CERVANTES: Un framework para el diseño y desarrollo de sistemas distribuidos

M.A. Barcelona[1,2], L. García-Borgoñón[1,2], J.I. Calvo[1], and M.J. Escalona[2]

[1] Instituto Tecnológico de Aragón, c/ María de Luna 7, 50018 Zaragoza,
mabarcelona@ita.es, laurag@ita.es, jicalvo@ita.es
[2] Universidad de Sevilla, Av. Reina Mercedes S/N, 41012 Sevilla,
miguel.barcelona@iwt2.org, laura.garcia@iwt2.org, mjescalona@us.es

Resumen En la actualidad vivimos una continua evolución en los sistemas informáticos que nos rodean: cada vez son de mayor tamaño, cubren más funcionalidades y es necesario que interactúen con otros sistemas ya existentes. El desarrollo de estos sistemas es por tanto una tarea compleja que sin embargo ha evolucionado a un menor ritmo. En los últimos años han surgido iniciativas como el desarrollo dirigido por modelos (*MDE*), la computación basada en servicios (*SOC*) o la ingeniería del software orientada a servicios (*SOSE*). En este trabajo se presenta una herramienta, basada en los conceptos de *MDE* y *SOC*, que desde el año 2002 se ha venido utilizando en la práctica con la industria para el diseño y desarrollo de sistemas distribuidos.

Palabras Clave *Model Driven Engineering*, *Service Oriented Computing*, *Service-Oriented System Engineering*

1 Motivación

En la actualidad vivimos una continua evolución en los sistemas informáticos que nos rodean: cada vez son de mayor tamaño, cubren más funcionalidades y es necesario que interactúen con otros sistemas ya existentes. El desarrollo de estos sistemas es por tanto una tarea compleja que sin embargo ha evolucionado a un menor ritmo. Para gestionar esta complejidad en los últimos años han surgido nuevos paradigmas e iniciativas como el desarrollo dirigido por modelos (*Model Driven Engineering, MDE*)[1], la computación basada en servicios (*Service-Oriented Computing, SOC*)[2] o la ingeniería del software orientada a servicios (*Service-Oriented System Engineering, SOSE*)[3]. Enmarcado en este complejo contexto, en el año 2001, antes de que surgiera un creciente interés por *MDE* y *SOC*, nos planteamos crear un marco que cubriera las siguientes necesidades:

- Debía servir para crear sistemas de sistemas en un entorno heterogéneo y distribuido.
- Se debía poder especificar un sistema de forma abstracta, sin entrar en los detalles de implementación.

- Debía facilitar la monitorización de los diversos subsistemas dentro del entorno distribuido en el que opera.
- Debía permitir describir semi-formalmente el comportamiento interno de cada subsistema, de forma que se pudieran añadir mecanismos de valiación y verificación incluso en tiempo de ejecución.
- Debía servir para crear sistemas que, dentro de una arquitectura orientada a servicios, se basaran en la composición de servicios.
- Se debía desarrollar una infraestructura de soporte y un entorno de desarrollo para facilitar su adopción de forma práctica en la industria.

2 Cervantes Framework

Tomando como base estas necesidades, se desarrolló una infraestructura básica para facilitar el despliegue de sistemas distribuidos, concebidos como subsistemas que intercambian información mediante mensajes asíncronos y los procesan mediante workflows como máquinas de estados. Su arquitectura básica se compone de los siguientes bloques, tal y como se muestra en la Figura 1 a):

- *Core Framework*: contiene los elementos comunes a toda la infraestructura como mecanismos de activación y parada, registro de métricas, manejo común de excepciones y logger.
- *Control Framework*: encapsula la lógica de actuación ante la llegada de un mensaje. Se compone de un gestor denominado *Manager* que implementa las funciones de un consumidor, que duerme hasta recibir una petición y va encolando las mismas cuando se encuentra en ejecución. Cada mensaje es procesado mediante un workflow, que en la implementación actual está basado en máquinas de estados.
- *Boundary Framework*: responsable de establecer la comunicación con otros sistemas mediante mensajes asíncronos independientes del protocolo (ej. XML, XML-RPC, SOAP, Java .RMI, .NET Remoting, etc). Está compuesto de un *Sender* responsable del envío y de un *Server* que recibe la información en un formato dado y lo convierte a la estructura general de mensajes internos del sistema.
- *Entity Framework*: almacena la información de los diversos workflows en ejecución en un espacio de memoria compartida independiente para cada subsistema y mantiene un espejo de los mismos en medio persistente. El sistema de memoria hace la función de pizarra compartida, siendo accesible en modo lectura y escritura por las diversas máquinas de estados del sistema.

En base a la arquitectura anteriormente expuesta, el comportamiento dinámico de la infraestructura sigue la siguiente secuencia de pasos, tal y como se muestra en la Figura 1 b). 1) Se recibe una petición, que llega a través de un protocolo determinado y es atendida por el *Server* correspondiente. La información es traducida al sistema de mensajes asíncrono interno. 2) El mensaje interno transformado es enviado al *Manager*. 3) El mensaje es almacenado en la cola de

