-Juzgar.

Sistemas como PC^2 encriptan los datos contenidos en la base de datos del servidor y tienen mecanismos de control de las sesiones de usuarios, así como uso del tráfico de red. Estas consideraciones de seguridad están especificadas en el documento de especificaciones de sistema para control de competencias[2] .

### 2.2 Tipo de Proyectos de Programación

Para las competencias del ACM-ICPC se tiene una especificación del formato de problemas[3] que claramente indica las partes y datos que un problema debe tener para ser incorporado en una competencia. De manera general la especificación contiene: datos de identificación (nombre del problema y límites, tanto de tiempo como de memoria), declaración del problema (texto que da una descripción del problema y aquello que se espera como solución, incluye un ejemplo de entrada y su respectiva salida), datos de validación (ejemplo de la entrada y resultados esperados), validador de formato de entrada, validador de salida, carpeta de soluciones (separadas por tipo de respuesta de evaluación [Aceptado, Respuesta Equivocada, etc.]).

Para el caso de ejercicios de programación que se presentan a estudiantes que están aprendiendo a programar (cursos iniciales en programación) el formato descrito para el ACM-ICPC resulta apropiado, dado que el problema es una descripción clara de la situación que se pretende resolver y presenta las restricciones que se espera para la solución. Además, como están incluidos ejemplos de entrada con sus respectivas salidas correctas; el estudiante es capaz de verificar los cálculos y concebir las restricciones básicas del problema.

Los estudiantes también conocen el formato de entrada esperado y el punto de entrada, así como el de salida (en algunos casos entrada y salida son archivos, en otros casos son la E/S estándar, pudiendo tenerse otros tipos de especificación de acuerdo a las condiciones del curso que el profesor precise)

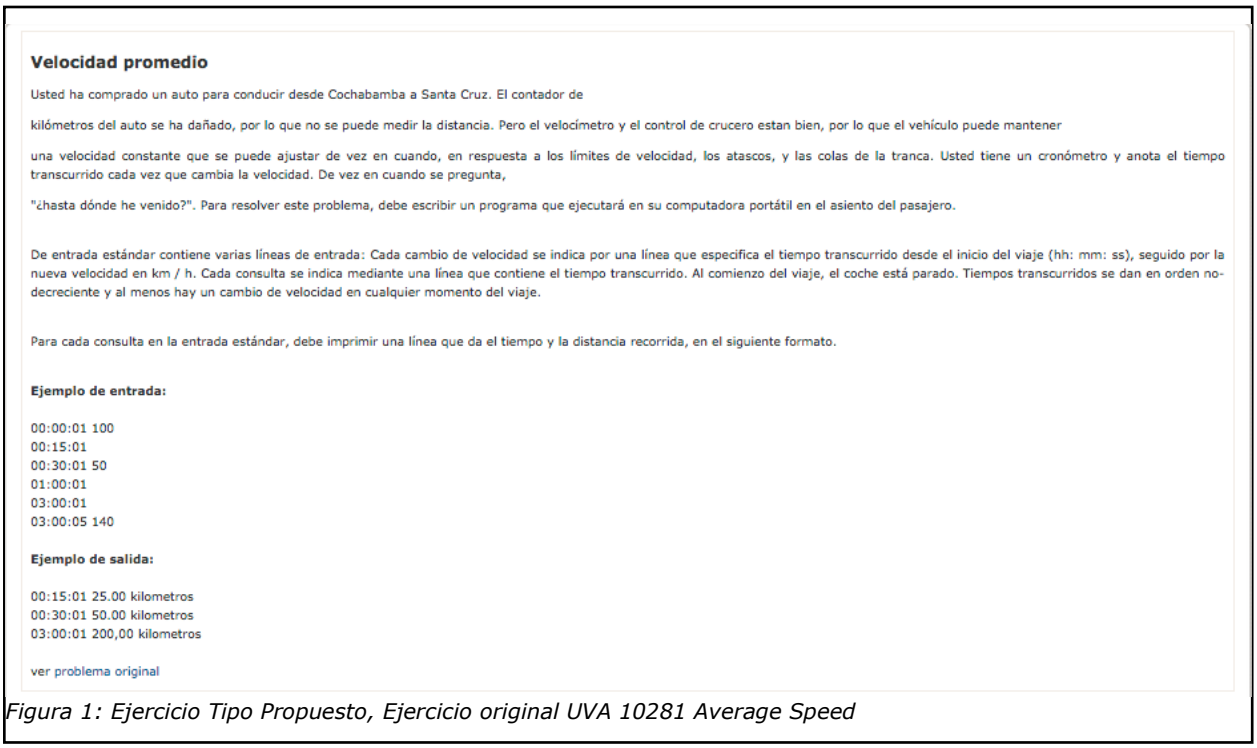
## 3. Experiencia de aplicación en la asignatura Taller de Programación

