Ensayos

Un framework para desarrollar sistemas distribuidos aplicados a enseñanza y aprendizaje, usando Erlang como componente principal.

Resumen

La educación tradicional es concurrente por naturaleza. Estudiantes, profesores y herramientas que la soportan interactúan entre sí para ayudar en los procesos de enseñanza aprendizaje. Hemos aplicado Erlang para tratar la parte concurrente de un sistema distribuido que da soporte las tareas de enseñanza aprendizaje. Como un efecto secundario hay beneficio del estilo de programación funcional de Erlang para que coincida con una metodología de programación extrema. Para formalizar la descripción de nuestro sistema hemos realizado en CCS y UML diseños que complementan la documentación del sistema. Junto con nuestro desarrollo, también se han resuelto algunos problemas con respecto a trabajo colaborativo y a asuntos educativos.

Abstract

By its very nature, traditional education is concurrent. Students, teachers and learning tools work together in the educational process. We have used Erlang to treat the concurrent part of a distributed system which supports teaching and learning tasks. As a side effect, we have benefited from Erlang's functional programming style that coincides with an extreme programming methodology. To formalize our system's description, we have developed designs in CCS and UML that complement the system's documentation. Together with our development, we have also resolved certain problems concerning collaborative work and educational issues.

Résumé

L'éducation traditionnelle est concurrente de nature. Etudiants, professeurs et outils interagissent pour aider dans les processus d'apprentissage/enseignement. Nous avons appliqué Erlang pour traiter la partie concurrente d'un système distribué supportant les tâches d'enseignement- apprentissage. Comme effet secondaire nous avons bénéficié d'un style de programmation fonctionnel d'Erlang pour qu'il coïncide avec une méthode de programmation extrême. Pour formaliser notre système, nous avons réalisé en CCS et UML des dessins qui complètent la documentation du système. Avec notre développement, nous avons également résolu certains problèmes relatifs au travail collaboratif et aux thèmes éducatifs.

* Cortés Hugo, Cruz Omar, García Mónica, Hernández Jorge, Hernández Manuel, Pérez-Cordoba Esperanza y Ramos Erik.

Palabras clave:

Concurrencia, Erlang, Paso de Mensajes, CCS, UML, Taxonomía de Bloom

1. Introducción

El nuestro, es un mundo concurrente: muchos procesos están activos, al mismo tiempo y en distintos lugares. Hemos aplicado Erlang a la parte concurrente del diseño e implementación de un sistema distribuido, siguiendo los lineamientos de la mezcla de lenguajes como en (Armstrong 2002), y en el espíritu de desarrollo de software como en (Cabrero, Abalde, Varela y Castro 2003). Este sistema distribuido tiene por objetivo apoyar algunas tareas comunes de enseñanza y aprendizaje. Después de algunos intentos fallidos en la implementación de un sistema distribuido orientado a la enseñanza-aprendizaje (Cortés, García, Hernández, Hernández, Pérez-Cordoba y Ramos 2009) percibimos que quizás unas de las razones de nuestros fracasos pudo ser el lenguaje de programación utilizado. Estábamos utilizando lenguajes de programación muy poderosos, pero también muy generales. Algunos detalles de implementación resultaron ser muy laboriosos y particularmente difíciles. Junto con

* Universidad Tecnológica de la Mixteca

nuestros intentos, hemos observado que el diseño e implementación de programas concurrentes es diferente al de sus homólogos secuenciales. En primer lugar, la programación concurrente es complicada, porque implica coordinación entre todas las partes involucradas. En segundo lugar, necesitamos el apoyo de nuestras herramientas, incluyendo (a) una filosofía adecuada de desarrollo de software, (b) un lenguaje de especificación adecuado, y (c) un lenguaje de programación adecuado. Para que, finalmente se logre integrar muchas partes de nuestro sistema con un punto de vista uniforme.

Necesitábamos un lenguaje de programación orientado a la concurrencia. Este lenguaje de programación es Erlang, que tiene una comunicación vía paso de mensajes, un lenguaje de programación funcional, mecanismos de tolerancia a fallas y una gestión de procesos automáticos. Erlang fue lo suficientemente poderoso como para en él implementar nuestro sistema en un corto periodo, facilitando los detalles de implementación mencionados, lo que nos permitió avanzar rápidamente en el proyecto. En nuestro caso, el problema consistía en diseñar e implementar un sistema distribuido para apoyar los procesos concurrentes de enseñanza y aprendizaje. Con esta propuesta, hemos querido apoyar el ambiente del aula tradicional a través de una herramienta de enseñanza aprendizaje asistida por computadora. Los usuarios (desde profesores a estudiantes) involucrados, interactúan con el sistema usando computadoras y dispositivos móviles, facilitando en gran medida sus tareas cotidianas.

Para implementar nuestro sistema distribuido experimentamos con un método de desarrollo de software dentro de un marco, mediante la adopción de las partes más útiles de la programación extrema (Kent 1999), la programación declarativa (Hughes 1989; MacLennan 1990; Bird 1998) y la programación literaria (Knuth 1992) (podemos considerar este documento como una técnica de programación literaria).

Cuando usamos el método de la programación extrema intentamos producir un código mediante la participación de los programadores y clientes desde el comienzo del proyecto. Como el código Erlang es tan claro que puede ser visto como una “especificación ejecutable”, la programación extrema actuó como un medio para llevar rápidamente a cabo algunos prototipos funcionales. En efecto, en las técnicas tradicionales un prototipo es considerado como un artefacto de desarrollo desechable; en contraste, en Erlang se pudo partir de este prototipo, tomándolo como base, para efectuar mejoras graduales, incorporando así características de tolerancia a fallas, mejoras en la eficiencia, y en general, enriqueciéndolo con nuevas características y funcionalidades.

La información del prototipo puede ser también documentada y explicada, y, tomando ventaja de las partes declarativas de Erlang, el código del prototipo puede ser modificado siguiendo métodos estrictos: abstracción de tipo(Guttag 1986), interpretación abstracta(Cousot 1992) y transformación de programas(Burstall y Darlington 1977; Pettorossi y Proietti 1996). Cuando sea necesario, trabajando con un concepto de módulo también ayuda a ocultar los detalles irrelevantes para el resto del equipo.

Hemos diseñado una arquitectura distribuida básica. Además de que hemos propuesto algunos comandos, (resultado de los eventos realizados por los actores) y hemos detectado esos mismos actores que los ejecutan, Por lo tanto, fue necesario, especificar nuestro sistema desde un par de puntos de vista complementarios: por una parte, se modeló el flujo de acción de las actividades llevadas a cabo por cada actor usando CCS (Cálculo de Sistema de Comunicaciones de Milner, Calculus of Communicating Systems), aunque por efectos de espacio sólo presentamos un caso particular de estudio . Por otro lado, modelamos el comportamiento externo de cada actor mediante el modelo de caso de uso, técnica que pertenece a UML (Lenguaje de Modelado Unificado, Unified Modeling Language), particularmente, en el momento de planificar el soporte de manejo de excepciones.

Una característica de nuestro sistema es que tiene un determinado recurso compartido: los test de evaluación para estudiantes de licenciatura. Para diseñar los exámenes (tests) hemos usado la taxonomía de Bloom. Cada examen tiene asociado un nivel particular siguiendo esta taxonomía. Después, hemos formalizado un conjunto de comandos para hacer la interfaz con estos exámenes. Estas interfaces involucran tanto a los profesores como a estudiantes. Nos hemos decidido por la automatización en la mayor parte de las interfaces, incluidas la aplicación y diseño de exámenes.

Hemos desarrollado algunas técnicas para tratar con recursos compartidos: (a) una planificación para hacer o modificar cada uno de los recursos; (b) una partición de cada recurso para tratar con cada miembro de la partición independientemente; y (c), un etiquetado para poner a cada recurso una etiqueta para el seguimiento del historial del recurso.

