Una propuesta de sistemas distribuidos con componentes autónomos definidos en SCEL e implementados en Erlang

Mónica García, Manuel Hernández, Ricardo Ruiz y Felipe Trujillo-Romero

Universidad Tecnológica de la Mixteca,
Huajuapan de Léon, Oaxaca, México

monicagg@mixteco.utm.mx
hg.manuel@gmail.com
rruiz,ftrujillo@mixteco.utm.mx

Resumen. *SCEL* es un formalismo para diseñar sistemas distribuidos con componentes autónomos. Erlang es un lenguaje de programación especializado en el tratamiento de problemas de cómputo distribuido. En este artículo, afirmamos que los programas escritos en Erlang son adecuados para proveer el soporte de software de *componentes autónomos* tal como están descritos por el formalismo SCEL. Damos una instancia mediante la descripción del núcleo de programación de un componente autónomo típico en un escenario robótico.

Palbaras clave: Erlang, sistema distribuido, sistema autónomo.

1. Introducción

SCEL (acrónimo del inglés *Software Component Ensemble Language*) [12] es un formalismo de modelado de sistemas distribuidos para el diseño, desarrollo e interacción de *componentes autónomos* y con un soporte teórico ahora bien establecido ([1,8]). La *programación en Erlang* (PE) [3,4,5] está apoyada principalmente por el lenguaje de programación funcional *Erlang*, y por la plataforma de desarrollo *OTP* (del inglés *Open Telecom Platform*). Este artículo presenta una propuesta para utilizar a los programas escritos en Erlang como una herramienta en la parte de software concerniente al manejo de comunicaciones de un componente autónomo según está definido por SCEL. Un objetivo similar fue planteado en [16] para el caso de agentes, en un contexto más amplio que el nuestro, al no utilizar Erlang solo para la parte de comunicaciones. Nosotros, y similarmente a [12], emplearemos un ejemplo de robots para describir el potencial de nuestra propuesta, afirmando que este esquema de trabajo se coordina bien con el formalismo de SCEL, manteniendo así una adecuada pureza teórica en una aplicación práctica.

Estructura de este trabajo

El panorama general de este trabajo es como sigue: Primero, daremos un recuento general del formalismo SCEL; segundo, presentaremos algunos fundamentos de la programación en Erlang (PE); finalmente, describiremos cómo Erlang es un buen candidato de algunas partes descritas en la teoría por el formalismo SCEL, dado su modelo computacional distribuido subyacente vía paso de mensajes sin recursos compartidos, anotando algunas conclusiones finales.

2. El formalismo SCEL

SCEL tiene como principal objetivo organizar algunas herramientas teóricas para el el diseño de sistemas distribuidos que consisten de *componentes autónomos*, mediante la identificación de *acciones* que tales componentes pueden realizar, y con la coordinación entre componentes para formar *grupos*, siendo el mecanismo principal de comunicación el intercambio o paso de mensajes. Cada componente autónomo (CA) está equipado con una *interface*, que a su vez consiste de una colección de *atributos*, tales como la identidad del CA, su memoria disponible, su poder computacional, su nivel de energía, su posición geográfica, su membresía de grupos, y otros más. Estos atributos permiten conformar grupos de CAs mediante la identificación de valores de atributos en común o por medio de la satisfacción de predicados. Cuando se forman grupos de CAs (GCA), tenemos posibilidad de realizar tareas coordinadas, que permiten una gran flexibilidad de desempeño y eficacia en el logro de los objetivos para los que fue diseñado el sistema.

Carecterizamos en SCEL un CA por lo siguiente:

1. *Conductas*, dadas por la ejecución secuencial o en paralelo de *acciones*, mismas que permiten un tratamiento teórico mediante sistemas con transiciones etiquetadas;
2. *conocimiento*, el cual permite que un CA almacene datos propios y en su caso, mediante *interfaces*, tenga acceso también a datos que no son los propios; se permite que este conocimiento de un CA sea manipulado internamente o por otros CAs (agregando, examinando, o eliminando la información);
3. *atributos*, los cuales se informan o se recuperan mediante las interfaces;
4. *políticas*, que permiten controlar un CA localmente y en su caso, globalmente.

