Implementación y análisis comparativo de una arquitectura NBED- IO (Non-blocking Event-Driven I/O) para una plataforma de telemetría.

Por: Alejandro Corredor
Director: Héctor Cadavid Rengifo

Tabla de Contenido

1. Contexto (Proyecto)

1.1 Planteamiento del problema y pertinencia del mismo

De acuerdo con un reporte reciente del Observatorio Nacional de Salud (ONS)[1], en los últimos años las enfermedades cardiovasculares han aumentado continuamente su porcentaje -dentro del grupo de enfermedades no transmisibles- como causales de muerte a nivel mundial, estimando que entre el 2010 y el 2020 dicho porcentaje alcance un 73%, que equivaldrá a unas 44 millones de muertes en el mundo. Esta situación ha impulsado el desarrollo de áreas de investigación como la salud móvil y la telemetría, orientadas al monitoreo preventivo de las señales cardíacas de pacientes en riesgo, en la que convergen la electrónica (en la creación de sensores corporales confiables) y la computación (para el análisis y el procesamiento de las señales capturadas).

Un enfoque importante dentro de estas áreas de investigación ha sido la creación de esquemas de monitoreo remoto más asequibles por parte de los pacientes, haciendo uso de los ahora muy populares 'teléfonos inteligentes', los cuales tiene tan sólo una fracción del costo de los tradicionales dispositivos tipo *holter*. En diversos trabajos realizados en dicha área[2345], se plantean arquitecturas en las que un dispositivo móvil (en principio el

---

[1] Boletin 1, Diciembre 9 de 2013 - Observatorio NacCardiovascular Colombia - Enfermedad Cardiovascular - http://www.ins.gov.co/lineas-de-accion/ons/boletin%201/boletin_web_ONS/boletin1.html. **2013**

[2] Simunic, D.; Tomac, S. & Vrdoljak, I.
Wireless ECG monitoring system
Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology, 2009. Wireless VITAE 2009. 1st International Conference on, IEEE, **2009**, 73-76

[3] Chan, C.-C.; Chou, W.-C.; Chen, C.-W.; Ho, Y.-L.; Lin, Y.-H. & Ma, H.-P.
Energy efficient diagnostic grade mobile ECG monitoring
New Circuits and Systems Conference (NEWCAS), 2012 IEEE 10th International, **2012**, 153 -156

[4] Saponara, S.; Donati, M.; Bacchillone, T.; Fanucci, L.; Sanchez-Tato, I.; Carmona, C. & Barba, P.
Remote monitoring of vital signs in patients with Chronic Heart Failure: Sensor devices and data analysis perspective
Sensors Applications Symposium (SAS), 2012 IEEE, **2012**, 1-6

[5] Gabriel, V. & Fedor, L.
Modern approach in multiple patients ECG monitoring
Biomedical and Health Informatics (BHI), 2012 IEEE-EMBS International Conference on, **2012**, 131 -134

teléfono celular del paciente) es utilizado para recibir las señales de un dispositivo electrónico adherido al paciente (red de área corporal), para posteriormente procesarlas, visualizarlas, y retransmitirlas a un servidor central en el que se realizan procesamientos adicionales.

Sumándose a estas líneas de investigación, los programas de Ingeniería de Sistemas e Ingeniería Electrónica de la Escuela Colombiana de Ingeniería desarrollaron conjuntamente una plataforma de telemetría para señales cardíacas bajo el esquema anteriormente planteado[6], teniendo como elemento diferenciador un enfoque de 'marco de trabajo', es decir, en el que fácilmente se pueden incorporar y modificar componentes. Con el enfoque de 'marco de trabajo', la plataforma permite, por ejemplo, incorporar fácilmente nuevos algoritmos de procesamiento de señales, y encadenarlo a otros ya existentes, de manera que ésta pueda usarse tanto en entornos académicos (para la experimentación con nuevos algoritmos) o en entornos reales, con pacientes para cuyas características (étnicas, sociales y demás) se tenga un conjunto de algoritmos de procesamiento orientados a una detección de riesgos más precisa.

Como resultado del proyecto anteriormente mencionado, se identificaron dos problemas importantes para su puesta en producción en el último escenario planteado (en producción, para una comunidad de pacientes):

1. El alto costo computacional de los algoritmos de procesamiento, hace que con la incorporación de cada uno de éstos (dentro de la secuencia de algoritmos que se aplica a cada paciente) se reduzca la capacidad de la plataforma, en cuanto a número de señales simultáneas a procesar.