Hemos organizado este artículo como sigue: En la Sección 2 describimos Erlang y proponemos una arquitectura distribuida; En la Sección 3 sugerimos usar programación extrema junto con lenguajes funcionales; en la Sección 4 adaptamos nuestro modelo distribuido a un sistema de enseñanza aprendizaje; en las Secciones 5 y 6 damos una especificación formal del comportamiento del sistema; en las Secciones 7 y 8 nos ocupamos de los tests y su tratamiento colaborativo. Por último, las conclusiones son presentadas en la Sección 9.

2. Erlang y nuestra arquitectura básica

En esta sección damos algunos conceptos básicos y características sobre Erlang. Para más información acerca de Erlang consultar en Internet[1] o en el libro(Armstrong 2007). También describimos en esta sección algunos aspectos básicos de nuestra arquitectura.

2.1. Erlang y concurrencia

Erlang es un lenguaje de programación funcional orientado a la programación de sistemas concurrentes; en la parte de concurrencia, Erlang tiene las siguientes características: Un nodo en Erlang es una activación de un sistema Erlang. Un proceso es un programa que se ejecuta en un sistema Erlang. Un sistema distribuido consiste de muchos nodos y procesos. En un sistema distribuido basado en paso de mensajes cada comunicación entre procesos se basa en mensajes (datos). En Erlang, podemos mandar o recibir cualquier término válido (incluso un proceso en sí). Este lenguaje de programación tiene otras características distribuidas: mecanismos de tolerancia a fallas, capacidad industrial (muchos procesos pueden estar activos sin una disminución significativa en la eficiencia del sistema en su totalidad), y un mecanismo de comunicación asíncrona. Erlang también tiene muchas bibliotecas o paquetes para resolver problemas específicos (desde hacer interfaces con otros lenguajes, incluido Java), es de código libre y excluye un explícito tratamiento de recursos compartidos.

En la parte de programación funcional, Erlang sigue un modelo de evaluación rápido. Erlang también tiene funciones de orden superior, listas compactas, y un sistema de módulos; estas características colocan a Erlang dentro de la corriente de lenguajes de programación funcionales más maduros, tales como: Haskell[2] o Clean[3].

Un sistema distribuido debe ser diseñado para ser tolerante a fallas mediante el uso de procesos vinculados y supervisados, un sistema puede tener tolerancia a fallas con un cierto costo elevado de cómputo en el sistema global.

FIGURA 1: CLIENTES Y SERVIDOR EN ESTADO DE ESPERA

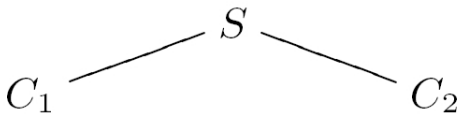

FIGURA 2: CLIENTES CONECTADOS AL SERVIDO EN EL TIEMPO T, CON T>T0

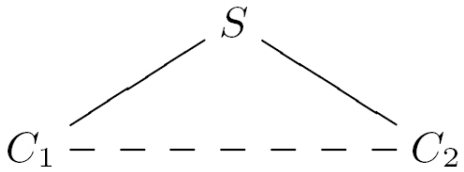

FIGURA 3: CLIENTES COMUNICÁNDOSE CON CLIENTES

2.2. Arquitectura básica

Para describir nuestra arquitectura básica, fue necesaria la siguiente terminología: Un servidor (server) es un proceso en ejecución desde un nodo. Un cliente (client) es un proceso que requiere los servicios o recursos del servidor. Un sistema distribuido centralizado es un sistema distribuido que tiene muchos clientes pero sólo un servidor. Una solicitud (request) es un mensaje que va desde el nodo al servidor. Una respuesta (answer) es

1. www.erlang.org

2. www.haskell.org
3. www.cs.kun.nl/~clean

un mensaje que va desde el servidor al nodo. La recepción de mensajes es administrada por un buzón (mailbox). Un buzón recibe mensajes. La recepción de mensajes es asíncrona; el emisor no esperará la respuesta del receptor.

Una configuración en un sistema distribuido es un diagrama que muestra los nodos y las conexiones en un momento dado. Este diagrama describe los componentes esenciales de un sistema distribuido así como las conexiones existentes actuales. Estas conexiones se muestran a través de líneas continuas conectando nodos (en el sentido de teoría de grafos). Estas conexiones son llamadas directas. En la configuración mostrada en la Figura 3, los clientes son capaces de comunicarse entre sí de forma indirecta. Estos tipos de canales de comunicación está indicada por líneas punteadas. El servidor S, en este caso, trabaja como un receptor y transmisor de mensajes universal. Otras tareas pueden ser implementadas directamente en el servidor, pero este es el esquema básico que permitió iniciar nuestra arquitectura.

Sea S un servidor, suponiendo la existencia de algunos clientes potenciales, C_1, C_2 (ver Figura 1), el servidor S se activa en el tiempo t_0; a partir del tiempo t_0, los clientes pueden abrir un canal de comunicación con el servidor (ver Figura 2). Una vez activado, el servidor S es capaz de aceptar o rechazar las solicitudes formuladas por los clientes. La solicitud más básica formulada por un cliente al servidor es conectar (connect). La respuesta más básica formulada por el servidor a una petición de conexión formulada por un cliente es aceptar o rechazar. Una petición de conexión se hace siempre antes de aceptar o rechazar una respuesta. De hecho, suponemos que la secuencia propuesta por las Figuras 1 y 2 es antes de formular peticiones y respuestas. Habiendo aceptado el servidor una conexión con un cliente, el servidor entonces ha abierto un canal de comunicación.

Supongamos que el servidor S acepta las peticiones formuladas por los nodos C_1 y C_2. El sistema distribuido actual está representado en la Figura 2. En esta figura, los clientes han sido aceptados por el servidor. Los nodos ejecutan los procesos del cliente, que son aceptados o rechazados por el proceso o procesos ejecutados desde el servidor.

3. Programación extrema y declarativa

Esta sección muestra las ventajas de aplicar la programación extrema en combinación con la programación declarativa a un sistema como el nuestro.

Hemos usado la metodología de la programación extrema debido a:

1. Nuestros clientes son nuestros profesores y estudiantes (algunas veces somos profesores y estudiantes al mismo tiempo);
2. Queremos trabajar en equipo;
3. Nuestro equipo de desarrollo es pequeño y no queremos ser especialistas en una parte concreta del sistema;
4. Nuestra aplicación es de tamaño medio;
5. No queremos solamente revisar el código de los demás (que es de gran valor), si no que también revisar cada una de las decisiones del diseño a nivel de especificación;
6. Queremos compartir ideas junto con la etapa de implementación;
7. Creemos que una buena especificación debe ser una buena guía del proceso del desarrollo global, pero estamos dispuestos a cambiar razonablemente la especificación cuando sea requerido;
8. Queremos reforzar la programación extrema con métodos formales, como refinamiento de pasos, la refactorización o la transformación de programas.
9. Consideramos que los primeros prototipos como programas simples pueden ser mejorados y desarrollados para obtener una implementación final, eficiente y correcta;
10. Tenemos que mejorar la productividad a través de ejemplos concretos de aplicaciones.

Para complementar las practicas de programación extrema, tenemos que escribir la documentación en todas las etapas del desarrollo de nuestra aplicación. Esta parte se inspira en los conceptos de la programación literaria. Por otra parte, creemos que la programación extrema es una metodología ligera muy útil, pero esta metodología debe estar acompañada de otras practicas útiles de programación; particularmente, los métodos formales siguiendo las directrices que se mencionan en (Bauer, Broy, Gnatz,

Hesse, Krieg-Brückner, Partsch, Pepper y Wössner 1978; Partsch 1990), y técnicas de transformación (Burstall y Darlington 1977), modelos matemáticos de concurrencia (Milner 1989), y lenguajes de modelado (Booch y Rumbaugh 1998). Y por encima de todo, la elección de un lenguaje de programación adecuado: Erlang. En este trabajo, de hecho, tratamos de usar una colección ecléctica de metodologías que trabajen juntas, teniendo como pivote a Erlang. Por otra parte, la programación declarativa debe ser un buen complemento para formular la creación rápida de prototipos y la transformación de programas.

