Revista
Española de

Innovación, Calidad e Ingeniería del Software

REICIS

Volumen 7, No. 3, diciembre, 2011

Web de la editorial: www.ati.es
Web de la revista: www.ati.es/reicis
E-mail: calidadsoft@ati.es
ISSN: 1885-4486

Publicado por la Asociación de Técnicos de Informática (ATI), Via Laietana, 46, 08003 Barcelona.

Secretaría de dirección: ATI Madrid, C/Padilla 66, 3º dcha., 28006 Madrid

Revista Española de Innovación, Calidad e Ingeniería del Software (REICIS)

Contenidos

REICIS

Editorial

REICIS

En esta ocasión, el editorial de REICIS se centra en la propia revista. Después de una exitosa presencia acercando a la comunidad en lengua española los trabajos de investigación e innovación desarrollados en el área de la calidad, la innovación y la ingeniería del software, creemos que ha llegado el momento de abordar una nueva fase de mejoras y expansión de esta publicación. Es cierto que su presencia en importantes índices en España e internacionales la sitúan como la referencia hispanohablante en el ámbito de la ingeniería del software. Pero es voluntad de la dirección de la revista y de la entidad editora ATI (la asociación más veterana y numerosa de profesionales de la informática en España) el intentar alcanzar niveles aún más altos de difusión y prestigio en la comunidad internacional.

Para ello se plantean dos acciones inmediatas destinadas a este propósito. La primera se centra en el ámbito del soporte administrativo y de publicación de REICIS: vamos a adoptar el sistema estándar OJS a lo largo de 2012 para mejorar la infraestructura de soporte y lograr un mejor servicio a todos los colectivos implicados: autores, lectores, revisores, responsables de índices internacionales, etc. La segunda innovación, tras una cuidadosa meditación junto con miembros del comité de redacción y el comité editorial, es la apertura de REICIS a contribuciones completamente escritas en inglés. En estos artículos, se conservará la duplicidad de título y resumen en inglés y español pero el resto del contenido se presentará en inglés. Estas contribuciones compartirán el espacio de REICIS junto con las que sigan el esquema tradicional de contenido en español. Ante las oportunidades de acercar a los autores no hispanohablantes a la comunidad de REICIS, no hemos dudado que esta oportunidad resultará beneficiosa en el futuro para todos los implicados. Se abre así, en 2012, un nuevo período de ilusión para todo el equipo de la revista que esperamos sea del mayor provecho para todos.

Luis Fernández Sanz
Director

Presentación

REICIS

Este número de REICIS publica, tras el proceso de revisión de nuestro comité editorial, dos contribuciones remitidas por los autores como parte del proceso regular de recepción de artículos de la revista.

El primero de los trabajos publicados corresponde al titulado "Patrones de adaptación para arquitecturas de software basadas en tecnologías del acuerdo" procedente de un equipo de autores de la Universidad Rey Juan Carlos. En este artículo, J. Santiago Pérez-Sotelo, Carlos E. Cuesta y Sascha Ossowski presentan su propuesta de aplicación de patrones para definir arquitecturas evolutivas para sistemas complejos mediante el uso de tecnologías del acuerdo. El trabajo muestra la adaptación de técnicas procedentes del mundo de los sistemas multi-agentes.

El segundo trabajo se titula "Análisis estático de código en el ciclo de desarrollo de software de seguridad crítica" presentado por Eduardo Lluna, del Instituto Tecnológico de Informática (ITI) vinculado a la Universidad Politécnica de Valencia. El trabajo investiga la aplicación de técnicas de análisis estático de código a los sistemas de control de seguridad crítica empotrados en productos y sistemas donde un fallo puede producir graves consecuencias que pueden incluir la pérdida de vidas humanas o daños ambientales irreparables. La propuesta se acompaña con datos de su aplicación a un proyecto real con código en C/C++.

Finalmente, en la columna de Actualidad Invitada, José Antonio Calvo-Manzano, director de la Cátedra de Mejora de Procesos de Software en el Espacio Iberoamericano, financiada por Everis, presenta las actividades de la misma y repasa las tendencias que podrán tener mayor influencia en esta área de procesos en el futuro.