2. A diferencia de una plataforma Web convencional, en la que las peticiones recibidas por el servidor son esporádicas (generadas por las acciones del usuario), en la plataforma de telemetría se debe dar un soporte continúo a decenas o centenares de clientes que envían continuamente datos. El solo hecho de mantener tantas conexiones abiertas implica una carga de procesamiento muy alta para los servidores de aplicaciones convencionales, ya que deben mantener un hilo de procesamiento por cada conexión.

[6] *Congreso Internacional en Ingeniería Clínica y Biomédica, Bogotá, Colombia, May 8-9, 2014. Ieee Engineering In Medicine And Biology Society / ISBN: 978-1-4799-5235-9,* **2014**

Dado que lo lógico para resolver los problemas antes mencionados es buscar un mecanismo de escalabilidad para la aplicación, de manera que se pueda distribuir la carga de la aplicación entre diferentes instancias de la misma, se planteó el diseño y desarrollo de una arquitectura alternativa para el *backend* de la plataforma de telemetría que (1) fuera más fácil de escalar, y (2) mitigara el problema de los servidores de aplicación tradicionales en los que se requiere un hilo de procesamiento por petición. Para esto, el proyecto presentado en este artículo plantea la evaluación del modelo de 'entrada/salida no bloqueante manejada por eventos', y su realización a través de un modelo de actores para el caso concreto de un sistema de telemetría.

1.2 Marco teórico y estado del arte.

1.2.1 Modelo de aplicación por eventos no bloqueantes

En contraste con el modelo de aplicación basado en hilos que requieren sincronización, el modelo de aplicación por eventos no bloqueantes se caracteriza por manejar un único hilo para múltiples conexiones. Como se observa en la figura, bajo este modelo el servidor encola todas las peticiones y las procesa consecutivamente (con ese único hilo) de manera asíncrona (es decir, al recibir cada petición inmediatamente queda disponible para una nueva).

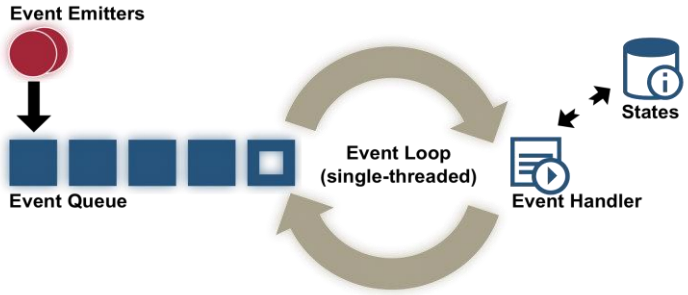
Con el modelo de eventos, el control de flujo de una aplicación es invertido -respecto al modelo imperativo-, pues en lugar de operaciones secuenciales definidas por el programador, éste se limita a invocar a los manejadores de los eventos recibidos. De esta manera, en lugar de tener un centenar de hilos compitiendo continuamente por una ventana de tiempo en el procesador (lo cual en sí mismo es costoso por los bloqueos que se dan con la sincronización de múltiples hilos), se apuesta por manejar controladamente dichas ventanas de tiempo de procesamiento, al dedicarlas a la ejecución consecutiva de los manejadores de los eventos recibidos.

Aunque con este modelo se evitan los tiempos de latencia que se dan con los bloqueos de los hilos, y se simplifica la programación respecto al modelo imperativo (la aplicación se define únicamente en términos de los manejadores de los eventos), tiene como inconveniente que resulta más difícil de seguir y de depurar el flujo de las aplicaciones[7].

1.2.2 Implementaciones del modelo de eventos no bloqueantes

Aunque el concepto de programación por eventos existe desde hace mucho tiempo, sólo hasta hace unos años, con el vertiginoso aumento del nivel de exigencia a los servicios prestados por Internet por parte de los usuarios, y el surgimiento de clientes alternativos tales como smartphones y tabletas, se planteó usar este modelo para atender más eficientemente -bajo ciertas condiciones- a un mayor número de clientes.