En resumen, estamos convencidos de que la programación extrema es una metodología adecuada para el desarrollo de software, si podemos elegir casos en donde aplicarla y las herramientas adecuadas o métodos que la apoyen.

4. Ejemplo básico de nuestro modelo

Para ser prácticos, nuestro modelo debe incorporar algunas funciones para interactuar con los usuarios. De hecho, es posible tener un sistema distribuido sin ningún tipo de interacción con humanos a partir de un tiempo t. Sin embargo, este no es nuestro caso: nuestro sistema está diseñado para ser altamente interactivo con los usuarios desde profesores hasta estudiantes.

Las interfaces modernas están basadas en interfaces gráficas de usuarios (GUIs). Erlang incorpora una GUI básica para hacer la interfaz con humanos. Este paquete se denomina graphics. Para seguir una arquitectura uniforme y disciplinada, hemos pensamos en las interfaces como procesos. Estos procesos facilitan la vida humana, aunque las interfaces deben ser diseñadas, desarrolladas y mantenidas al margen del sistema fundamental. Para lograr esta meta, preferimos crear canales de comunicación virtuales entre una GUI y un cliente, o entre una GUI y un servidor. Cada comunicación debe estar obligada a utilizar con eficiencia los canales directos abiertos. Ver Figura 4. Esta "separación de preocupaciones" (Dijsktra 76) es una técnica útil omnipresente en el desarrollo de programas. En particular, podemos extender nuestra arquitectura incluyendo los usuarios finales vinculados al sistema a través de dispositivos móviles.

Siguiendo con el diseño, podemos tener un sistema como el de la Figura 4. Después de la activación de los canales virtuales, la configuración dada en la Figura 4 se verá como la de la Figura 5.

Las interfaces gráficas pueden también ser consideradas como interfaces de usuario final. Estas interfaces de usuario final puede ser localizadas en un dispositivo móvil. De hecho podemos soportar sistemas híbridos, en donde hemos designado a Erlang como el administrador de procesos, pero con interfaces implementadas en otro lenguaje. Java y su máquina virtual son buenos candidatos, considerando que la implementación de las máquinas virtuales en dispositivos móviles es cada vez más común en estos días. En este caso, el flujo de mensajes entre un nodo Erlang y una máquina virtual de java podría llevarse a cabo a través de técnicas de tuberías (pipeline) o a través de flujo de datos binarios. Hemos hecho experimentos usando tecnología Bluetooth, de tal forma que los estudiantes utilicen dispositivos móviles como terminales tontas. Las GUIs están "encapsuladas" dentro de un nodo final, o las GUIs pueden ser ejecutadas desde dispositivos móviles usando un sistema hibrido: una máquina virtual como J2ME puede ser ejecutada desde cada dispositivo y la comunicación puede ser reducida al

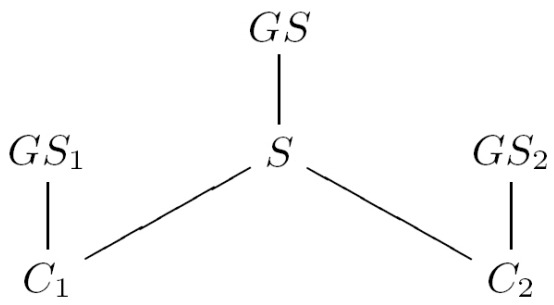

FIGURA 4: ACTIVACIÓN DE LOS PROCESOS GRÁFICOS

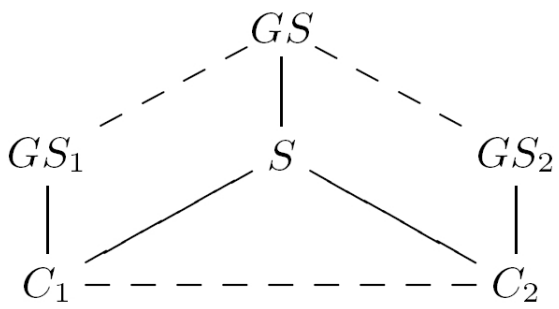

FIGURA 5: CANALES VIRTUALES

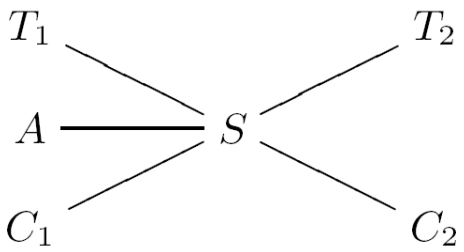

FIGURA 6: UNA CLASIFICACIÓN DE USUARIOS

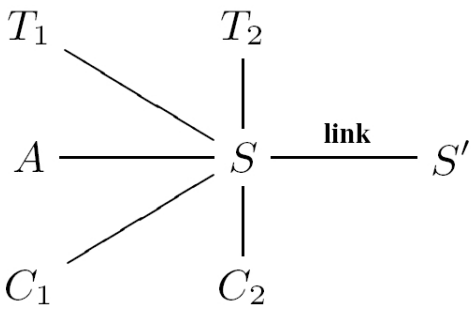

FIGURA 7: CLIENTES Y UN SISTEMA TOLERANTE A FALLAS BÁSICO

mínimo, mandando datos binarios desde los dispositivos a computadoras y viceversa; el resto de este artículo omite la descripción y el desarrollo de esta parte del sistema.

A continuación describimos la arquitectura básica y el modelo de nuestro sistema distribuido. Hemos clasificado a los usuarios en tres tipos básicos: profesores, estudiantes, y un administrador. Después, se describe cada cliente y su(s) funciones o acciones asociadas.

4.1. Clientes

Para continuar, a pesar de tener un sistema centralizado, los clientes son clasificados de acuerdo a sus características. Añadimos a nuestro modelo un nuevo tipo de usuarios. Estos usuarios se agregan al sistema Un tercer tipo de usuarios es también deseable: el personal necesario para estar en contacto con los servidores principales. Para evitar considerar los requisitos de bajo nivel, la arquitectura sólo incluye un administrador. Esta decisión es principalmente para considerar el servidor como un "front-end" de los procesos. A continuación, consideramos una clasificación de los nodos C_1 y C_2 como estudiantes y profesores. También consideramos que los nodos A representan al personal técnico o especializado que atienden al sistema en general.

(Un profesor puede ser un técnico;. Pero en cualquier caso, es sano separar claramente sus actividades o roles.) Dentro de nuestro modelo, los técnicos tienen una actividad de bajo nivel con respecto al diseño del sistema. Por último, se identifican los nodos de T_1 y T_2 como profesores.

El sistema dado en la Figura 6 puede ser mejorado para soportar la tolerancia a fallos como se muestra en la Figura 7 mediante un enlace (link). A partir de ahora usaremos este esquema para dar vida a un sistema virtual a distancia de enseñanza aprendizaje, donde la tolerancia a fallos es incluida. El aprendizaje hace hincapié de acuerdo a las características propias del estudiante, la enseñanza se pone de relieve, según los requisitos y funciones desde el punto de vista de los docentes.