Las propiedades auto-* (*self-* properties*) son una parte imprescindible de los modernos componentes autónomos. SCEL modela adecuadamente este tipo de propiedades, pues se catalogan como atributos que pueden ser visibles o no mediante interfaces y de acuerdo con las políticas vigentes. Ejempo de tales propiedades son los sensores de auto-evaluación (tales como los de nivel batería existente), de auto-protección (como los sensores de inclinación o temperatura, o bien la auto-detección de averías), o los de revisión de comunicaciones (detectando la intensidad de señal de comunicación, por ejemplo).

La siguiente es una seudo-ecuación que describe concisamente un CA:

$$CA = \mathcal{I}[\mathcal{K}, \Pi, P]$$

donde $\mathcal{I}$ representa la interface, $\mathcal{K}$ el conocimiento, y Π es la política que en un momento dado está vigente (se pueden manejar varias políticas para lidiar con condiciones cambiantes del ambiente). Aquí, también, P es un proceso que maneja la activación de acciones. Las *acciones*, precisamente, son parte de la instanciación que SCEL permite para que un sistema encaje en su paradigma. Identificar qué acciones son relevantes al sistema es una actividad fundamental del diseño de un sistema distribuido. SCEL tiene predefinido un conjunto básico de acciones relacionadas con el manejo de bases de conocimiento: **get**, **qry**, **put**, **fresh** y **new**. Cabe mencionar que una acción se describe completamente por medio de su ejecutor y del objetivo de la acción, con una posible referencia a otros CAs. Las políticas, en tanto, son utilizadas para reglamentar las acciones.

Ahora bien, la ejecución de las acciones no es trivial, al permitirse secuencialidad, concurrencia, compromiso (*commitment*) y posible monitoreo externo. Más información y definiciones formales extensas en [12]. En la siguiente sección mostramos los fundamentos de la programación en Erlang.

3. Erlang y sistemas distribuidos

Erlang y los sistemas distribuidos. Erlang es un lenguaje de programación de capacidad industrial orientado al tratamiento de problemas de sistemas distribuidos [3,5]. Erlang adopta un modelo de sistemas distribuidos basados en paso de mensajes sin recursos compartidos, con asincronía y concurrencia. Erlang concisamente puede ser descrito como sigue: *Erlang nodos* hospedan *procesos* (pensemos que a su vez los Erlang nodos están ejecutándose en una computadora determinada); tales procesos pueden comunicarse entre sí a través de *mensajes*. En principio, no existen canales de comunicación preestablecidos, y el flujo de información vía mensajes con remitentes es mediante el envío a *identificadores de procesos*, en tanto que del receptor se espera que el mensaje sea aceptado basándonos en casamiento de patrones (*pattern matching*) y en la identificación de los remitentes; de otra forma, los mensajes simplemente se ignoran. Los mensajes almacenan información heterogéna, desde bits hasta texto, o expresiones de tipo Erlang o inclusive procesos Erlang en sí.

Erlang y SCEL. En SCEL cada componente es diseñado siguiendo un enfoque de comunicación vía paso de mensajes, es equipado con una implementación de *conocimiento*, y rige sus acciones bajo algunas políticas de conducta. Con estas guías de diseño, el análisis de un sistema distribuido se facilita, haciendo incluso factible de forma inmediata la verificación por construcción de modelos (*model checking*) [11]. La programación en Erlang aquí mostrada fue realizada según el método incremental descrito informalmente en [3][p.148]. La idea básica de este tipo de programación es la creación de un proceso y el tratamiento gradual de los casos que se involucran en recibir y enviar mensajes. Siendo Erlang declarativo,

existen también posibilidades de modificar el código vía refactorización [14], transformación de programas [13], y evaluación parcial [10] (cuando se requieran de especializaciones de programas sobre computadoras con recursos limitados).