La plataforma *Node.js* fue pionera en la implementación del modelo de eventos no bloqueantes para la Web, y ha mostrado ser eficiente en aplicaciones de alto desempeño cuando se combina con bases de datos NoSQL[8][9]. Sin embargo, el estar basada en el motor V8 de Google Chrome[10], y por ende en el modelo de programación de *javascript* (el cual no es considerado muy robusto), ha llevado a plantear modelos equivalentes basados en lenguajes funcionales, combinando de esta manera las bondades del modelo de eventos con el rigor que exigen



---

[7] Hohpe, G.
Programming Without a Call Stack: Event-driven Architectures
Objekt Spektrum, **2006**

[8] McCune, R. R.
Node. js Paradigms and Benchmarks
STRIEGEL, GRAD OS F, **2011***, 11*

[9] Paudyal, U.
Scalable web application using node. js and couchdb
2011

[10] Dahl, R.
Node. js: Evented I/O for V8 JavaScript
2012

dichos lenguajes.

En este sentido, los leguajes funcionales Scala y Haskell han tomado mucha relevancia al ser la base de plataformas basadas en eventos de creciente popularidad: *Akka Vert.x*, en el caso de Scala, y *Yesod* y *Happstack*, para el caso de Haskell.

Como criterio para elegir con qué plataforma implementar la prueba de concepto (un *backend* para la plataforma de telemetría basada en eventos no bloqueantes), se tuvo en cuenta en primera instancia la cantidad casos de uso, la documentación disponible y la comunidad de usuarios que pudieran dar soporte. En este sentido, la selección se redujo a las opciones de Akka y Node.js. Para esta segunda instancia, se consideraron las facilidades de escalabilidad, tras lo cual se optó por Akka, y su modelo de actores distribuidos[11].

1.2.3 Akka y el Modelo de actores

El modelo de actores, propuesto por Hewitt, Carl y Bishop[12] en 1973, plantea una forma de definir aplicaciones como una composición de entidades 'ligeras' y concurrentes denominadas actores. Dichos actores son independientes entre sí (no comparten estados) y pueden comunicarse, de forma asíncrona, mediante un mecanismo de 'buzón' de correo. Este modelo cumple con la filosofía de los 'eventos no bloqueantes', dado que cada actor, usando un único hilo de procesamiento (el cual puede ser compartido por varios actores), procesa eventos recibidos en su 'buzón' sin necesidad conocer el estado de los demás (es decir, sin considerar ningún tipo de bloqueos por sincronzición).

Bajo este esquema, un actor puede, de manera concurrente[13] :

1. Enviar mensajes a otros actores. Los emisores de los mensajes son identificados por dirección, algunas veces llamadas dirección de correo. Así un actor puede comunicarse únicamente con los actores de los cuales posee la dirección
2. Crear nuevos actores.
3. Designar el comportamiento que se tendrá para el siguiente mensaje recibido.

---

11 Charousset, D.; Schmidt, T. C.; Hiesgen, R. & Wählisch, M.
Native actors: a scalable software platform for distributed, heterogeneous environments
Proceedings of the 2013 workshop on Programming based on actors, agents, and decentralized control, **2013**, 87-96

12 Hewitt, C.; Bishop, P. & Steiger, R.
A universal modular actor formalism for artificial intelligence
Proceedings of the 3rd international joint conference on Artificial intelligence, **1973**, 235-245

13 Charousset, D.; Schmidt, T. C.; Hiesgen, R. & Wählisch, M.
Native actors: a scalable software platform for distributed, heterogeneous environments
Proceedings of the 2013 workshop on Programming based on actors, agents, and decentralized control, **2013**, 87-96

La plataforma Akka, elegida para este proyecto, fue creada por Jonas Bonér[14]en el año 2009, como respuesta a la necesidad de simplificar la creación de aplicaciones distribuidas y escalables. Al día de hoy, es el modelo de preferencia en el caso Scala y su popularidad ha ido creciendo respecto a las plataformas tradicionales.