En los años 2013 y 2014, se experimentó con la aplicación de ejercicios estilo ACM-ICPC, inicialmente presentando ejercicios del repositorio de competencias ACM-ICPC para ser resueltos por los estudiantes; la experimentación incluyó presentar en laboratorio un

2http://www.ecs.csus.edu/pc2/cli/Contest_Control_System_1.0.pdf
3http://www.ecs.csus.edu/pc2/cli/Problem_format_1.0.pdf

ejercicio que debía ser resuelto en noventa minutos y ser entregado como una tarea mediante la plataforma Moodle[4]. Los estudiantes debían programar la solución de acuerdo a la declaración del problema y a los casos ejemplo representados en la Figura 1.

Para este tipo de ejercicios, el docente califica haciendo uso de un caso de prueba generado junto con sus soluciones para el efecto y midiendo el tiempo de ejecución desde consola. La calificación es dada por el porcentaje de casos de prueba que el programa solución coincida con la solución correcta obtenida a priori por el docente. Los casos de prueba se dividen en tres grupos: Aquellos que son situaciones directas de las entradas ejemplo, aquellos que son restricciones especiales del problema y aquellos considerados triviales. No siempre existen los tres grupos de casos prueba.



**Velocidad promedio**

Usted ha comprado un auto para conducir desde Cochabamba a Santa Cruz. El contador de

kilómetros del auto se ha dañado, por lo que no se puede medir la distancia. Pero el velocímetro y el control de crucero estan bien, por lo que el vehículo puede mantener

una velocidad constante que se puede ajustar de vez en cuando, en respuesta a los límites de velocidad, los atascos, y las colas de la tranca. Usted tiene un cronómetro y anota el tiempo transcurrido cada vez que cambia la velocidad. De vez en cuando se pregunta,

"¿hasta dónde he venido?". Para resolver este problema, debe escribir un programa que ejecutará en su computadora portátil en el asiento del pasajero.

De entrada estándar contiene varias líneas de entrada: Cada cambio de velocidad se indica por una línea que especifica el tiempo transcurrido desde el inicio del viaje (hh: mm: ss), seguido por la nueva velocidad en km / h. Cada consulta se indica mediante una línea que contiene el tiempo transcurrido. Al comienzo del viaje, el coche está parado. Tiempos transcurridos se dan en orden no-decreciente y al menos hay un cambio de velocidad en cualquier momento del viaje.

Para cada consulta en la entrada estándar, debe imprimir una línea que da el tiempo y la distancia recorrida, en el siguiente formato.

**Ejemplo de entrada:**

00:00:01 100
00:15:01
00:30:01 50
01:00:01
03:00:01
03:00:05 140

**Ejemplo de salida:**

00:15:01 25.00 kilometros
00:30:01 50.00 kilometros
03:00:01 200,00 kilometros

ver problema original

*Figura 1: Ejercicio Tipo Propuesto, Ejercicio original UVA 10281 Average Speed*

Este tipo de ejercicios ha resultado adecuado para un taller de programación, ya que se trata de un curso práctico en el que los estudiantes conocen aspectos prácticos de la programación, como la definición de variables, las estructuras de control condicionales y de iteración, así como las estructuras de datos básicas.

4 Sistema de gestión de cursos virtuales e híbridos

En el curso, el docente además de evaluar los resultados en base a casos de prueba, observa el código fuente de la solución entregada por el estudiante para evaluar criterios de estándares de codificación y buenas prácticas.