Para facilitar los diferentes dominios de posible aplicación, SCEL es *paramétrico* con respecto a algunas decisiones de posible implementación, tales como el lenguaje para expresar las políticas, los predicados regulando la interacción y agrupamiento (*ensembles*) entre componentes (con *coaliciones* cuando se realizan acciones con un objetivo en común), y la implementación de "conocimiento". SCEL no tiene un compromiso (por su alto nivel), en particular, con un lenguaje de programación específico. Queda a cargo del programador seleccionar las herramientas más adecuadas para el tratamiento de cada instancia. Por ejemplo, se podría utilizar un lenguaje tal como Python para programar las políticas, Prolog para el tratamiento del conocimiento, y Erlang para implementar la interacción distribuida entre componentes vía paso de mensajes. Más aún, Arduino podría proveer el acceso a los sensores y actuadores de los robots.

Aún más, como es descrito en [11], una componente podría tener algunas partes faltantes (o funcionalidades indicadas, pero no implementadas) y aún así ser completamente operacional como sistema. En escenarios donde los componentes podrían fallar y eventualmente quedar inoperantes, se podría delegar trabajo y funciones a otros componentes que cubrirían al componente faltante. En Erlang, esto es posible naturalmente vía el reemplazo de código en ejecución.

A continuación describimos por qué los programas en Erlang, basados en nociones de procesos infinitos en espera, son buenos candidatos para la formulación de la comunicación descrita en SCEL con *paso de mensajes*, *vectores de información*, y *satisfactibilidad de predicados*.

4. Erlang y los componentes autónomos de SCEL

De entre los lenguajes existentes actualmente, Erlang es un buen candidato para implementar el diseño teórico de un sistema y sus componentes siguiendo las directivas de SCEL. No obstante, debido al nicho de aplicación de otros lenguajes, no se excluye que exista partes de código escritas en su forma especializada para satisfacer requerimientos específicos. Para este fin, Erlang facilita la comunicación con otros lenguajes mediante filas de bits, y en particular, de utilizar, por ejemplo, Prolog, Prolog y Erlang interactuarían independiente en un solo nodo mediante un canal virtual construido mediante un puerto de comunicación serial (tal como ya hicimos en un experimento). Para el caso de comunicación remota la tecnología de *bluetooth* y *wi-fi* está suficientemente madura, y es económica, para la satisfacción de algunas necesidades de comunicación inalámbrica, aunque habría que valorar la distancia física cubierta contra el consumo de energía o la versatilidad en la estructuración y compartición de datos.

Cuando fuera necesario compartir recursos, se pueden programar agendas básicas ya sea locales o globales. En el caso en que SCEL enmarca el conjunto de políticas de uso de recursos, estas políticas están descritas mediante sistemas de transición etiquetados. La parte de depósitos de conocimiento (*knowledge*

repositories) podría tratarse con ETS y DETS (que son tecnologías de bases de datos pertenecientes al sistema OTP auspiciado por Erlang). En efecto, en OTP algunos módulos son específicos para el tratamiento intensivo de datos, tales como Amnesia. El tratamiento para estas bases de datos quedaría supervisada, nuevamente, por las políticas en boga dentro del marco teórico de SCEL.

A continuación describimos el conjunto de *abstracciones* que ligan a un nodo de Erlang (y sus procesos) con un componente autónomo descrito en SCEL.

1. Primero, describimos a nuestro componente autónomo como un *robot*: el robot está equipado con dispositivos para permitirle movilidad, relativa independencia energética, sensores, actuadores, y dispositivos de comunicación; bajo estas circunstancias, nuestro robot queda definido por la siguiente seudo-ecuación: robot = software (procesamiento, políticas, conductas, acciones, conocimiento) + hardware (sensores, actuadores)
2. Segundo, y específicamente, ahora consideramos que la parte esencial de procesamiento en software es llevado a cabo por un nodo de Erlang, con los procesos que sean necesarios poner en ejecución al robot. Para este fin, consideremos esencial que haya al menos un proceso inicial que bien puede apoyarse en automonitoreo o replicación para permitir la tolerancia a fallas; este proceso especial es nombrado `PSRobot`.
3. Adicionalmente al núcleo de programación en Erlang, es posible complementar las necesidades de programación con lenguajes específicos, tales como Prolog para el caso de la creación de planes, Arduino para las interfaces con los dispositivos sensoriales y actuadores, y Java si se requiere soporte vía bibliotecas para el manejo de comunicación mediante *wi-fi* o *bluetooth*.