Aunque el lenguaje Scala cuenta con una implementación nativa del modelo de actores, Akka proporciona soluciones a varios de los retos que presenta una plataforma de procesamiento masiva como la que se pretende diseñar en este ejercicio. Las principales inclusiones adicionales más relevantes de Akka respecto al modelo de Scala son

- Supervisores: En Akka cada actor es supervisor de los actores que él cree (conocido con el término de hijos) y así mismo define una estrategia de supervisión de los mismos. Esta es la principal forma que tiene Akka para manejar la tolerancia a fallos, considerado esto esencial en una aplicación de procesamiento distribuido. En el caso de que alguno de los hijos de un actor falle es la estrategia de supervisión del actor padre la que decidirá entre:
    - Volver a poner en funcionamiento el actor hijo, manteniendo su estado previo, siempre y cuando no haya sido corrompido por el error generado.
    - Reiniciar el actor y a sus hijos, limpiando su estado interno
    - Detener el hijo permanentemente.
    - Detenerse a sí mismo y escalar el error al padre de este actor.
- Registro de actores (Actor Register): Como su nombre lo indica, este componente es el registro de los actores presentes en el sistema, el cual permite buscar y manipular los actores activos en el sistema (por ejemplo apagarlos). Los distintos parámetros por los cuales es posible buscar un artículo son:
    - identificador único (UUID) campo del actor que lo identifica frente a los demás.
    - id: campo único del actor accesible y modificable por el usuario
    - Clase: devuelve un nuevo actor por su clase(tipo de actor)

[14] Bonér, J.; Klang, V.; Kuhn, R. & others
Akka library
h ttp://akka. io,

- Subtipo: devuelve todos los actores los cuales son un subtipo del tipo de actor mencionado

1.3 Objetivo

Objetivo general:

Definir e implementar una arquitectura de software para una plataforma de procesamiento y análisis de señales biométricas bajo un modelo reactivo (no bloqueante), y comparar su desempeño con la arquitectura convencional basada en hilos bloqueantes.

Objetivos específicos:

- Apropiar los conocimientos del lenguaje Scala y el conjunto de herramientas AKKA, que permitan el manejo y procesamiento de peticiones asíncronas no bloqueantes

- Construir la especificación de la arquitectura y documentar, de forma detallada, las particularidades del lenguaje y herramientas antes mencionados.

- Analizar de forma comparativa el desempeño y la escalabilidad de la plataforma reactiva contra la implementación en el modelo convencional de hilos bloqueantes.

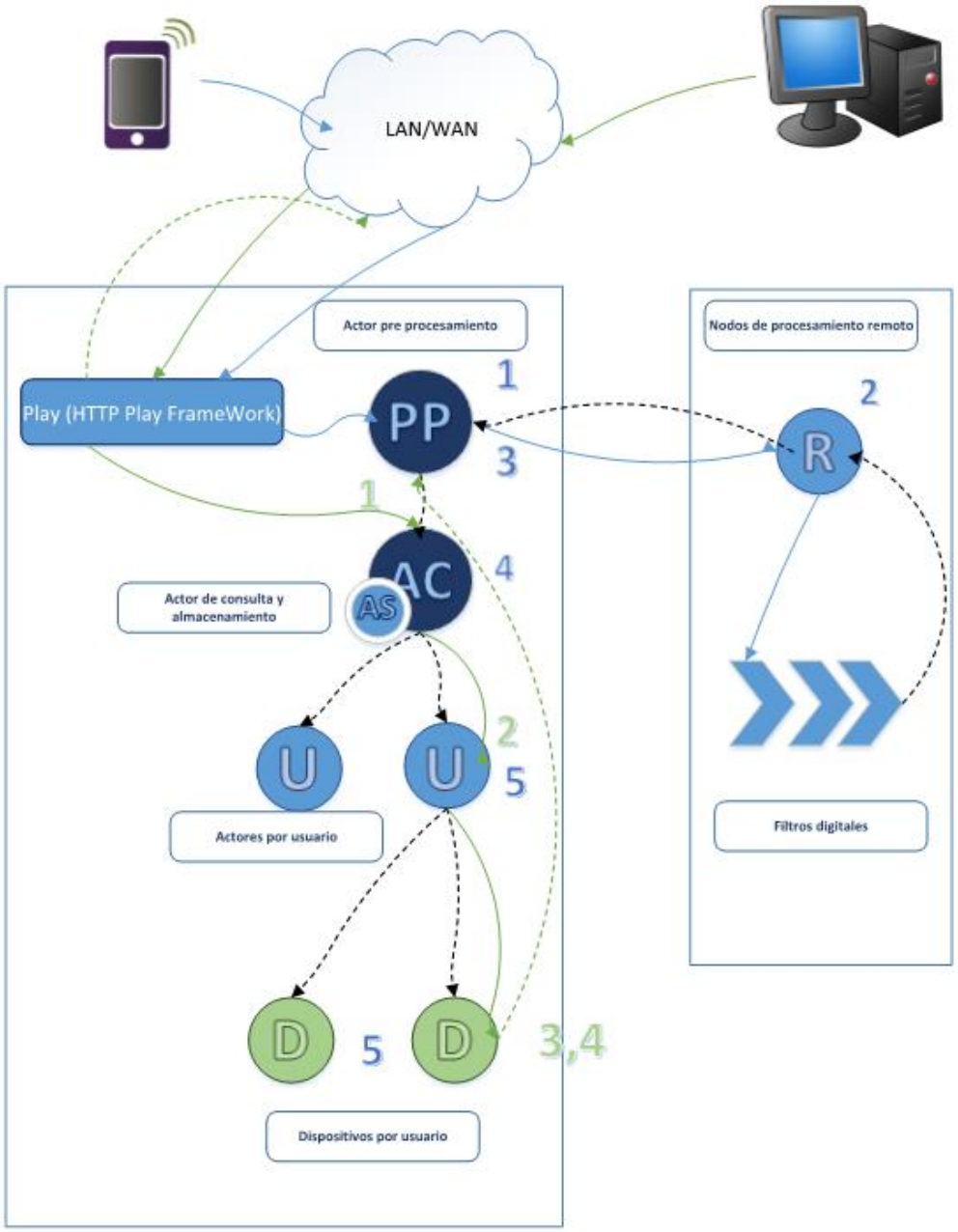
- Implementar una estrategia de escalamiento horizontal a través del sistema remoto de actores.