En el problema mostrado en la figura 1, es clara la anotación de que los tiempos son no decrecientes, por lo tanto no hay necesidad de verificar por ellos, el formato de entrada es importante y es necesario diferenciar el patrón de aquellos que son consultas (que obviamente no cuentan con un registro de nueva velocidad) y los que representan un nuevo registro de velocidad.

La solución del problema debe considerar la acumulación del tiempo, con las restricciones que conlleva sumar segundos y minutos.

Los estudiantes normalmente fallan en la acumulación de las unidades de tiempo y en el orden del cómputo de la velocidad antes o después de la actualización del estado de la velocidad. Este tipo de error se refiere a la lógica del estudiante con respecto a la secuencia del algoritmo.

El otro tipo de error se refiere al formato, muchos de ellos tienen la dificultad de descomponer los archivos de entrada para obtener la entrada correcta y descomponer por los separadores que conforman el formato y luego diferenciar una consulta del ajuste de velocidad; por otro lado suelen descuidar el formato de la entrada y normalmente ingresan espacios adicionales. Este último es interpretado como respuesta con errores de formato y en una competencia no se considera como solución, en el caso del curso el docente penaliza el error con una evaluación menor.

El docente realiza la retroalimentación durante los 90 minutos de desarrollo del ejercicio respondiendo a consultas de los estudiantes y validando la respuesta en el momento que el estudiante presenta el ejercicio en laboratorio. Al finalizar los 90 minutos el estudiante recién ingresa su solución en Moodle para su calificación final. El problema de esta modalidad se encuentra en las dificultades del docente para atender concurrentemente a varios estudiantes en laboratorio. En los grupos de laboratorio en que se ha llevado a cabo esta experiencia y se han tenido 7, 13, y 20 estudiantes registrados; ha sido evidente que con el grupo de mayor número la insatisfacción por el tiempo de espera para retroalimentación ha producido un mayor número de estudiantes que consiguen calificaciones menores por retroalimentaciones tardías.

Los tipos de retroalimentación que muestra la experimentación son los siguientes:

- Sugerencias para validar formato de la entrada y descomposición de la misma
- Sugerencias para pensar y considerar en detalle el formato de salida
- Sugerencia de entradas posibles que producen un fallo en la salida (conociendo los casos que fallan, ya sea por la secuencia de instrucciones y/o ejecución del programa con el caso de prueba que el docente posee para evaluación)

En ejercicios más complejos se han tenido que recomendar arreglos para resolver problemas de compilación y excepciones durante la ejecución del programa.
Ha sido también evidente que en ejercicios más complejos la retroalimentación temprana ayuda a conseguir un mejor desempeño de los estudiantes, comprobado cuando los grupos reducidos obtuvieron mejores calificaciones en esta clase de ejercicios.

## 4. Propuesta

### 4.1 Criterios de evaluación propuestos

Un juez de competencia (autómata o humano) entrega solo seis posibles respuestas sin ningún tipo de pista que permita resolver los casos omitidos, el estudiante ni siquiera puede percibir los puntos de error; en competencia es deseable destrezas para enfrentar a esta situación, pero en un ambiente de aprendizaje se espera una evolución con asistencia.

Por la experiencia en los talleres de programación como asistente humano, durante el desarrollo de las soluciones por parte de los estudiantes y su interacción frente a la experiencia de juez de competidores en el ACM-ICPC, se ha podido evidenciar reacciones de los estudiantes y la necesidad de cambios en los criterios de evaluación, así como la calidad y cantidad de retroalimentación requerida.
En una competencia tipo ACM-ICPC un equipo novato tiene grandes dificultades y normalmente intentan usar el procedimiento que les permite preguntar a los jueces para aclaraciones como: ¿por qué no compila?, ¿cuáles son los casos con los que falla el programa?, ¿por qué hay un error de excepción en ejecución? y ¿dónde puede estar el error?. Mientras que en los ejercicios del curso taller, el estudiante podía verificar el mensaje de error de ejecución y hallar por sí solo o con asistencia del docente, la solución a su problema. De igual manera, cuando el programa parece funcionar con los casos de prueba ejemplo que se provee en la declaración del problema, el docente sugiere casos de excepción que el estudiante no consideró para la solución, esto permite al estudiante reflexionar sobre su manera de enfrentar la resolución de los problemas, clasificar las posibilidades de entrada y excepciones en el cómputo en futuros problemas. También le permite una mejor experiencia positiva y menos frustrante al momento de enfrentarse a una dificultad con asistencia cuando está imposibilitado de encontrar mejores opciones (por