La programación del núcleo de procesamiento del robot es por medio de la función `ac()`, y este núcleo es programado como ciclos eternos (*forever-loops*) para permanecer en un estado vigilante, y, en su caso y ya siendo parte de un proceso Erlang, activar las funciones asociadas al envío y recepción de mensajes. Los componentes autónomos pueden tener algunas propiedades auto-*; por ejemplo, el monitoreo propio del estado de energía, la auto-detección de fallas para el caso de auto-reparación (o al menos, para reportarse como inhabilitado), y la auto-preservación en ambientes riesgosos, adversos o peligrosos (por ejemplo, con un sensor de temperatura el robot se alejaría de una fuente de calor).

De las partes esenciales de estado de latencia de un robot, existe la construcción de Erlang `receive`, que trata diversos casos de recepción de mensajes mediante casamiento de patrones; este mecanismo es lo suficientemente versátil para permitir diversos tipos de entrada asociando *indicadores* (*tags*) a entradas numéricas, estructurando la información mediante vectores o bien diccionarios. Es importante en esta parte de programación tratar todos los casos exhaustivamente, con un uso bien planeado de las estructuras *atrapa-todo* (*catch-all*) que en ocasiones dificultan seriamente la depuración de programas en Erlang. En SCEL, el planteamiento teórico para una *interfaz* es mediante un conjunto de vectores, donde cada entrada señala un estado o bien un valor. La lectura de las entradas de este vector, como ya mencionamos, es manejada vía las políticas

en boga. Bajo una jerarquización armoniosa, es posible que un robot escriba también en la interfaz del vector de otro robot. Las entradas de los vectores también tienen otra función definida en SCEL: la de formación de grupos vía *publicaciones* (*broadcasting*). Esta formación de grupos mediante indicadores es una forma rápida y precisa de llevar a cabo planes mediante coaliciones: aquellos grupos que deberían trabajar en conjunto para lograr un objetivo. Por ejemplo, supongamos el diccionario: **{**activo:A, energía:E, tarea:T**}** señalando que el robot se halla activo (A puede ser activo o inactivo), con un nivel de energía E, y realizando en ese momento una tarea T, en donde la tarea *ocioso* (*idle*) se toma como no haciendo ninguna tarea. Supongamos que un robot requiere formar un grupo con aquellos elementos activos con un nivel de energía mayor del 50 % y estando ociosos. La petición se realiza mediante publicación de una solicitud y para cuando pasa un tiempo prudente los robots con las características solicitadas estarían listos para formar parte de un plan y llevarlo a cabo, con un objetivo común. El robot proactivo (quien convocó al grupo) puede decidir no llevar a cabo nada si, por ejemplo, no existen tantos robots como se esperaba. Dado que este mismo mecanismo de obtención de consenso podría llevarse a cabo concurrentemente, se puede optar por priorizar los objetivos y el mismo robot proactivo pasar a formar parte de un grupo.

Este mecanismo de interfaces es manejado por un conjunto de políticas. Dada una política, se estaría en posición de permitir que entre los robots se compartan información o no. El ejemplo a tratar sería el de un robot que esta *hackeado* e intenta confundir al sistema en su conjunto. Para esto, la política verificaría el número de accesos intentados de formación de grupos, un certificado de confianza por medio del robot proactivo, y un historial acerca de pautas de comportamiento para catalogar a un robot como estando en funcionamiento normal o anormal.

El diseño de una interfaz es particionada de acuerdo con un criterio basado en obtener información por cada uno de sus atributos de algunas posibles fuentes. En SCEL se utilizan predicados además de atributos para obtener grupos. Los predicados pueden a su vez formar fórmulas lógicas (o ecuaciones), y éstas deben de satisfacerse para el caso de formar grupos. Notemos también que la recepción de información no requiere necesariamente una respuesta como mensaje: tal información puede activar un actuador o, hasta en un caso extremo y por razones de seguridad, apagar al robot mismo.