Luis Fernández Sanz

Patrones de adaptación para arquitecturas de software basadas en tecnologías del acuerdo

J. Santiago Pérez-Sotelo[1], Carlos E. Cuesta[2] y Sascha Ossowski[1]
[1] Centro para las Tecnologías Inteligentes de la Información y sus Aplicaciones (CETINIA)
[2] Grupo de Investigación VorTIC3, E.T.S. de Ingeniería Informática
Universidad Rey Juan Carlos - Madrid, España
{josesantiago.perez, carlos.cuesta, sascha.ossowski}@urjc.es

Resumen

La creciente complejidad de los sistemas software requiere volver a concebir las estrategias de desarrollo y mantenimiento. Esto lleva a considerar a la auto-adaptación como una cuestión básica para la arquitectura. Simultáneamente, los sistemas multiagentes constituyen un enfoque genérico para resolver problemas complejos. Ciertas propuestas avanzadas usan organizaciones para proporcionar estructuras nuevas y más complejas. Sin embargo, todavía no proveen mecanismos para cambiar su composición o los tipos de elementos y, así, lograr una auténtica auto-adaptación. Este artículo propone una solución arquitectónica. El dinamismo necesario será proporcionado por un acuerdo emergente: una estructura arquitectónica evolutiva, basada en la combinación de controles y protocolos predefinidos. El objetivo es definir organizaciones adaptativas, con énfasis en los mecanismos (adaptativos) de coordinación. En este contexto, grupos de agentes orientados a servicios se reúnen, y pueden evolucionar, mediante patrones de adaptación hasta transformarse en una organización "estable". En este artículo se presenta un ejemplo específico para demostrar el interés del enfoque y para debatir su aplicabilidad.

Palabras clave: Auto-adaptación, arquitecturas adaptativas, sistemas multi-agente, tecnologías

Adaptation patterns for software architectures based on agreement technologies

Abstract

The growing complexity of software systems is causing a re-conception of their development and maintenance strategies. Self-adaptation has been recently recognized as a basic architectural concern. Concurrently, multi-agent systems have been developed as a generic approach to solve complex problems and advanced approaches use organisations to provide further complex structuring. However, they still do not provide mechanisms to change their composition patterns and element types, to achieve real self-adaptivity. This work proposes an architectural solution: the required dynamism will be supported by an emergent agreement: an evolving architectural structure, based on combining predefined controls and protocols. The objective is to provide adaptive organisations, and the emphasis is in the coordination mechanism (also adaptive). Service-oriented agents gather together in this context and the resulting aggregate can evolve applying adaptation patterns. Eventually it would reach a "stable" agreement. A case study is also provided in order to show the interest of this approach, and discuss its applicability.

Keywords: self-adaptation, adaptive architecture, multi-agent Systems, agreement technologies, dynamism.

Pérez-Sotelo J.S., Cuesta C.E. y Ossowski, S., "Patrones de adaptación para arquitecturas de software basadas en tecnologías del acuerdo", REICIS, vol. 7, no.3, 2011, pp. 6-24. Recibido: 22-11-2011; revisado:1-12-2011; aceptado: 1-12-2011.

1 Introducción

Los sistemas actuales son cada vez más complejos, y este hecho está llevando a los diseñadores de sistemas a replantear la estrategia para su gestión. Las dificultades van más allá de la administración de sistemas individuales; las tareas rutinarias que previamente eran gestionadas por humanos son ahora realizadas por los propios sistemas, incluso acciones relacionadas con sus propias funcionalidades. Los sistemas complejos pueden observarse a sí mismos y adaptar sus estructuras y comportamientos según sea necesario. Esto incluye no sólo al comportamiento funcional, sino también a las propiedades no-funcionales. Por lo tanto, si se considera la auto-adaptación como una cuestión básica a nivel de arquitectura, hay muchas características a tener en cuenta, según Hübner et al. [1]. Algunos de sus beneficios potenciales incluyen, por ejemplo, la escalabilidad en la construcción de sistemas-de-sistemas, o la abstracción para describir cambios dinámicamente, entre otros [2].

Al mismo tiempo, en el campo de la Inteligencia Artificial (IA), los Sistemas Multi-Agente (SMA) se han desarrollado como un enfoque genérico para resolver problemas complejos. Algunas de las propuestas avanzadas utilizan el concepto de *organizaciones* para proveer estructuras adicionales. Sin embargo, éstas aún pueden tener ciertas limitaciones para lograr una verdadera auto-adaptación, es decir: no disponer sólo de la capacidad de afectar su *configuración* sino también su propia *composición* y los tipos de los elementos que las forman.

La solución propuesta se basa en un *ecosistema de servicios* con Tecnologías del Acuerdo (AT – Agreement Technologies) [3][4][5], en el que se pretende proporcionar el dinamismo necesario. En este enfoque, los servicios se ofrecen a través de las *organizaciones* de agentes, independientemente de cómo éstos los exporten a su vez. El objetivo es proveer *organizaciones adaptativas*, haciendo énfasis en que los mecanismos de coordinación también habrán de ser adaptativos. El propósito es definir una "tecnología inteligente" genérica que pueda resolver problemas complejos, lista para ser utilizada en entornos *abiertos*, y capaz de *adaptarse* a futuras evoluciones.