cansancio o falta de instrumentos le es imposible atravesar el muro de la perplejidad que le ha impuesto el error sin retroalimentación).

Por las razones mencionadas se propone un criterio de retroalimentación más flexible que el presentado en competencias de programación, que considere:

- Emisión de errores de compilación con rastro completo de los mensajes del compilador.
- Emisión de errores de ejecución con rastro completo de mensajes de excepciones.
- Permitir al estudiante conocer la entrada del caso de prueba que falla para permitirle reflexionar sobre su algoritmo solución y encontrar una manera de incorporar la corrección en el mismo.
- Mostrar al estudiante los elementos faltantes y/o excedentes en su formato de salida para reflexionar sobre la manera que ha estructurado la salida y permitir su rápida corrección

Así mismo se propone una evaluación:

- Sin penalizaciones por soluciones erróneas (en la ACM-ICPC cada solución errónea es penalizada con 20 minutos de tiempo adicionales para el ejercicio)
- Donde, el tiempo de entrega de una solución correcta debe otorgarle al estudiante un puntaje de acuerdo a su posición entre los que consiguieron resolver el ejercicio, este puntaje se puede traducir en medallas u otro de tipo de medida (sin afectar la calificación), además construye una competencia sana entre los estudiantes y los incentiva a mejorar su efectividad, no solo en resolver problemas sino en términos de realizar una reflexión y razonamiento en tiempos más eficientes.
- Donde, la calificación del problema es otorgada con el resultado final del ejercicio (en el caso de estar en un laboratorio con limitaciones de tiempo hay posibilidad de estudiantes que fallen en la entrega), en el caso de tiempo abierto, al menos habrá una penalización por entregas tardías, para favorecer la responsabilidad en el cumplimiento, sin desmerecer una calificación a aquellos que por dificultades diversas han entregado la tarea en tiempos posteriores
- Donde, la calificación no se limite a los que resuelven la totalidad de los casos de prueba, más bien se pese a cada caso de prueba por una clasificación de acuerdo al ejercicio:
    - Casos difíciles, ya sea por excepciones de cómputo o limitaciones de tamaño de tipo de datos tienen un mayor peso.
    - Casos generales, tienen un peso medio.

- Casos básicos o sencillos, tienen un peso intermedio entre los difíciles y generales.
- Errores de compilación deben ser evaluados por debajo de la nota media pero sobre el código realizado (este es necesario con intervención humana).
- Errores de tiempo de ejecución (debiera omitirse los casos de falla) para evaluar por peso, de persistir debe ser evaluado por criterio docente.
- Errores de formato, se advierten en ejercicios iniciales y se penalizan con pesos realmente bajos en posteriores ejercicios.

Los errores de tiempo de ejecución y compilación, cuando son tratados por el docente llevan un criterio de cuán cerca está el programa de convertirse en funcional (por el momento este es un criterio dejado al docente)

Se propone, de manera general que todas las soluciones pueden ser penalizadas (dependiendo del criterio del profesor) cuando:

- No se cumplen estándares de codificación.
- No existe uso adecuado de las estructuras de datos.
- No existe uso adecuado de las estructuras de control.
- El tiempo que tarda el programa en resolver los casos de prueba.
- Hay uso indiscriminado de la memoria.

  *¿Es posible tener un catálogo de criterios que pueden ser dirigidos a penalizar o guiar hacia lo que se quiere de un buen programador?*

**4.2 Arquitectura planteada del laboratorio virtual**

Se describe la arquitectura del Laboratorio Virtual, que incorpora la idea de un sandBox, Comunicación con Moodle y otros elementos arquitecturales

El laboratorio virtual es una aplicación propuesta en el modelo Cliente-Servidor y el ambiente en el que funciona es la Web/Internet. Para detallar la arquitectura se la divide en sus dos componentes generales: Cliente, y Servidor.