peticiones (que está ordenada por un conjunto de valores que representan prioridades). 4) Atendiendo a esa cola de mensajes ordenada por prioridad, el mensaje siguiendo el patrón consumidor es procesado. 5) Si se trata de una petición, es necesario crear una máquina de estados que ejecute la lógica de la solicitud, y pasaría a formar parte del anillo de workflows denominado *SMList*. En el caso de que se tratara de un mensaje respuesta, el sistema salvaguarda quién solicitó la petición, de forma que la respuesta llega automáticamente a a la máquina de estados que estaba en ejecución en un estado *esperando la respuesta*. 6) La máquina de estados forma parte de la lista de workflows en ejecución y espera la llegada del *Token Ring* para poder comenzar. 7) Cuando le toca el turno de ejecución, comienza la lógica del workflow, pudiendo acceder a la estructura de datos y enviar mensajes a otros subsistemas. 8) Generalmente toda petición lleva asociada una respuesta, por lo que como resultado final de la ejecución del workflow, el subsistema invocado remitirá una respuesta final. Para enviar los resultados se utiliza el mismo sistema de mensajería, por lo que el mensaje resultado será trasladado al *Manager*, que en este caso hará de simple *proxy* hacia el módulo *Boundary*. 9) Llegado el mensaje al *Boundary Framework*, quedará almacenado en el *Sender* correspondiente y cuando se hayan enviado los anteriores el mensaje será emitido. Tras terminar esta operación la máquina de estados que solicitó el envío es informada para que finalice del todo su ejecución.

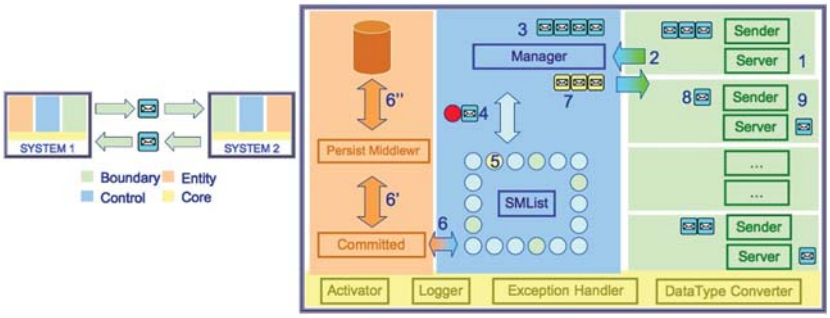

Fig. 1. a) Componentes del CERVANTES Framework; b) Pasos de ejecución

Posteriormente se desarrolló una herramienta para diseñar sistemas que fueran a ser desplegados en la infraestructura de Cervantes. Se trata de un plugin basado en el Eclipse Modeling Framework(EMF)[4] y en el Graphical Modeling Framework (GEF)[5], que permite a los ingenieros del software modelar los sistemas, creando las máquinas de estados y los mensajes y generando de forma automática el código para diversas plataformas como Java y C#.NET.

3 Conclusiones

En este trabajo se ha presentado CERVANTES, un marco para el diseño y desarrollo de sistemas distribuidos. Se compone de una infraestructura de ejecución y un entorno de desarrollo integrado que, siguiendo un enfoque basado por modelos, permite la generación automática de código en diversas plataformas. Este

marco se creó en el año 2001, cuando todavía los conceptos relacionados con *MDE* y *SOC* no estaban ampliamente desarrollados en la literatura y ha tenido desde su inicio una clara componente de aplicación práctica en la industria.

Como demostración se plantea partir de un sistema de gestión de flotas distribuido y añadir alguna nueva funcionalidad, haciendo uso del Studio para diseñar los nuevos mensajes y máquinas de estados y desplegarlos en la infraestructura de ejecución. También se propone recoger las métricas y analizarlas dentro del Studio como mecanismo para identificar problemas en el despliegue de los sistemas distribuidos.

Como trabajo futuro planteamos los siguientes pasos: 1) Integrar un soporte metodológico que cubra todo el ciclo de vida del desarrollo, permitiendo guardar una trazabilidad desde los requisitos hasta la ejecución. Para ello planteamos extender la metodología NDT[6] para el desarrollo de sistemas distribuidos. 2) Refactorizar el Studio para seguir un enfoque basado en modelos, valorando la opción de utilizar SOAML[7]. 3) Incorporar la generación automática de casos de prueba en etapas tempranas[8]. 4) Integrar aspectos de interacción persona ordenador en el diseño de sistemas distribuidos, valorando la opción de utilizar IFML[9]. 5) En base a las métricas de ejecución recogidas, dotar de inteligencia a la infraestructura para conseguir la reconfiguración automática de los sistemas.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto NDTQ Framework (TIC-5789) de la Junta de Andalucía y el proyecto MEGUS del Ministerio de Economía y Competividad (TIN2013-46928-C3-3-R).

References

1. Schmidt, D.C.: Model-driven engineering. (2006)
2. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: a research roadmap. International Journal of Cooperative Information Systems **17**(02) (2008) 223–255
3. Tsai, W.T.: Service-oriented system engineering: a new paradigm. In: Service-Oriented System Engineering, 2005. SOSE 2005. IEEE International Workshop, IEEE (2005) 3–6
4. EMF: Eclipse modeling framework project. Eclipse Foundation (2014) http://www.eclipse.org/modeling/emf/?project=emf, last accessed 04/2014.
5. GMF: Graphical modeling project. Eclipse Foundation (2014) http://eclipse.org/gmf-tooling/, last accessed 04/2014.
6. Escalona, M.J., Mejías, M., Torres, J., Reina, A.: The ndt development process. In: Web Engineering. Springer (2003) 463–467
7. OMG: Omg's service oriented architecture modeling language (soaml), http://www.omg.org/spec/soaml, accessed 2014-04-11
8. Gutiérrez, J.J.: Generación de pruebas del sistema a partir de la especificación funcional. PhD thesis, Universidad de Sevilla (2011)
9. OMG: Omg's interaction flow modeling language (ifml), http://www.ifml.org, accessed 2014-04-10