Los agentes (orientados a servicios, en este caso) se reúnen en un contexto o entorno predefinido de *controles* y *protocolos*; el resultado de su unión puede evolucionar mediante la aplicación reglas definidas por ciertos *patrones de adaptación*, llegando eventualmente a un *acuerdo* "estable".

Este artículo está estructurado como sigue: en la sección 2 se discute el núcleo de la propuesta, del que emerge una arquitectura adaptativa dentro de un contexto de SMA. También se presentan referencias a trabajos relacionados. La siguiente sección describe los conceptos que definen la estructura de estas tecnologías, la estructura del *acuerdo* en sí, que se define como transversal a cinco capas conceptuales y cuenta con la base de las llamadas Tecnologías del Acuerdo. La estructura del *acuerdo* también es incorporada en la definición de la plataforma THOMAS [6], que implementa sus conceptos y características, y que sirve como soporte a los desarrollos y experimentos adicionales. Asimismo se expone su evolución, ya que con sus características actuales la plataforma presenta ciertas limitaciones; por ejemplo, precisamente, en la adaptación a los cambios en el entorno, en lo relativo a la estructura de las organizaciones, en la aparición de nuevas normas y relaciones sociales entre agentes, etc. A continuación, se discuten conceptos y mecanismos que deben ser colocados en una capa por encima de THOMAS, capaces de generar las estructuras que definirán organizaciones emergentes, estructuras auto-organizadas y, en última instancia, *arquitecturas adaptativas*. En la sección siguiente se analizan los ya mencionados *patrones de adaptación*, incluyéndose una breve descripción de un patrón objeto de estudio y un ejemplo de protocolo especificado en *cálculo-π*. Finalmente se presentan conclusiones y futuras líneas de trabajo.

2 Hacia una arquitectura adaptativa

Actualmente los SMA son cada vez más populares en IA para resolver problemas complejos con eficacia. Se han propuesto diversas estrategias de desarrollo para que sean flexibles y puedan coordinarse a efectos de adaptarse a los cambios en el entorno. Sin embargo, también se acepta que los SMA no tienen el éxito esperado en la industria del software [7] [8], y probablemente se deba a una cultura diferente de desarrollo. El enfoque de esta propuesta es salvar esta brecha utilizando *conceptos orientados a servicios*, los que sí cuentan con una mayor popularidad. Si, además, este enfoque demuestra la capacidad auto-adaptativa deseada, se podrá cumplir con la promesa original de los SMA, que el sistema sea capaz de adaptarse por sí mismo a las condiciones cambiantes del problema por resolver.

La arquitectura que dé soporte al modelo se define como *SMA abierto* y también como *orientada a servicios* y *centrada en organizaciones*. En las secciones siguientes se describirá brevemente el diseño básico de esta arquitectura, así como también su

evolución para que sea capaz de describir las estructuras emergentes. El objetivo es que este diseño describa una arquitectura verdaderamente auto-adaptativa.

2.1 Una arquitectura basada en agentes y orientada a servicios

Debido a que se propone un entorno verdaderamente flexible y dinámico, se requiere el uso de capacidades semánticas y de alta tecnología. Por lo tanto, se considera el uso de agentes en un contexto más amplio, con el agregado de una capa superior de servicios para proveer una función de *interoperabilidad*. Es fácil concebir un servicio como una manera de presentar las capacidades operacionales de un agente, o mejor aún, de una colección de agentes como una *organización*, la que proveerá el *servicio*. El utilizar agentes permite, entre otras cuestiones, el tratamiento explícito de la semántica, una coordinación estructurada, su capacidad de aprendizaje, usar una metodología para el desarrollo de servicios y estructurarlos en organizaciones, etc.

Este trabajo tiene tres metas principales: la definición de una plataforma general para identificar la arquitectura subyacente *basada en agentes, orientada a servicios* y *centrada en organizaciones*, que conduzca a la plataforma fundamental para las Tecnologías del Acuerdo; segundo, la introducción de estructuras adicionales, para hacerla *adaptativa*; y tercero, la identificación de las estructuras genéricas de adaptación para las organizaciones, en la forma de una construcción del *acuerdo*, y su evolución.

La noción central es el *servicio*, el componente básico de la arquitectura es el *agente*, y la estructura que unifica los conceptos es la *organización*, concebida como una composición jerárquica y recursiva de agentes. Implícita en la definición de SMA se encuentra la necesidad de *registrar* a los agentes en el sistema, para separar aquellos que pertenecen a la arquitectura de los que no. Un enfoque similar será utilizado con los servicios. Para permitir su acceso externo, primero deberán ser explícitamente registrados y agrupados como parte de una entidad mayor – incluso un servicio "compuesto". Estos servicios podrán ser descubiertos, posteriormente, por otras entidades, así como dentro del registro distribuido del sistema.