2. Requerimientos

2.1. Descripción del sistema

La arquitectura presentada en la figura a continuación representa el funcionamiento de la aplicación en dos escenarios:

- Escenario de transmisión y registro de la señal generada por el sensor ECG y transmitida por el móvil:
    - Se recibe la señal en un actor de pre procesamiento (1), se envía al actor de procesamiento, que devolverá la trama después de la aplicación de varios filtros. La trama procesada vuelve al actor de pre procesamiento para realizar el proceso de almacenado (3), posteriormente la trama es enviada al actor de consulta y guardado (4) y finalmente Si el paciente no tiene datos registrados en el sistema se crea un nuevo actor que represente dicho paciente y se almacenan los enviados. De estar registrado se actualizan todos los actores que representan a aquellos dispositivos consultando al paciente(5)
- Escenario de consulta de una señal procesada por parte de un cliente:
    - El actor de consulta recibe la petición del dispositivo móvil o navegador de consultar una petición (1), en caso de existir un paciente identificado con el identificador proporcionado se procede a enviar una petición de consulta a dicho actor (2).Estando en el actor representante del paciente, este obtiene el identificador del dispositivo que solicita la consulta de datos para determinar si ya está registrado. En caso de no estarlo se crea un nuevo actor que represente dicho dispositivo y se devuelve la última trama de datos registrada del paciente (3).Finalmente El actor representante del dispositivo devuelve al actor de consulta principal la trama de datos más reciente no consultada de dicho dispositivo. Entendiendo que este mantiene una lista de tramas que va siendo consumidas cada vez que el dispositivo en cuestión haga una consulta, manteniendo de esta manera la continuidad así halla perdidas de Internet o se corte la solicitud de datos por periodos cortos de tiempo (4).



2.2. Usuarios

Dentro del sistema solo existen dos tipos de usuario, siendo el principal el paciente; este transmite las distintas señales que se registraran en el sistema para ser consultadas por el segundo tipo de usuario. Este último puede ser desde un miembro del hospital hasta un familiar de un paciente.

2.3. Retos de la plataforma

Los principales retos de una plataforma de telemetría son:

-La capacidad de procesamiento: Dada la gran cantidad de solicitudes de registro de señales y de usuarios solicitando datos, este tipo de plataformas requieren un esquema de procesamiento eficiente y en la mayoría de casos distribuidos, es decir teniendo plataformas externas que se dediquen solo al procesamiento de señales o consultas de usuarios.