En el componente Servidor, se tienen los siguientes elementos:

- *Manager*, Es el componente que maneja las peticiones generales que puede hacer un cliente. Estas peticiones pueden enfocarse en dos grandes ámbitos: la creación de ejercicios-problema con sus respectivos componentes que incluyen los casos de prueba, y el envío de solución a un problema determinado.
- *Problem Manager*, encargado de registrar y mantener la consistencia de la información referida a un problema (título, descripción, casos de prueba, etc.)

- *Solutions Manager*, gestiona las peticiones para ejecución de soluciones enviadas y organiza los resultados obtenidos
- *Queue Control*, en este se encuentra el control de las tareas de ejecución que serán tomadas en el sandbox para evaluar cada resultado obtenido en los casos de prueba contra las soluciones correctas indicadas para el problema.
- *SandBox*, estrechamente relacionado al sistema operativo, es un entorno protegido donde los programas se compilarán y ejecutarán asegurando que el servidor no es afectado por ejecuciones inestables (que en otros casos podrían causar la finalización de ejecución del servidor y/o daños en la ejecución de otros componentes)
- *Data Repository*, es un repositorio mixto (file system/ base de datos) que permite organizar los datos de problemas, y soluciones ejecutadas
- *LTI Interface*, es una interfaz que sigue los estándares para interoperabilidad dentro de aplicaciones orientadas al apoyo del aprendizaje, este elemento permitirá que el Laboratorio Virtual se comunique con sistemas como Moodle y otros que cumplan las convenciones de LTI[5], este estándar es soportado por IMS Global Learning Consortium.

En el componente Cliente se tienen los siguientes elementos:

- *Client Manager*, Encargado de manejar todas las interacciones y solicitudes del cliente
- *Local Data Repository*, es un repositorio local para datos temporales que el cliente requiere en el tiempo que dura la interacción

Bajo este modelo, el cliente se verá empotrado en aplicaciones como Moodle, esta visión es alcanzada mediante el uso de una interfaz que cumple los estándares LTI.