Algunas acciones descritas directamente en SCEL están relacionadas con el envío y la recepción de mensajes. Esto es conveniente en términos de la contraparte de Erlang. Mediante tuplas (o diccionarios) y sus entradas, esta emisión y recepción de mensajes toma un papel preponderante. Mediante predicados, como fue ya comentado, las exigencias en la formación de grupos varía desde la comunicación entre pares (*peer-to-peer*) hasta la comunicación grupal mediante publicaciones (*broadcasting*) (orientada-a-grupo). Los filtros correspondientes se formulan mediante la inspección y verificación de los atributos así como la satisfacción de algunos predicados, para que la información llegue a un determinado número de robots. Si, aún más, existen permisos para explorar las bases de datos (conocimiento) de los robots, emergen complejas formas de interacción así como

objetivos de largo alcance que se verán a su vez reflejados en las secuencias de acciones que se planeen para lograr tales objetivos. En el trabajo en proceso que estamos realizando, por el momento, estamos enfocados más en el tratamiento reactivo de un robot, y hemos diferido el tratamiento de planes grupales (parte racional de un robot).

5. Nodos de Erlang y procesos en robots

En esta parte continuamos con una iteración de precisión de los anteriores conceptos por medio de "seudo-código" en Erlang. Afortunadamente, tal seudo-código está muy próximo a su implementación real. (Es necesario mencionar que "proceso" es una palabra que se utiliza bastante en Erlang pero también en los fundamentos teóricos de SCEL (cálculo de procesos).) Ahora analizamos el proceso `PSRobot`. El código mostrado en la Tabla 1 está destinado a casar muy cercanamente a las características de un robot en el escenario ejemplificado en [12], en hemos nombrado a una función `ac()` como la principal en el manejo de las funciones del robot. Notemos que hemos enfatizado el carácter *autónomo* en el diseño del robot, aunque un mensaje urgente de asistencia humana sería un buen último recurso.

Aparte de algunas funciones auxiliares definidas en otro lugar (y apoyadas quizás en otros lenguajes, tales como C, Python —para el caso de un micro-controlador Raspberry—, o Arduino). La pieza de código en Erlang debería casar bien en un robot en general a sus respectivas instancias de posibilidades, dispositivos, y actuadores, y de preferencia con un mínimo de asistencia humana.

Según la descripción de alto nivel de SCEL, los componentes autónomos deben ser lo suficientemente versátiles para permitir actividades de auto-configuración: desde jerarquías de diversa índole entre los componentes, agrupamientos de comunicación (*peer-to-peer*, comunicacion grupal), acuerdos consensados, enjambres (*swarms*), o la formación de coaliciones para cooperación o defensa. De manera importante, las conductas pueden redefinirse de acuerdo con los objetivos, y estos a su vez pueden cambiar, auto-adaptándose, de acuerdo con las circunstancias. A través de los atributos e interfaces, los componentes autónomos pueden alterar su configuración jerárquica (si existe) en aras de salvaguardar el objetivo principal del sistema como un todo. La decisión de si el mismo software estará en todos los robots es paramétrica de las posibilidades de que este software sea adaptable a diversas situaciones. No excluye la posibilidad, desde luego, de escribir programas que utilicen las capacidades especiales que un robot pudiera tener, aunque en nuestra simulación se está trabajando con robots que tienen exactamente los mismos componentes de hardware (con una suposición de homogeneidad en capacidades). En la parte restante de este trabajo queda por ver cómo Erlang brinda satisfacción concreta (con codificación asociada) a los requerimientos previos.

El programa mostrado en la Tabla 1 tiene una seudo-codificación en Erlang, con funcionalidades previsibles de acuerdo con el equipo de hardware que contamos, así como otras herramientas accesorias (varios procesos, microcontroladores

de Arduino y micro-computadoras Raspberrys). Desde este programa en seudo-código es posible obtener diversas instancias que permitirían una implementación efectiva de robots o inclusive de otras instancias de componentes autónomos (tales como agentes virtuales). En el caso de una simulación robótica, es posible crear incluso procesos que simulen el ambiente en donde los robots coexisten, así brindando un mayor realismo para mejor comprensión del sistema distribuido resultante.