-Escalabilidad: el rápido crecimiento y aceptación por parte de los distintos tipos de usuarios en este tipo de plataformas, hace que la consideración de una forma de escalar el sistema conforme el número de usuarios y requerimientos aumenta es esencial

Adicionalmente a lo anterior, la aplicación en particular presenta los siguientes retos:

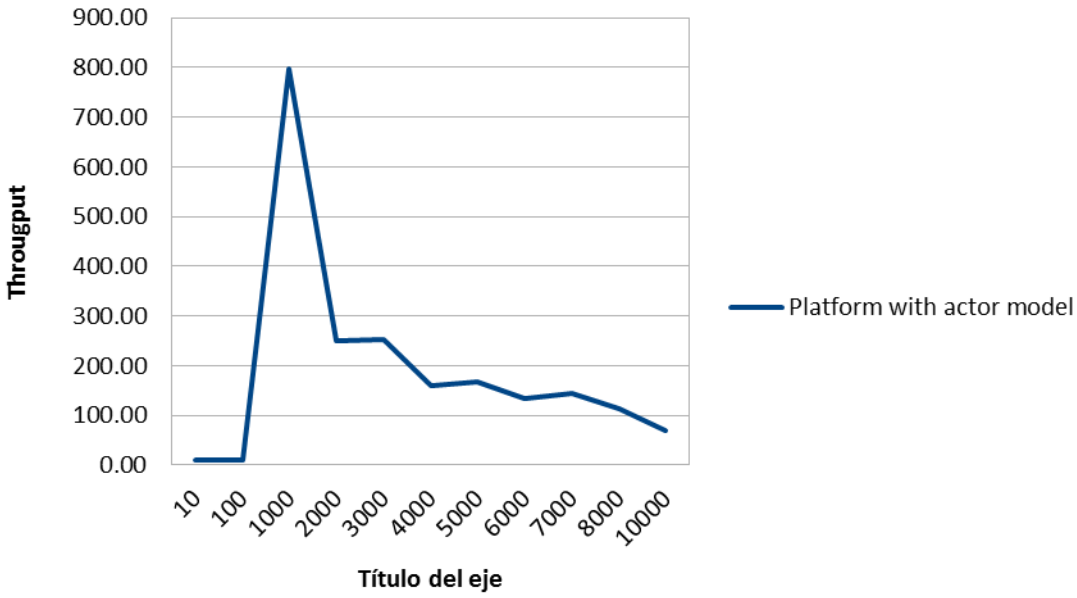
- Continuidad en la consulta de señales: La competencia de los distintos usuarios que consultan por obtener la señal de un mismo paciente, hace que sea un reto mantener el flujo de datos constante, para que la señal solicitada se pueda observar de forma continua y no con cortes

- Pérdidas de la conexión a internet momentáneas: Dado que la mayoría de dispositivos que consultan este tipo de plataformas son móviles, las fluctuaciones de la señal es un hecho común al día de hoy, sin embargo eso se convierte en un reto a la hora de evitar la pérdida de información una vez el dispositivo retoma la conexión.

3. Pruebas

3.1. Especificación de pruebas de carga

Mediante un sistema automatizado de ingreso de datos el sistema se llena de registros el sistema con datos de tres pacientes diferentes. Por otra parte la aplicación de pruebas carga al sistema con distinta cantidad de peticiones de consulta simultánea por veinte segundos. Con lo cual se pueden observar los siguientes resultados:



4. Liberación

4.1. Herramientas

A continuación se detallan las herramientas y tecnologías usadas para el desarrollo de la aplicación:

-Scala: El lenguaje principal bajo el cual está programada la aplicación. Es un lenguaje de carácter funcional creado en el 2003 por Martin Odersky. Se caracteriza por usar el paradigma orientado a objetos, brindando todas las características de un lenguaje funcional, entre ellas siendo destacable la inmutabilidad.

Cabe destacar que dicho lenguaje usa al igual que java la JVM, sin embargo proporcionando un rendimiento y comprensión natural de lectura que el mismo Java gracias a sus características funcionales.

-AKKA: Se define como un conjunto de herramientas (toolkit) desarrollado en el 2006 tanto para Scala como Java, que les permite el uso del modelo de actores y la implementación de procesamiento distribuido, así como las estrategias para afrontar fallos.

-PLAY: Framework diseñado para la construcción de aplicaciones web con el modelo MVC(Modelo,Vista,Controlador), totalmente escrito en Scala y basado en AKKA para proporcionar rendimiento y facilidades al programador, como convenciones en vez de configuraciones, cambios de código en “Caliente” y la muestra de errores detallados en el navegador.