Si el servidor acepta la conexión con estos usuarios, esto genera un nuevo tipo de configuración del

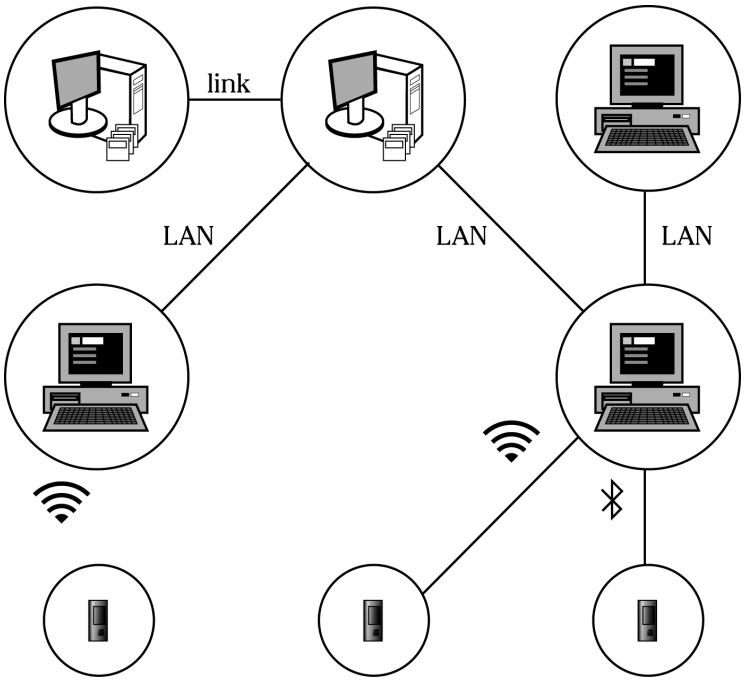

FIGURA 8: UN SISTEMA DISTRIBUIDO PARA ENSEÑANZA-APRENDIZAJE

sistema. Nosotros representamos a la configuración generada en la Figura 7, en donde hemos añadido una clase especial de clientes llamados administradores (en este caso, sólo uno), y un mecanismo básico para apoyar la tolerancia a fallos: ligamos un servidor S con un servidor S'. Si el servidor S falla, S' está dispuesto a apoyar todo el sistema mediante la técnica de las funciones integradas de Erlang. Es sorprendente cómo conseguir un comportamiento sofisticado del sistema con tan sólo conocimientos básicos de Erlang.

Una posible configuración de nuestro sistema distribuido se da en la Figura 8, en donde se han omitido los canales de comunicaciones virtuales, la Figura 8 describe el caso de GUIs que se ejecutan en dispositivos móviles, en donde los recursos locales son a veces muy limitados, dejando el trabajo fuerte a las computadoras de escritorio. Las tecnologías Bluettoth y Wi-Fi son necesarias para implementar la comunicación entre dispositivos móviles y computadoras (Huang y Rudolph 2007; Raymond 2003) utilizando Java (Knudsen 03; Keogh y Keogh 2003).

4.2. Acciones y eventos

Después de haber dividido el conjunto de los usuarios del sistema en tres subgrupos, tenemos que especificar qué funciones o actividades están disponibles para cada subconjunto. Cabe que señalar que, a pesar de tener la transparencia de referencia a nivel local, a nivel mundial, Erlang depende del flujo del tiempo cuando el sistema se está ejecutando. Podemos pensar en los usuarios como agentes: los usuarios tienen un comportamiento descrito por un conjunto de acciones llevadas a cabo a través de eventos. Para nosotros, lo eventos importantes están dados por la interacción con el resto del sistema distribuido, tanto con los procesos como con otros eventos. Creemos que los acontecimientos en algún razonamiento temporal establecido: los eventos pueden tener asociado un reloj lógico o se incorporarse dentro de un tiempo absoluto.
Los estudiantes tienen las siguientes actividades disponibles:

1. El comando subscribe: un alumno puede tratar de pertenecer al sistema, esta petición se envía al servidor, después de llenar un formulario, la respuesta del servidor puede ser aceptar (accept) o rechazar (reject);
2. El comando logon: un estudiante puede iniciar sesión en el sistema, el servidor intenta iniciar sesión con el estudiante: las respuestas pueden ser: you are log on o you are not accepted;
3. El comando test: una respuesta puede ser: Type of test?, y el estudiante puede seleccionar algún tipo de examen; después, el examen es mandado al estudiante desde el servidor; proponemos dos tipos de exámenes: simulados o reales; después de la selección, el estudiante recibe un examen del tipo seleccionado;
4. El comando do_test: un estudiante puede hacer un examen en el intervalo (tiempo absoluto) $[t_1, t_2]$, $t_1<t_2$; el tiempo puede ser monitoreado desde el servidor o directamente desde una computadora local;
5. El comando chat(amigo): un estudiante puede platicar con otro estudiante llamado amigo si este no está ocupado (por ejemplo, si el no está haciendo un examen);
6. El comando invitation(amigo): este comando es una petición de comunicación con otro estudiante llamado amigo. La respuesta puede ser aceptado (accept) o rechazado (reject);
7. El comando cgroup(nombre): es un comando para crear un grupo de trabajo: los estudiantes pueden decidir unirse a este grupo; el líder del grupo es la persona que creó tal grupo;
8. El comando who: revela quien está dentro del sistema;
9. El comando group: muestra los grupos existentes en el sistema;
10. El comando chat(profesor): un estudiante puede platicar con un profesor, si el profesor está disponible;
11. El comando exit: un estudiante puede salir del sistema, y así terminar la sesión;
12. El comando resources: este comando indica los recursos disponibles (notas, libros, etc);
13. El comando history_resources: este comando permite saber un historial de los recursos obtenidos;
14. El comando update_resources: permite el acceso a un repositorio local para obtener los últimos recursos.

Comandos adicionales pueden estar disponibles para los estudiantes y profesores, pero estos comandos pueden ser estándares o locales a la computadora. Un trace es una sucesión de eventos. Cada agente tiene asociado un trace con respecto al sistema. Los eventos tienen asociados un tiempo absoluto, pero a veces sólo un tiempo relativo es necesario para validar el flujo de eventos. Un estudiante debe saber quién o quiénes está disponibles para platicar (chat) (incluidos otros estudiantes o profesores). Esto puede ser regulado por el servidor.

Profesores tienen las siguientes actividades:

1. El comando subscribe: un profesor puede tratar de pertenecer al sistema, esta petic se envía al servidor, después de llenar un formulario, la respuesta del servidor puede ser aceptar (accept) o rechazar (reject);
2. El comando logon: un profesor puede iniciar sesión en el sistema, el servidor intenta iniciar sesión con el estudiante: las respuestas pueden ser: you are log on o you are not accepted;
3. El comando desing_test: un profesor puede diseñar un examen, por la naturaleza de nuestro modelo creemos que un examen puede ser un recurso compartido, explicaremos esta parte con más profundidad después;
4. El comando public(examen): un profesor puede publicar un examen;
5. El comando chat(amigo): un profesor puede platicar con otra persona (estudiantes o profesor);
6. El comando who: revela quien está dentro del sistema;
7. El comando exit: un profesor puede salir del sistema, y así terminar la sesión;
8. El comando resources: este comando indica los recursos disponibles (notas, libros, etc.);
9. El comando history_resources: este comando permite saber un historial de los recursos obtenidos;
10. El comando update_resources: permite el acceso a un repositorio local para obtener los últimos recursos.

El servidor administra cada petición y coordina la ejecución de todo el sistema. Otras tareas del servidor deben bloquear la comunicación con el mundo exterior, archivos log (es un archivo especial oculto) usados por los estudiantes, y limitar el tiempo para resolver un examen. La idea es utilizar la computadora como un asistente bien entrenando para estudiantes y profesores.