Tabla 1. Listado de un programa en Erlang siguiendo directivas de SCEL.

```
1  ac() −> receive
2          {From, Qattributes} −> policy(From,attributes,Answer),
3                  if
4                          Answer==yes −> From ! attributes, ac();
5                          Answer==no −> From ! ‘‘Denied access’’, ac()
6                  end;
7          {From, Item} −> policy(From,Item,Answer),
8                  member(Item,attributes),
9                  if
10                         Answer==yes −> From ! yes, ac();
11                         Answer==no −> From ! ‘‘Denied access’’, ac()
12                 end;
13         {From, sensorsInput} −> policy(From, TypeofSencor,Answer),
14                 if
15                         Answer==yes −> From ! knowlege(sensors(type,input)), ac();
16                         Answer==no −> From ! ‘‘Denied access’’, ac()
17                 end;
18         {From, sensorsOutput} −> ....; %Idem;
19         {To, knowledge, acNew} −> knowledge(add(To,listTrust));
20         {To, knowledge,rmOld} −>
21                 knowledge(remove(To,listTrust)),
22                 knowledge(add(To,listDeny));
23         {gps,knowlege, gpsNew} −> knowlege(add(gps(Measures))); %Examples
24         {battery,knowlege, batteryLevelNew} −>
25                                 knowlege(add(batteryLevel(Measures)))
26         end.
27 policies (yes) −> listTrust ();
28 policies (no) −> listDeny().
29 knowledge(add,Items) −> add_list(Items);
30 knowledge(remove,Items) −> remove_list(Items);
31 knowledge(consult,Items) −> consult_list(Items).
32 attributes () −> list_of_attributes .
33 ensemble() −> .... %Some constraints to be satisfied to set up ensembles...
34 sensors(Input) −> obtain_values(Internal,Input);
35 sensors(Input) −> obtain_values(External,Input);
36 sensors(Output) −> send_value(Internal);
37 sensors(Output) −> send_value(External).
```

Cada CA tiene varias propiedades auto-*. Cada propiedad debería ser codificada como una función en Erlang, aunque con posible soporte de funciones externas para el manejo de periféricos, actuadores y otros dispositivos. De preferencia, existe una centralización del control del robot, aunque esto es preliminar en el diseño. Actualmente, por ejemplo, existen actuadores con sus propios procesadores (microcontroladores), y en este sentido, con un grado considerable de autonomía por sí mismos. Erlang tiene posibilidades de interactuar con los procesadores de estos actuadores debido a sus capacidades inherentes de concurrencia. Estamos investigando técnicas para que Erlang controle todos los procesadores disponibles mediante una programación de alto de nivel.

El componente de software aquí presentado es el resultado de varias iteraciones, como ya se ha mencionado, comenzando con las definiciones de SCEL, y siguiendo fielmente cada característica de un componente autónomo descrito por el formalisma. En cada iteración se ha llenado un "hueco" de precisión en código, haciendo eco de paradigmas de programación incremental y transformacional.

Nuestro escenario es una región planar con varios objetos diseminados para ser recolectados. Hay algunos obstáculos que sortear, y al menos dos sensores para este fin: uno ultrasónico y una cámara, los cuales son requeridos para la ubicación de los objetos. Los robots deambulan en la región de forma autónoma y aleatoria. Varias marcas son colocadas en el piso, y éstas son detectadas por los robots. Dado un grupo dedicado a la recolección (o al menos, detección) de estos objetos, la comunicación entre los elementos del grupo fluye rápidamente, para obtener consensos fortalecidos y evitar trabajo redundante.

6. Conclusiones

Obtenemos las siguientes conclusiones del trabajo presentado:

1. Seguimos la filosofía de desarrollo de programas de transitar tersamente de una especificación de alto nivel a una implementación orientada a un lenguaje de programación [13];
2. hemos instanciado uno de los parámetros del formalismo SCEL [12] en la parte de un lenguaje de programación seleccionado, ya que proponemos que Erlang, un lenguaje funcional con semántica bien identificada matemáticamente [6,15], sea el lenguaje de programación central en las aplicaciones distribuidas que un componente autónomo requiere;
3. aunque en etapa de diseño y con una propuesta todavía por perfeccionar, nuestro enfoque para la aplicación robótica es asequible yrealista, al utilizar tecnología de microprogramadores actuales así como la completa provisión de funciones de concurrencia preconstruidas en Erlang;
4. SCEL no tiene un compromiso explícito con un modelo computacional: aquí adoptamos el de concurrencia asíncrona con pasos de mensajes y sin recursos compartidos, modelo que es naturalmente dado en Erlang, con el conocimiento de que otros modelos computacionales podrían utilizarse también, cuando sea requerido.

