Caso práctico de un proceso de ingeniería inversa implementado sobre una arquitectura reactiva

Pablo J. Hernández López [1] [0000-0002-5802-1117], Noé A. Rodríguez González [1] [0000-0002-4661-1478], Alfonso A. Vitale Zamorano [1][0000-0003-0493-1083].

[1] Open Canarias, S.L. C/ Elías Ramos González, 4
Edificio Sovhispan, Oficina 304
38001 Santa Cruz de Tenerife, España.
{pablojhl, nrodriguez, avitale}@opencanarias.es

Abstract Los sistemas heredados son, habitualmente, de grandes dimensiones. A la hora de abordar un proceso de ingeniería inversa sobre estos sistemas, las herramientas para el tratamiento de modelos muestran problemas de rendimiento y escalabilidad. En este artículo se presenta la solución, propuesta por Open Canarias, para solventar este problema mediante el uso de los principios del manifiesto reactivo.

Keywords: Transformaciones de Modelos, MDA, Sistemas Reactivos, Escalabilidad.

1 Introducción

A lo largo de diez años, Open Canarias ha desarrollado herramientas basadas en el tratamiento de modelos y MDE [1] para el soporte al ciclo de vida del software y la automatización de procesos de modernización, siguiendo las directrices de MDA [2] y ADM [3], respectivamente.

Según la experiencia de la empresa, procesos que requieren la ejecución de un gran número de transformaciones modelo-a-modelo pueden presentar problemas de escalabilidad cuyo impacto se refleja en su rendimiento general [4]. Más aún, si los modelos sobre los que actúan son de gran tamaño y complejidad. Esto es especialmente cierto en herramientas de ingeniería inversa. Es por ello que se recurre al paradigma de los sistemas reactivos buscando fortalecer el escalado de la ejecución de dichos procesos.

Este artículo presenta la solución reactiva desarrollada por Open Canarias, cuyo principal objetivo es resolver los problemas de rendimiento identificados en herramientas de extracción o ingeniería inversa de grandes sistemas heredados. El documento estará organizado del siguiente modo: (1) Definición del problema, (2) Diseño de la solución, (3) Resultados y conclusiones y (4) Referencias.

2 Definición del problema

Las herramientas de modernización desarrolladas por Open Canarias pretenden la reducción de costes de mantenibilidad, extracción de conocimiento, soporte al ciclo de

vida de los desarrollos, etc. Sin embargo, aun sabiendo el valor que estas herramientas aportan a los clientes, el reto de enfrentarse a sistemas de grandes dimensiones (varios millones de líneas de código) ha provocado que estas herramientas sean difíciles de operar y con un rendimiento insuficiente para el usuario.

El objetivo de Open Canarias es disponer de una herramienta que presente información sobre la arquitectura, de una manera rápida e intuitiva, con la que el cliente pueda tomar decisiones sobre cómo abordar los procesos de modernización.

3 Diseño de la solución

En la consecución de los objetivos señalados en la definición del problema, se busca una solución que provea de escalado horizontal a las herramientas desarrolladas.

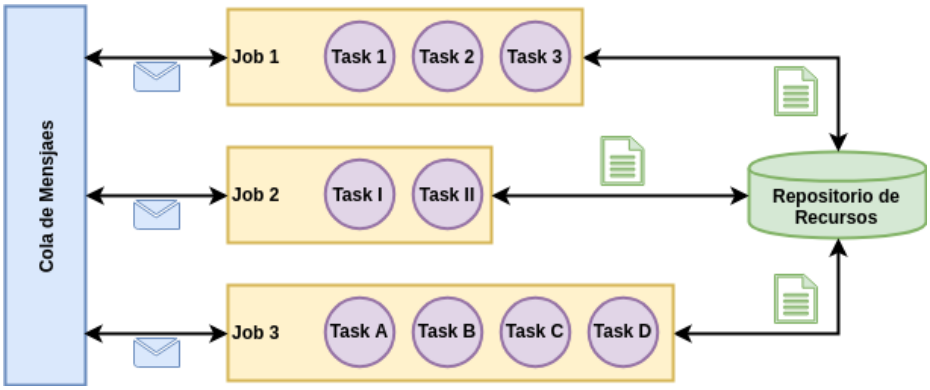

Fig. 1. Arquitectura de la solución planteada.

La Figura 1 muestra un esquema de la solución planteada cuya arquitectura está dirigida por mensajes. Los elementos fundamentales del sistema son:

- Mensajes: notificaciones que circulan por el sistema. Son utilizados para requerir ejecuciones de tareas, recursos o indicar el estado de los mismos.
- Cola de mensajes: almacenamiento y distribución de mensajes desde y hacia el sistema.
- Task / Tarea: unidad mínima de trabajo del sistema. Pueden componerse para aumentar su complejidad.
- Job / Trabajo: agrupación lógica de tareas que además gestiona el ciclo de vida de ejecución de las mismas.
- Recursos: datos calculados por las tareas, por ejemplo modelos.
- Repositorio de recursos: sistema de almacenamiento de recursos calculados por las tareas.

Cabe destacar que las tareas trabajan con la premisa de que todos los recursos, ya sean de entrada o salida, son inmutables, generando nuevas versiones de un recurso en

caso de que así se requiera. Este punto simplifica la solución, ya que no habrá cambios concurrentes sobre una misma versión de un recurso.