5. Especificación formal a nivel interno

Para caracterizar el flujo interno de las acciones de cada usuario, se utiliza CCS (Milner 1989). Para expresar la activación secuencial de los procesos (siguiendo las pautas dadas en (Noll y Roy 2005; Roy, Noll, Roy y Cordy 2006)); aquí hay que entender la definición de proceso como:

$$\text{Proc} \stackrel{\text{def}}{=} \text{signalIn.Proc1} + \overline{\text{signalOut}}.\text{Proc2}$$

En donde la notación del punto significa secuencia y el signo más (+) es una opción exclusiva, es decir, sólo puede elegir una sola de las opciones a la vez. (La barra sobre signalOut indica que es una señal de salida; en cambio, signalIn no tiene barra porque es una señal de entrada). Algunas notas generales sobre un formalismo para modelar procesos de Erlang se presenta en la tesis (Chugunov 2004). Las señales son generadas por procesos para los propósitos de sincronización. Diferenciamos entre una señal y un proceso mediante la revisión de la primera letra en el nombre, letra minúscula para las señales y letra mayúscula para los procesos, por ejemplo, thisIsASignal y ThisIsAProcess. También incluimos un estado de inactividad de la siguiente manera:

$$\text{Idle} \stackrel{\text{def}}{=} \text{Idle.Something.}$$

Tenemos un modelo que se compone de tres partes: procesos Admin, Student y Teacher. Un proceso es un módulo de software que implementa la comunicación con otros módulos y ofrece la funcionalidad del sistema. Admin es un proceso que coordina todo el sistema al permitir o bloquear las comunicaciones de las otras partes. Admin generará otros procesos cuando sea necesario según la interacción del usuario. Los procesos Teacher y Student interactúan directamente con un estudiante o profesor humano. En cualquier tiempo los estudiantes o profesores necesitan recursos del sistema, los cuales se le pedirán a Admin.

Este enfoque toma en cuenta la capacidad de Erlang para generar procesos a voluntad, así como su interfaz de comunicación transparente. Además, para conectar a los

procesos que no son de Erlang siempre es posible usar los sockets para resolver este problema. A continuación se muestra por razones de simplicidad sólo una parte de la definición del proceso Student; ver Figura 9.

Student es un proceso que modela las actividades de usuario del estudiante y oculta los aspectos de intercambio de comunicación. Un modelo análogo se mantiene para Admin y Teacher. En esta definición algunos de los estados son etapas finales y tendrán una actividad de trabajo normal resultando en el cierre de la aplicación como un medio para guardar información importante y reportarla a Admin. Después de cerrar una aplicación el sistema siempre regresará al estado inicial. En un ámbito más grande, Admin es un proceso central que coordina actividades de educación y acepta las actividades de usuario conectado o desconectado, así como el control de acceso y privilegios. El startEnviroment y el startInterface serán el punto de comunicación con el usuario final que podría hacer uso de cualquier dispositivo con pocos recursos como un PDA o un teléfono celular. Podemos definir un modelo distribuido de Admin para que podamos pensar en una actividad como un caso limitado en tiempo y espacio.

Nuestro modelo de comunicación oculta los detalles de las actividades del proceso y nos concentramos en el paso de mensajes para la activación del proceso. En primer lugar, se mostrará la forma de definir el proceso Clock, el cual genera las señales de onTime y de timerOut que interactúan con los procesos Login y Counter de manera concurrente, para juntos controlar el acceso al sistema.

$$\mathrm{Clock} \stackrel{\mathrm{def}}{=} \overline{\mathrm{onTime}}.\mathrm{Clock} + \overline{\mathrm{timerOut}}.\mathrm{Clock}$$

De esta forma, el Clock y los procesos login trabajan juntos de la siguiente forma: Login | Clock. Siguiendo una idea análoga podemos agregar un proceso que cuente los intentos de entrada: Counter | Login | Clock. Actualmente, estamos trabajando en dar alguna explicación formal y detallada de cada proceso concurrente.

6. Especificación formal a nivel externo

Habiendo descrito como tratar con la sucesión de las acciones, ahora presentamos algunas técnicas pertenecientes al lenguaje de modelado unificado (UML) para especificar el comportamiento externo del sistema, incluido el manejo de excepciones. Esta parte ilustra la aplicación de técnicas de especificación de acuerdo a las necesidades del desarrollo del sistema.

Para especificar el comportamiento del sistema desde un punto de vista externo, hemos aplicado el modelo de casos de uso de UML (Cockburn 1997). Este modelo se aplicó para especificar la interacción de los distintos actores (estudiantes, profesores o administradores) con el sistema distribuido en desarrollo. Un actor puede

```
Student def= Student.Login

Login def=
    ReqAdminAut(user,pswd).(Ok +
    Bad +
    noMoreAttempts +
    noGoodClosing.Recover +
    cancel.Student)

Recover def=
    LoadStudentLastGoodState.startEnv(stateInfo)

Ok def=
    goodLogin.LoadStudentPreferences.StartMenu +
    goodLogin.LoadStudentLastActivity.startEnv(lastActivity)

Bad def=
    timerOut.LockStudentAccount +
    wrongUser.increaseCounter.Login +
    wrongPasswd.increaseCounter.Login

NoMoreAttempts def=
    maxCount.LockStudentAccount

LoadStudentLastGoodState def=
    GetAdminConnection(stateInfo).SearchStudentLog(stateInfo)

LoadStudentPreferences def=
    StartStudentCommModel.CheckSchedule.
        JoinCourse.SendSyncInfo

StartMenu def=
    LoadOptions.StartInterface

LoadStudentLastActivity def=
    SearchStudentLog(lastActivity)

StartEnv(x) def=
    InitProperState(x).StartInterface

GetAdminConnection(command) def=
    RequestSafeMode.SendCommand(command).GetSystemResponse

SearchStudentLog(x) def=
    GetAdminConnection(x).QueryDataBase(studentId, x)

StartStudentCommModel def=
    LoadGroupInfo(courseAtendantList).SetPrivilegies.StartCommSetMenu

CheckSchedule def=
    GetAdminConnection(scheduleInfo).
    queryDataBase(studentId, scheduleInfo).
    LoadProperActivity

JoinCourse def=
    RequestCourseServConnection(courseSelection).LoadCourseMaterial
```

FIGURA 9: LA CARACTERIZACIÓN DE UN ESTUDIANTE VÍA FLUJO DE ACCIONES

ser una persona, un grupo de personas o un proceso Erlang (este proceso por sí mismo está fuera del sistema distribuido). Un caso de uso también describe las características del sistema que se está diseñando, sin ningún compromiso con los detalles de implementación, aunque en nuestra aplicación concurrente se han mezclado algunas etapas de implementación y especificación. La metodología de programación extrema sugiere escribir historias de usuario, para el mismo propósito que los casos de uso; pero los casos de uso y las historias de usuarios no constituyen la misma especificación técnica, en primer lugar, las historias de usuario se escriben en un formato breve que incluye un párrafo de texto, en segundo lugar, son escritas por el propio usuario (especificando las funciones que el o ella necesitan del sistema), además estas historias son usadas para definir las estimaciones de tiempo para la liberación y, finalmente constituyen el reemplazo de un documento muy detallado y grande de requerimientos. La principal desventaja de las historias de usuario es que pueden dejar de servir como documentación fiable del sistema y también pueden ser difíciles escalarlas hacia proyectos grandes. En lugar de aplicar historias de usuario, hemos adoptado el modelo de casos de uso para utilizarlo en la documentación de los requisitos funcionales del sistema, y de esta forma especificar el comportamiento externo del mismo. El modelo de casos de uso nos ofrece adicionalmente una base para esti-

1	**Caso de uso:** Ingreso al sistema
2	**Descripción breve:** El usuario realiza el proceso de ingreso al sistema.
3	**Ámbito:** Sistema.
4	**Nivel:** Meta de usuario.
5	**Precondiciones:** El usuario debe estar registrado previamente (a través del comando subscribe).
6	**Poscondiciones:** El usuario es autentificado exitosamente e ingresa al sistema.
7	
8	**Escenario principal de éxito:**
9	1.El usuario introduce sus datos de autentificación en el sistema.
10	2.El sistema notifica al usuario que ha sido aceptado.
11	3.El sistema carga las preferencias del usuario.
12	**Extensiones:**
13	2a. El sistema detecta que los datos de autenticación son incorrectos:
14	2a.1. El sistema notifica al usuario que ha sido rechazado.
15	2a.2. El sistema registra el evento en su bitácora.
16	2b. El sistema detecta que el usuario ha realizado n intentos fallidos para ingresar al sistema:
17	2b.1. El sistema bloquea la cuenta del usuario.
18	2b.2. El sistema registra el evento en su bitácora.
19	2c. El sistema detecta que el tiempo para ingresar los datos ha expirado:
20	2c.1. El sistema bloquea la cuenta del usuario.
21	2c.2. El sistema registra el evento en su bitácora.
22	2d. El sistema detecta que la última sesión del usuario no fue cerrada correctamente:
23	2d.1. El sistema carga el último estado correcto.
24	2e. El sistema detecta que el usuario ya ha ingresado previamente:
25	2e.1. El sistema notifica al usuario que ha sido rechazado.
26	2f. El usuario cancela el procedimiento de autentificación:
27	2f.1. El sistema notifica al usuario que procedimiento de ingreso ha sido cancelado
28	2f.2. El usuario retorna al paso 1 (línea 6).
29	3ª. El sistema detecta que hay actividades pendientes desde la última sesión del usuario:
30	3ª.1. El sistema carga las actividades pendientes desde la última sesión del usuario.