5 Learning Tools Interoperability. http://www.imsglobal.org/lti/index.html

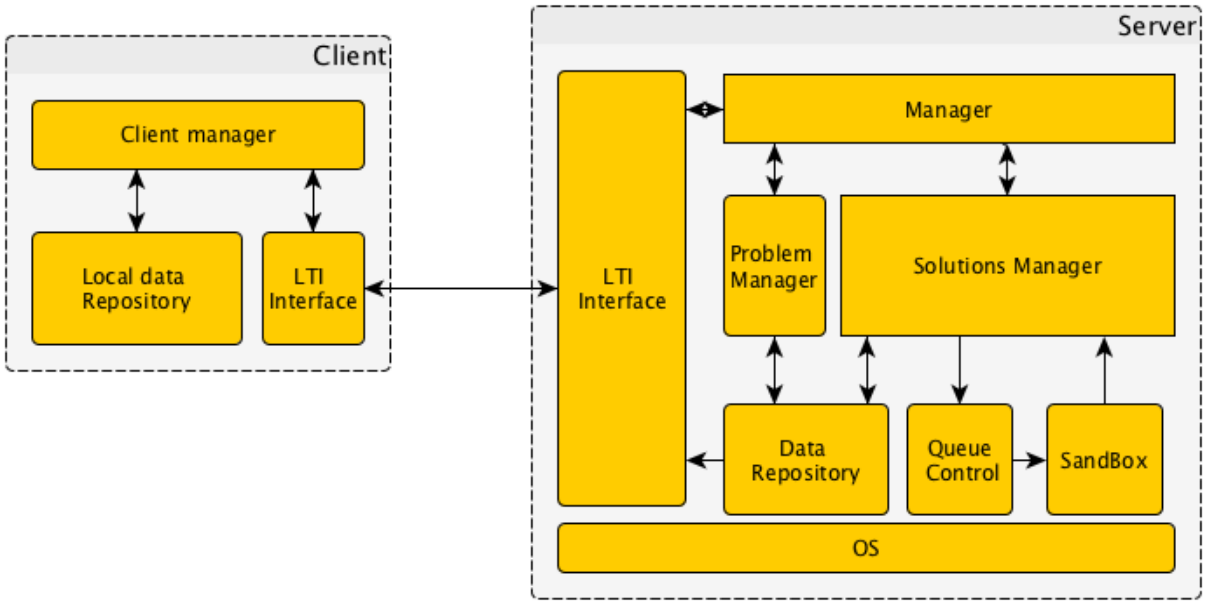



*Figura 2: Arquitectura general propuesta para el Laboratorio Virtual*

Actualmente se tiene implementada la parte básica de un evaluador de casos de prueba para problemas. En el lado Cliente se tiene una representación básica del "Client Manager"; en el lado Servidor se ha implementado en un sistema Linux de forma básica un "Manager" que es capaz de mantener espacios tipo competencias con un conjunto de problemas con su definición completa (es decir el título, descripción, casos de prueba); un "Solutions Manager" que en su estilo más básico controla un SandBox que simplemente ejecuta los programas en un entorno Linux sin considerar aún todos los aspectos de seguridad que le competen a esta clase de componentes.

Las pruebas iniciales, descartando soluciones con código malicioso ó código inestable han logrado resultados satisfactorios; posteriormente se han adicionado controles para límites de tiempo que controlan ejecuciones infinitas. El límite de tiempo establecido va de 1 a 10 segundos, tiempo en el que no se han visto dificultades que pudiesen desestabilizar el equipo servidor.

Los componentes "Manager" tanto en el Cliente como en el Servidor incorporan un editor de programas y capacidades de almacenamiento remoto de los programas del estudiante, de esta manera la herramienta permite un acceso y uso desde cualquier ubicación sin necesidad de contar con un entorno de trabajo previamente instalado y un compilador; el Laboratorio Virtual propuesto proveerá este entorno mediante la Web.

El lenguaje utilizado para esta implementación es JavaScript con el FrameWork Meteor.

Por lo tanto, hasta el momento se tiene la experiencia en el desarrollo de los elementos esenciales de un Sistema de Control de Competencias, al que le falta completar aspectos de seguridad; que deberán enfocarse sobre todo en la implementación del SandBox y algunos criterios en los estándares de construcción de los formatos de datos para almacenar un problema con todas sus partes (además de la descripción del problema, restricciones de tiempo, una solución correcta, y los casos de prueba respectivos). También falta trabajar en aspectos de retroalimentación, así como el editor de programas y el repositorio propio de cada estudiante que contenga sus soluciones en proceso y concluidas. Finalmente queda desarrollar la interfaz LTI para integrarse con otros sistemas de aprendizaje como Moodle.

## 5. Conclusiones

El Laboratorio Virtual es una aplicación que tiene un impacto perceptible cuando la cantidad de estudiantes es muy grande para tener un tutor humano a cómoda disposición, ya que la posibilidad de realizar un apoyo automatizado en aspectos de compilación, entrega de mensajes de excepción con mayor contenido semántico, entrega de cómputos de tiempo además de una decisión lógica acerca del cumplimiento del objetivo del problema, son elementos que favorecen el aprendizaje de la programación.

El Laboratorio Virtual pretende entregar una herramienta que es accesible desde la web con un editor integrado, y un compilador disponible, cerrando las dificultades de acceso a laboratorios físicos al que los estudiantes deban acceder, y permitiendo un acceso asíncrono al entorno, como espacio de práctica y retroalimentación del progreso de su solución frente a los casos de pruebas.