El sistema se conforma mediante la unión de un conjunto de Jobs que contribuyen tareas que pueden ser ejecutadas. Cuando un agente externo requiere la ejecución de una tarea, *Tarea X*, envía una petición a la cola de mensajes. El *Job 1* que conoce las tareas de tipo X captura el mensaje y comprueba la disponibilidad de los recursos requeridos por las mismas. En el caso de que estén disponibles, el *Job 1* ejecuta la *Tarea X*. En caso contrario, solicita los recursos que necesita, a través de la generación de nuevas peticiones en la cola de mensajes, y la *Tarea X* queda pendiente de ellos. Otro Job de los conectados a la cola, el *Job 2*, se da cuenta de que una de sus tareas, *Tarea Y*, puede calcular los recursos que se han solicitado. Captura los mensajes e inicia las tareas necesarias para la generación de recursos. Cuando una tarea finaliza se notifica la creación de los recursos calculados a través de mensajes. Esto permite que el *Job 1* retome la ejecución de la *Tarea X*.

Las tecnologías involucradas en el sistema son:

- Kafka [5] como cola de mensajes.
- Akka Actors [6] como librería para implementar la coordinación y ejecución de las tareas siguiendo el modelo de actores.
- HDFS (Hadoop Distributed File System) [7] como repositorio para almacenar recursos. El sistema permite la contribución de diferentes repositorios como sistema de ficheros local, etc.
- Cassandra [8] como base de datos para almacenar información adicional derivada de la ejecución de las tareas así como el registro de la ejecución de las mismas.

La arquitectura dirigida por mensajes y la asincronía del sistema permiten que el hardware sea aprovechado al máximo, ejecutando tareas de forma paralela en función de la capacidad de cómputo disponible. La contribución de nuevas instancias de un Job en nuevas máquinas permite que el sistema escale horizontalmente. Por otra parte, el sistema puede ser extendido en funcionalidad mediante la contribución de nuevos jobs que aporten tareas diferentes.

En el contexto de un proceso de ingeniería inversa, las tareas del sistema son transformaciones de modelos hacia descripciones abstractas. En este caso el sistema recupera modelos del repositorio, los transforma y almacena los resultados en el mismo. Destacan las siguientes ventajas:

- Ajuste a requisitos hardware de las transformaciones. Cuando se requiere de la ejecución de diferentes transformaciones de modelos, los requisitos hardware de cada una pueden ser diferentes. Pueden contribuirse recursos hardware en función de los requisitos de la transformación mediante la creación de instancias de Jobs. Transformaciones más complejas o pesadas tendrán, por tanto, un mayor número de instancias de Jobs.
- Paralelismo. Gracias a que se puede disponer de múltiples instancias de un Job, se consigue la ejecución de n transformaciones de forma simultánea.

4 Resultados y conclusiones

El sistema presentado en este artículo se ha probado en el contexto del análisis de un sistema heredado de más de 32 millones de líneas de código en lenguaje COBOL. Los ficheros de código fuente fueron analizados gramaticalmente (mediante un proceso de parsing) y transformados hasta su representación en KDM [9] previo cálculo de los modelos de sintaxis concreta y abstracta. Los intentos usando soluciones anteriores basadas en Eclipse no lograban finalizar por colapsar los recursos de la máquina en la que se ejecutaba. Las transformaciones se ejecutaban de forma secuencial y las dependencias entre ellas resultaban en esperas en las ejecuciones, desperdiciando recursos hardware. Además, el rendimiento estaba limitado por ejecutarse toda la secuencia de transformaciones en una única máquina. Tras el uso de la solución propuesta se logró realizar el análisis en 7 horas usando sólo 3 PC.

La solución se adapta al elevado consumo de CPU y memoria que tienen algunas de las tecnologías usadas en desarrollos que involucran el tratamiento de modelos, por ejemplo transformaciones de modelos implementadas con Eclipse QVTo.

Si bien el uso de HDFS no es la solución ideal para el manejo de ficheros pequeños [10] (menores del tamaño de bloque mínimo por defecto 64Mb), esto no fue problema para las pruebas realizadas ya que el consumo de CPU y memoria de las tareas era muy superior al coste en E/S. Aun así, la solución propuesta no está acoplada al uso de HDFS, por lo que podría adaptarse a repositorios con recursos de pequeño tamaño.

Este trabajo ha sido desarrollado bajo el paraguas de un proyecto CDTI con fondo FEDER.

5 Referencias

1. Douglas C. Schmidt: Model - Driven Engineering pp. 25 - 31. IEEE. Computer Society (2006).
2. OMG: MDA Guide Version 1.0.1, http://www.omg.org/mda, último acceso 03/04/2019.
3. Kshusidman, V.: Adm transformation. ADM Task Force. White Paper (2008).
4. Óscar Sánchez Ramón, Francisco Javier Bermúdez Ruiz y Jesús García Molina: Una valoración de la Modernización de Software Dirigida por Modelos. Departamento de Informática y Sistemas Universidad de Murcia (2017).
5. Apache: Apache KAFKA. https://kafka.apache.org, último acceso 03/04/2019.
6. Akka: Akka Actors. https://akka.io/docs/, último acceso 03/04/2019.
7. Apache: Hadoop Distributed File System (HDFS). https://hadoop.apache.org/, último acceso 03/04/2019.
8. Apache: Apache Cassandra. http://cassandra.apache.org/, último acceso 03/04/2019
9. OMG: Knowledge Discovery Metamodel (KDM) Version 1.4. Object Management Group (2016). https://www.omg.org/spec/KDM/, último acceso 03/04/2019.
10. Sachin Bendea, Rajashree Shedgeb: Dealing with Small Files Problem in Hadoop Distributed File System. Elsevier B. V. (2016).