En un trabajo a futuro, dotaremos con mayores capacidades sensoriales y de razonamiento a los CAs (tales como el razonamiento temporal [2,9]). Para ello, será necesario implementar algunos algoritmos de bases de datos y tratamiento de tiempo, sin menoscabo en los tiempos de reacción. La implementación en hardware tiene ya una factible realidad, pero es de notar que los CAs pueden tener otras aplicaciones a la aquí presentada; por ejemplo, pueden ser agentes que asistan de forma "personalizada" a estudiantes y profesores en un sistema de enseñanza/aprendizaje a distancia, como ya fue presentado, auxiliándose con varias otras herramientas, en [7], o bien en un sistema distribuido que gestione actividades colaborativas entre expertos.

Agradecimientos. Agradecemos las facilidades brindadas por la Universidad Tecnológica de la Mixteca para llevar a cabo la realización de este trabajo. El Dr. Manuel Hernández agradece al Dr. Andrés Fraguela Collar su aprecio y confianza a través de los años.

Referencias

1. Aceto, L., Ingólfsdóttir, A., Larsen, K.G., Srba, J.: Reactive systems. Cambridge (2007)
2. Aguilar-López, J.Y., Trujillo-Romero, F., Hernández, M.: Comunicación entre agentes inteligentes mediante cálculo de eventos usando Erlang. Research in Computing Science pp. 151–165 (2013)
3. Armstrong, J.: Programming Erlang: Software for a concurrent world. The pragmatic programmers (2007)
4. Bird, R., Wadler, P.: An introduction to Functional Programming. Prentice-Hall (1988)
5. Cesarini, F., Thompson, S.: Erlang programming. O'Reilly (2008)
6. Claessen, K., Svensson, H.: A semantics for distributed Erlang. In: ERLANG '05: Proc. of the 2005 ACM SIGPLAN workshop on Erlang. pp. 78–87. ACM (2005)
7. Cortés, H., García, M., Hernández, J., Hernández, M., Pérez-Cordoba, E., Ramos, E.: Development of a distributed system applied to teaching and learning. In: ERLANG '09: Proc. of the 8th ACM SIGPLAN workshop on ERLANG. pp. 41–50. ACM (2009)
8. Hennessy, M.: A distributed Pi-Calculus. Cambridge University Press (2007)
9. Hernández, M.: Event Calculus for Reasoning about Erlang Systems. 2013 12th Mexican International Conference on Artificial Intelligence 0, 29–35 (2011)
10. Jones, N.D.: An introduction to partial evaluation. Association for Computer Machinery (ACM) Computing Surveys 28(3), 480–503 (September 1996)
11. Nicola, R.d., Lluch-Lafuente, A., Loreti, M., Morichetta, A., Pugliese, R., Senni, V., Tiezzi, F.: Programming and verifying component ensembles (8415) (2014)
12. Nicola, R.D., Loreti, M., Pugliese, R., Tiezzi, F.: A formal approach to autonomic systems programming: The SCEL language. ACM Trans. Auton. Adapt. Syst. 9(2), 7:1–7:29 (Jul 2014), http://doi.acm.org/10.1145/2619998
13. Partsch, H.: Specification and Transformation of Programs. Texts and Monographs in Computer Science, Springer-Verlag (1990)

14. Sagonas, K., Avgerinos, T.: Automatic refactoring of Erlang programs. In: PPDP '09: Proc. of the 11th ACM SIGPLAN Conf. on Principles and practice of declarative programming. pp. 13–24. ACM (2009)
15. Svensson, H., Fredlund, L.Å.: A more accurate semantics for distributed Erlang. In: ERLANG '07: Proc. of the 2007 SIGPLAN workshop on ERLANG Workshop. pp. 43–54. ACM (2007)
16. Varela, C., Abalde, C., Castro, L., Gulías, J.: On modelling agent systems with Erlang. In: Proceedings of the 2004 ACM SIGPLAN workshop on Erlang. pp. 65–70. ERLANG '04, ACM, New York, NY, USA (2004), http://doi.acm.org/10.1145/1022471.1022481