Es probable que el aprendizaje guiado en problemas y en la ejercitación de solución de problemas para el área de programación esté un poco alejado de la realidad, puesto que en la práctica profesional real de la programación, los problemas a solucionar no tienen una definición completa y cabal, entendible, ni enunciable sin ambigüedades. La mayoría de los problemas de programación son rescatados de fuentes que originan/sufren los problemas de forma que no siempre conocen aspectos cabales de E/S, no siempre se tiene una descripción precisa y menos matemática/lógica del problema. Estas características muy normales de los problemas de programación en ámbitos de desenvolvimiento profesional son una motivación para pensar que la orientación de un aprendizaje en las descripciones de los problemas no es muy apropiado y la motivación va en sentido de pensar en un aprendizaje guiado en las descripciones de pruebas de validación, es decir; aceptar que las descripciones de los problemas no serán la fuente principal del desarrollo del trabajo, pues este puede estar pobremente descrito o carente de completitud, sin embargo la base del

desarrollo de la solución debe estar en las descripciones de las pruebas de aceptación que debe cumplir el programa. Esto por ahora es solo una motivación, y se espera que la versión futura a construir del Laboratorio Virtual contenga adicionalmente esta característica.

Otros problemas comunes de experimentar con este tipo de proyectos son en su mayoría debidos a aspectos administrativos propios de las instituciones educativas, como ser: aspectos de definición curricular, manejo distinto de tiempos, objetivos y la diferenciación del programa de clases teóricas/prácticas, y de la exigencia de la presencia de los estudiantes en determinadas actividades (como las prácticas). Estas dificultades deben ser resueltas con la visión de una educación abierta y disponible en el espacio/tiempo de acuerdo al avance de uso de TICs en el ámbito educativo, situación que indefectiblemente alcanzará a toda institución educativa.

**Referencias bibliográficas**


GOMES A., MENDES A.J. (2007), Learning to Programming - difficulties and solutions, International Conference on Engineering Education – ICEE 2007 IMS Learning Tools

ICPC Collaborative Learning Institute (2014), http://www.ecs.csus.edu/pc2/cli/index.html

Interoperability (2014),http://www.imsglobal.org/iti/index.htm

Moodle (2014),http://docs.moodle.org/27/en/external:tool:sttings NEVE P., HUNTER G., L.ivingstone D. (s.d), NoobLab: An Intelligent Learning Enviroment for Teaching Programming

NEVE P., LIVINGSTONE D. (2012), Developing virtual programming laboratories to inform the pedagogy of programming, Proceedings of the HEA STEM Learning and Teaching Conference

STEVENSON D.E. (2001), Problem-Based Learning Applied to Programming Instruction, Submitted to ACM SIGCSE 2001



**Vladimir Costas**

Es docente a tiempo completo en la Universidad Mayor de San Simón, desenvuelve su actividad docente desde el Centro de Mejoramiento de la Enseñanza de la Matemática y de la Informática (MEMI). En los últimos años ha estado dedicado a la participación, como juez local, en las competencias de programación a nivel universitario (AMC-ICPC) y al apoyo en el entrenamiento de los estudiantes universitarios que participan en el ACM-ICPC; así mismo es miembro del Comité Científico Departamental Cochabamba de las Olimpiadas de Informática en Bolivia (Olimpiada en la que participan estudiantes de secundaria en el área de Informática). Las áreas de interés en las que trabaja son: Teoría de Ciencias de la Computación, Algoritmos, Enseñanza de la Programación en Ciencias de la Computación, Programación Web.

**Marcelo Flores**

Marcelo Flores es Investigador del Centro MEMI y docente de las carreras de Ingeniería Informática e Ingeniería de Sistemas de la UMSS. Actualmente investiga aspectos de Idiomas en el desarrollo de bases de datos, además de alternativas para tecnologías ORM.

**Jorge Orellana**

Ingeniero de Sistemas. Consultor para USAID, Técnico en Informática para el Instituto Boliviano de Tecnología Agropecuaria, Supervisor del Departamento de Sistemas de la Empresa Minera Inti Raymi (Newmont Company), Contratista en la Empresa Nacional de Ferrocarriles, Corporación Minera de Bolivia y otras empresas del medio. Profesor universitario desde hace más de 20 años en diferentes Universidades de Bolivia. Docente-Investigador en el Centro MEMI - UMSS (Mejoramiento de la Enseñanza de la Matemática e Informática-Universidad Mayor de San Simón) en proyectos de investigación y extensión de esta unidad.

Subir