FIGURA 10: ESPECIFICACIÓN DEL CASO DE USO DEL COMANDO LOGIN

mar, agendar y validar los esfuerzos de programación. En resumen, hemos adoptado la técnica de modelos de caso de uso como sustituto de historias de usuario propuestas por la metodología de programación extrema.

Debemos construir un caso de uso para cada actividad realizada por un actor, con el objetivo de crear una visión general de las funciones que el sistema debe proporcionar a los usuarios. La Figura 10 muestra la especificación del caso de uso para el comando login[4] cuando es ejecutado por un usuario del sistema, los demás casos se omiten por razones de brevedad. Hemos hecho énfasis especial en el manejo de las excepciones. El manejo de las excepciones es una contribución importante de la especificación del caso de uso para el sistema, debido a que permiten realizar la planeación de todas las formas alternativas o posibles eventos anormales que pueden ocurrir durante la ejecución del escenario principal exitoso del caso de uso, de esta manera se pueden anticipar los comportamientos erróneos así como su tratamiento, logrando así una mejor robustez del sistema.

Una visión alterna de esta especificación de casos de uso se puede conseguir usando diagramas de secuencia UML, siguiendo este camino, los diagramas de secuencia describen la lógica de los eventos entre un actor y el sistema en un estilo gráfico, permitiendo documentar tanto el flujo de mensajes como la validación lógica del paso de mensajes de Erlang. Los diagramas de secuencia muestran el caso de uso principal en un escenario exitoso, además de varios escenarios alternativos para las excepciones.

En las próximas secciones trataremos con cuestiones educativas y trabajo colaborativo.

7. Diseño y gestión de exámenes

Ahora centramos nuestra atención en la única parte de colaboración que tenemos actualmente: los examenes, en (Joy, Muzykantskii, Rawles y Evans 2002) existes cuestiones relacionadas.

Los exámenes pueden tener distintos formatos: desde opción múltiple a preguntas abiertas (Crawley 2001). El punto principal a tener en cuenta en el sistema es el grado de automatización deseable.

Los profesores tiene experiencia sobre las tareas necesarias a realizar, las cuales pueden ser aburridas, cansadas y repetitivas, como revisar tareas y calificar exámenes. Una prueba bien diseñada va de la mano de un buen método de evaluación. Excepto en los cursos que requieren recursos especiales en el aula (por ejemplo, la evaluación de proyectos podría ser difícil que la evaluara una computadora). Los exámenes de opción múltiple son un ejemplo de una tarea que puede ser automatizada (en el diseño y evaluación del mismo).

Cada pregunta a evaluar debe estar asociada a un nivel n, en donde n es el nivel descrito por la taxonomía de Bloom (Bloom 1956) (conocimiento, compresión, aplicación, análisis, síntesis y evaluación). Esta taxonomía es la clasificación de niveles de abstracción de las preguntas que comúnmente ocurren en las escuelas. La taxonomía también proporciona una estructura útil para caracterizar las preguntas del examen ya que los profesores hacen preguntas con niveles en particular. Esta taxonomía es reconocida oficialmente en México[5]. Las preguntas precisamente, son el tema a tratar en la siguiente subsección.

7.1. Preguntas

Los exámenes consisten de preguntas. Nosotros seguimos dos criterios para diseñar exámenes: evaluación automática o evaluación directa por el profesor.

Creamos una base de datos de preguntas, la siguiente lista de comandos devuelve aceptar (accept) o rechazar (reject) dentro de nuestro sistema debido principalmente a las restricciones de integridad de la base de datos. Los profesores usan estos comandos. La n es el número asociado siguiendo la taxonomía de Bloom.

1. questionTF (pregunta, respuesta,n): almacena una pregunta con respuesta verdadero o falso.
2. questionMO (pegunta,respuesta,op1,op2, op3,n): almacena una pregunta de opción múltiple, la posibles respuestas aparecen en un orden aleatorio. (Normalmente, cuatro respuestas son presentadas para este tipo de preguntas, al menos tres son nombradas distractores y sólo una de ellas es llamada respuesta; sólo existe una respuesta correcta)
3. questionSA (pregunta,respuestaComplementaria,n): guarda una pregunta a ser complementada con una respuesta corta, normalmente, una palabra o un número.

4 Un caso de uso detallado, siguiendo la terminología UML

5 CENEVAL: Un acrónimo del Centro Nacional de Evaluación para la Educación Superior

Las pruebas bajo supervisión son creadas por un comando question(plantilla). Este comando crea y almacena plantillas usadas para preguntas abiertas. Otros comandos para agregar información a la base de datos son los siguientes:

1. cSubject(tema): permite crear un nuevo tema, regresa: aceptado o rechazado;
2. oSubject(tema): se abre la base de datos para la búsqueda o actualización, regresa: aceptado o rechazado;
3. close(): cierra una base de datos abierta.

Actualmente estamos usando una base de datos sencilla, aunque una base de datos como Mnesia nos daría un buena apoyo para esta parte de nuestra propuesta. A través de ODBC se puede interactuar con algunas bases de datos más comunes, como PostgreSQL[6] y MySQL[7]

7.2 Elementos de los exámenes

Considerando el número y tipo de evaluaciones que se pueden presentar dentro de una universidad teniendo como caso específico la Universidad Tecnológica de la Mixteca. Existen al menos ocho tipos de evaluaciones las cuales se muestran en al tabla 1.

Elementos para la creación de un examen.

1. createExam(subject,tipoExam,num_exam,time). Crea el contenedor del examen. Tiene como parámetros el tema o materia a la que pertenece el examen, el tipo que corresponde a alguno de los indicados en la tabla anterior, el número de examen de ese tipo y por último el tiempo designado por el profesor para su solución. Se deberá crear el contenedor del examen antes de adicionar preguntas. En un inicio se considera que el examen sólo puede tener preguntas de respuesta automática o sólo preguntas de calificación supervisada.
2. openExam(identificador). Abre el contenedor del examen para su revisión o modificación.
3. searchQuestion(subject,tipo,nivel). Busca las preguntas candidatas a ser insertadas en el examen, requiere el tema de la base de preguntas, el tipo de pregunta y el nivel de acuerdo con la taxonomía de Bloom. Devuelve los elementos de la pregunta más el identificador de ésta.
4. selectQuestion(identificador,percentil). Selecciona las preguntas que serán anexadas al examen. Requiere identificador de la pregunta y el percentil asignado por el profesor.
5. deleteQuestion(identificador). Elimina la pregunta del contenedor del examen. Sólo puede eliminar una pregunta el creador del examen.
6. closeExam().Cierra el contenedor del examen para su revisión o aplicación.
7. revisión(contenedor,group). Envía el examen al grupo para su revisión.
8. aplication(contenedor,group,date). Programa la aplicación del examen para un grupo o alumno determinado en la fecha especificada.
9. automatic(). Calcula y muestra la calificación obtenida por el grupo o personal al que se le

id	Tipo	Número	Descripción
1	Diagnostica o formativa	Lo determina el profesor	Son aquellas que se realizan para determinar el estado de conocimiento inicial o temporal que tiene el alumno.
2	Parcial	Tres	Estas se llevan a cabo durante el proceso de aprendizaje, y permite determinar el grado de adquisición del conocimiento que tiene el alumno en un tema dado.
3	Sumativa	Al menos una	Evalúan el conocimiento global que tiene un alumno en una materia dada.

TABLA 1. TIPOS DE EVALUACIÓN.

6. www.postgresql.org
7. www.mysql.com

aplicó el examen. Válido sólo para exámenes con preguntas con calificación automática.

10. supervión(contenedor,alumno). Abre el contenedor de respuestas para un examen con calificación supervisada.
11. calif(question,percentil). Da el valor designado a la respuesta del alumno en percentiles. Requiere previa apertura del examen del alumno.
12. total().Calcula y muestra la calificación alcanzada por el alumno, en un examen con calificación supervisada.
13. final(contenedor, alumno). Muestra la calificación obtenida en el examen por un alumno determinado.

8. Trabajo colaborativo entre profesores

En un sistema como éste, existen muchas tareas que se pueden realizar de forma colaborativa. Mencionamos las más destacadas: el diseño de pruebas. Esta tarea requiere:

1. Modelos de recursos compartidos
2. Administración de la base de datos

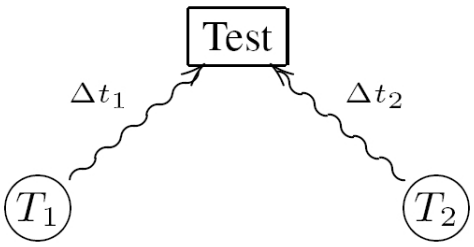

FIGURA 11: EXÁMENES COLABORATIVOS (T_1 Y T_2 SON PROFESORES)

Para tratar con exámenes, damos una descripción de trabajo colaborativo entre profesores:

1. Planificación de recursos: varios profesores modifican el mismo examen en distintos intervalos de tiempo, y
2. Particionamiento de recursos: varios profesores modifican un fragmento del mismo examen.

Para la planificación del diseño de un examen (Figura 11), dos o más profesores pueden diseñarlo, aunque siguiendo algunas restricciones dadas por la Ecuación 1 de la Figura 12: no puede ser posible que dos profesores (T_1 y T_2) puedan modificar el mismo examen en el mismo intervalo de tiempo. La flecha curveada en la Figura 11 indica que cada profesor modifica el mismo examen en distintos intervalos de tiempo. Para generalizar, los profesores pueden modificar un examen en un intervalo de tiempo que satisfaga la Ecuación 2 de la Figura 12.

$$\Delta t_1 \bigcap \Delta t_2 = \emptyset, \tag{1}$$

$$_1 \bigcap \Delta t_2 \bigcap \cdots \bigcap \Delta t_n = \emptyset, \tag{2}$$

FIGURA 12: ECUACIONES QUE DAN RESTRICCIONES SOBRE EL TIEMPO PARA LA MODIFICACIÓN DE EXÁMENES

Ahora describiremos el particionamiento de recursos. Dado un recursos R, si podemos particionar R como

$$R = \bigcup_i R_i \text{ and } \bigcap_i R_i = \emptyset \tag{3}$$

Podemos aplicar una técnica de particionamiento de recursos.

Para la partición de recursos, el examen debe dividirse en m partes, como se muestra en la Figura 13 en el tiempo t_1, aunque un fragmento R_i puede se modificado por un profesor T (Figura 14), y cada fragmento puede también ser modificado siguiendo la planificación de recursos previa: dos profesores no pueden modificar un fragmento al mismo tiempo. En la Figura 13 podemos ver el posible desarrollo de un recurso a través de un particionamiento: por ejemplo, en el tiempo t_0 no se ha fragmentado el recurso, pero en el tiempo t_1 el recurso R fue fragmentado en m

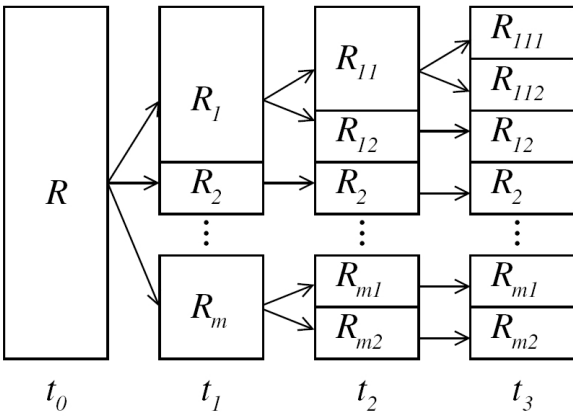

FIGURA 13: EVOLUCIÓN DE LA PARTICIÓN

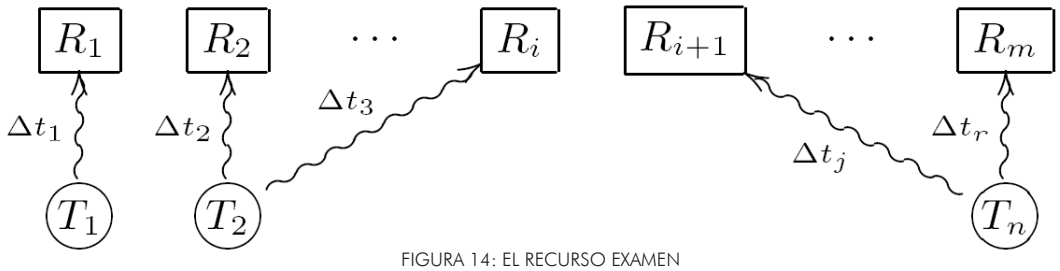

FIGURA 14: EL RECURSO EXAMEN

partes, y en el tiempo t_2 algunas partes pequeñas (R_1) pueden ser fragmentadas de nuevo en partes más pequeñas (R_{11} y R_{12}), y así sucesivamente. La condición para dejar de fragmentar está dada quizás por algún criterio humano.

Los subíndices asociados a cada parte del recurso está basado en un dominio de árboles, siguiendo el concepto de dominio de árboles, la partición tiene la siguiente descripción: R_{uv}, en donde u es el índice de la rama de su ancestro y v es e índice de la rama de su descendiente. Además, cada partición tiene adjunta información adicional (ver Figura 15), la cual es importante para mantener información relevante del recurso R_{uv}

Como se ha señalado en (Wiger 2007), el etiquetado es una parte importante de un sistema distribuido (en la obra citada, el etiquetado se utiliza para mejorar el mecanismo de nombres de procesos en Erlang). Nosotros proponemos un mecanismo de etiquetado para tratar con los recursos. Este mecanismo otorga cierta información relevante para cada recurso, para indicar como un recurso debe ser usado cuando está siendo compartido.

Para asegurar la persistencia de los datos, nosotros manejamos el concepto de versiones. Dado un recurso R, damos la función:

$$v : \text{ResourceType} \longrightarrow \text{Labels} \qquad (4)$$

Etiquetando cada recurso R, en donde las etiquetas son un tipo de dato que contiene información relevante sobre el recurso R. Normalmente información sobre fecha, autor y permisos.

R_i		
Autor	*Date*	*Permission*

FIGURA 15: RECURSOS ETIQUETADOS

Si deseamos administrar de forma segura las versiones (Legaria, Rivera , Vásquez y Hernández 2010) de un recurso, procedemos de la siguiente forma: dado un número n, podemos mantener una cola de tamaño n, en donde los elementos más antiguos de la cola corresponden a una versión más vieja y los elementos más recientes corresponden a versiones más nuevas del recurso. Aunque el número n es fijo, usamos la función v como un identificador de etiquetado de las versiones más recientes del recurso R. Este mecanismo tiene dos ventajas; por un lado, si n es muy grande, siempre tendremos un respaldo del trabajo previo; por otro lado, podemos asegurar la validez de nuestras nuevas versiones sin perder las anteriores. La principal desventaja es el desperdicio de memoria.

9. Conclusiones

En este trabajo hemos tratado de ampliar el ambiente del aula tradicional a través de una herramienta de enseñanza-aprendizaje asistida por computadora. Como muchos han experimentado en algún momento de su vida, un profesor es como un moderador o facilitador en el proceso de la transferencia de conocimiento a los alumnos, teniendo el control de casi todos los pasos que el alumno puede dar. El contenido y el orden del material de las clases son siempre preparadas por uno o más miembros del personal. Como retroalimentación, un profesor (o grupo de profesores) planea los exámenes, pruebas, ejercicios, etc. A veces también se puede diseñar un proyecto de aplicación para evaluar el progreso de los estudiantes. Todas estas tareas pueden ser abrumadoras y no siempre son suficientes para satisfacer las necesidades de los estudiantes. Hemos decidido que el proceso de la enseñanza se puede modelar como un sistema distribuido y concurrente realizado por varios actores colaborando juntos para

hacer que el objetivo común de la transferencia del conocimiento sea una tarea agradable. Mantener el seguimiento del rendimiento de todos es una tarea que puede ser realizada con la ayuda de una computadora. En un marco metodológico, hemos tratado de complementar la programación extrema mediante el uso de la programación declarativa y Erlang. Dos puntos clave hacen de Erlang una buena herramienta para experimentar el desarrollo de un sistema de educación asistido por computadora: en primer lugar, eventos concurrentes y actividades colaborativas implican una carga fuerte de comunicación, en segundo lugar, como una metodología hemos elegido programación extrema para mostrar que los lenguajes funcionales pueden adecuarse a las restricciones asociadas a esta metodología. En el desarrollo de la producción de las especificaciones del sistema hemos confiado en algunos métodos formales como: CCS y UML. Algunas decisiones se siguen madurando por nuestro grupo de investigación para producir un sistema que transfiera nuestra experiencia como profesores y alumnos.

Desde sus inicios la programación extrema ha sido criticada por dejar a un lado la documentación. En la próxima etapa, queremos crear un sistema informático destinado a apoyar la programación extrema y la transformación de programas. También estaríamos felices si logramos incorporar inteligencia artificial a nuestro sistema: eventualmente, las tareas de los profesores automatizadas completamente, o proveer un administrador como un agente racional. Finalmente, se requiere una mayor investigación en la administración de recursos compartidos.

Erlang, con su filosofía de paso de mensajes, tolerancia a fallas y programación funcional, está dispuesto a aceptar el desafío de programación de la nueva generación de problemas de cómputo ubicuo como un caso particular de la programación distribuida. Aspectos similares pueden ser aplicados a computadoras multinúcleos. La ingeniería de software tendrá que hacer frente a este tipo de tecnologías.

10. Referencias

Armstrong, Joe.

2002 Getting Erlang to talk to the outside world. In ERLANG '02: Proceedings of the 2002 ACM SIGPLAN workshop on Erlang. New York, NY, USA. pp. 64–72.

Armstrong, Joe.

2007 Programming Erlang. Software for a concurrent world. The Pragmatic Programmers.

Bauer, F. L., Broy, M., Gnatz, R., Hesse, W., Krieg-Brückner, B., Partsch, H., Pepper, P., Wössner, H.

1978 Towards a wide spectrum language to support program specification and program development. SIGPLAN Not., 13(12). 15–24pp.

Bird, Richard.

1998 An introduction to functional programming using Haskell. Prentice–Hall, Second Edition.

Bloom, Benjamin S.

1956 Taxonomy of Educational Objectives, Handbook 1: Cognitive Domain. Addison Wesley Publishing Company.

Booch, Grady & Rumbaugh, Jame.

1998 The unified modeling language user guide. Addison-Wesley.

Burstall, Rod M., Darlington, John.

1977 A transformation system for developing recursive programs. Journal of the Association for Computing Machinery, 24(1). 44–67pp.

Cabrero, David, Abalde, Carlos, Varela, Carlos, Castro, Laura.

2003 Armistice: an experience developing management software with erlang. In ERLANG '03: Proceedings of the 2003 ACM SIGPLAN workshop on Erlang. New York, NY, USA. 23–28pp.

Chugunov, Gennady.

2004 Reasoning about side-effect free Erlang code in a modal μ-calculus based verification framework. SICS, dissertation. KTH, Sweden.

Cockburn, Alistair.

1997 Goals and use cases. JOOP, 10(5). 35–40pp.

Cortés, Hugo, García, Mónica, Hernández, Jorge, Hernández, Manuel, Pérez-Cordoba, Esperanza, Ramos, Erik.

2009 Development of a distributed system applied to teaching and learning. In Erlang '09: Proceedings of the 2009 SIGPLAN workshop on Erlang Workshop. New York, NY, USA. 41–50pp.

Cousot, Patrick, Cousot, Radhia.

1992. Abstract interpretation and application to logic programs. Journal of Logic Programming. 13. 103–179pp.

Crawley, E.F.

2001 The cdio syllabus a statement of goals for undergraduate engineering education. Technical

report, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics. Cambridge, Massachusetts.

Dijsktra, E.W.

1976 A discipline of programming. Prentice–Hall.

Guttag, John.

1986 Notes on Type Abstraction. In N. Gehani and A.D. McGettrick, editors, Software Specification Techniques. International Computer Science Series. Adison–Wesley.

Huang, Albert S., Rudolph, Larry.

2007 Bluetooth Essentials for Programmers. Cambridge University Press.

Hughes, John.

1989 Why functional programming matters. Computer Journal, 32(2).

Joy, Mike, Muzykantskii, Boris, Rawles, Simon, Evans, Michael

2002 An infrastructure for web-based computer-assisted learning. J. Educ. Resour. Comput., 2(4):4.

Beck, Kent.

1999 Extreme Programming Explained. Embrace change. Addison-Wesley Professional.

Keogh, James, Keogh, James, Edgard.

2003 J2ME: The Complete Reference. McGraw-Hill/Osborne.

Knudsen, Jonathan

2003 Wireless Java: Developing with J2ME. Apress.

Knuth, Donald

1992 Literate Programming. Center for the Study of Language and Information.

Legaria, Fernando, Rivera, Luis, Vásquez, Nashielly, Hernández, Jorge.

2010 Módulo colaborativo centrado en un moderador: una herramienta para los Sistemas de Gestión de Aprendizaje (LMS). SENIE por aparecer.

MacLennan, Bruce J.

1990 Functional programming. Practice and theory. Addison–Wesley.

Milner, Robin.

1968 Communication and concurrency. Prentice-Hall.

Noll, Thomas, Roy, Chanchal, Kumar.

2005 Modeling Erlang in the pi-calculus. In ERLANG '05: Proceedings of the 2005 ACM SIGPLAN workshop on Erlang. New York, NY, USA. pp. 72–77

Partsch, Helmut.

1990 Specification and Transformation of Programs. Texts and Monographs in Computer Science. Springer-Verlag.

Pettorossi, Alberto, Proietti, Mauricio.

1996 Rules and Strategies for Transforming Functional and Logic Programs. ACM Computing Surveys, 28(2). 360–414pp.

Raymond, Smith.

2003 Wi-Fi Home Networking. McGraw-Hill/TAB Electronics

Roy, Chanchal, Kumar, Noll, Thomas, Roy, Banani, Cordy, James, R.

2006 Towards automatic verification of erlang programs by p-calculus translation. In ERLANG '06: Proceedings of the 2006 ACM SIGPLAN workshop on Erlang, New York, NY, USA. 38–50pp.

Wiger, Ulf T.

2007 Extended process registry for Erlang. In Erlang '07: Proceedings of the 2007 SIGPLAN workshop on Erlang Workshop. New York, NY, USA. 1–10pp.