2.2 Trabajos relacionados: el rol de la coordinación

Posiblemente sea mejor considerar el concepto de *coordinación* como una cuestión previa a la *adaptabilidad*. Cuando se utiliza un SMA como solución, el problema de la coordinación siempre está presente, y consecuentemente, la adaptabilidad también se ve comprometida. Un modelo de coordinación debería abarcar los temas de creación y destrucción de los agentes, su comunicación y su distribución espacial, así como la

sincronización y la distribución de sus acciones en el tiempo [9]. En un sistema de coordinación los componentes son: *entidades* (llamados también *coordinables*, cuyas interacciones se rigen por el modelo); *medios* (las abstracciones que rigen las interacciones); y *leyes* (que definen el comportamiento de los medios de coordinación en respuesta a la interacción) [9].

De acuerdo con [10], pueden tenerse en cuenta dos tipos de modelos de coordinación: los modelos *control-driven* (dirigidos por control) y los modelos *data-driven* (dirigidos por los datos). Los primeros se centran en el acto de la comunicación, mientras que los segundos lo hacen en la información intercambiada en ella. De hecho, en lugar de considerárselos modelos opuestos y diferentes, se podrían considerar como dos maneras diferentes de *observar* la coordinación.

Cuando se trata el caso específico de un meta-modelo de *espacio de tuplas* [9], tal vez el caso mejor conocido de coordinación *data-driven* (o generativa), las entidades basan sus interacciones (cooperación, competición, etc.) en las propias tuplas de datos. La coordinación, obviamente, tiene lugar mediante la producción, el consumo, etc. de las tuplas. En resumen, se crea un entorno de *comunicación generativa*.

Avanzando en la evolución de la coordinación, los *sistemas auto-organizados* tienen un creciente nivel de organización interna entre sus componentes en términos de interacciones, de su estructura, etc. [11][9]

Una reciente definición de *auto-organización de la coordinación* tomada de [12] plantea una gestión de interacciones del sistema con propiedades de auto-organización, donde las interacciones son locales y los efectos de coordinación global deseados aparecen por *emergencia*. Constructivamente, la auto-organización de la coordinación se consigue mediante un medio de coordinación distribuido sobre el entorno topológico, estableciendo reglas de coordinación *probabilísticas* y *dependientes del tiempo*.

Un enfoque significativo, y cada vez más popular, en el contexto de SMA ha sido considerar a los agentes dentro de *organizaciones*, y no de manera aislada. Sin embargo, esto también puede ser considerado como un enfoque de coordinación: en lugar de tener una coordinación normal de todo el sistema, las organizaciones permiten tener un *ámbito de coordinación*. En este sentido, hay una relación intrínseca entre la coordinación auto-organizada, que se mencionó anteriormente, y el enfoque de este artículo, basado en la definición de una arquitectura adaptativa en base a organizaciones *emergentes* - por lo que se obtendrá una *coordinación auto-organizada* y *con ámbito*.

3 Tecnologías del acuerdo: la plataforma y su evolución

La noción central en esta tecnología es el *acuerdo* entre entidades computacionales: *organizaciones*, en el nivel superior, y también *agentes*, en el nivel inferior. Se concibe como una construcción arquitectónica, que debe ser capaz de evolucionar para permitir la definición de un *acuerdo emergente* entre las entidades.

3.1 Tecnologías del acuerdo

En este trabajo se utiliza el conjunto de tecnologías y enfoques conocidos globalmente como *Tecnologías del Acuerdo* [1]. Los aspectos considerados para proponer una coordinación basada en acuerdos se estructuran en forma de "torre" y por niveles, de modo que cada uno proporciona funcionalidad al nivel inmediatamente superior (ver Figura 1). De esta manera el acuerdo, por definición, está *estructurado en capas*. Intuitivamente puede verse que cuando se alcanza un acuerdo, los elementos situados en los niveles inferiores deben respetarlo en su propio nivel. Los agentes que forman una organización deben cumplir con los términos del acuerdo.

La "estructura en torre" define un conjunto de capas que expresan la esencia conceptual de un acuerdo. Las capas son:

Semántica. Es la capa inferior, ya que las cuestiones semánticas influyen sobre todas las otras. La alineación semántica de ontologías [13] debe tenerse en cuenta tanto para evitar las no correspondencias, como para tener un entendimiento común.

Normas. Esta capa tiene que ver con la definición de normas/reglas que determinan las limitaciones que los acuerdos, y el proceso para llegar a ellos, tienen que satisfacer. Pueden suponer roles estructurales que afectan (o controlan) el comportamiento de los agentes, entrelazados con el dominio semántico.

Organizaciones. Implican una súper-estructura que restringe la manera en que se consiguen los acuerdos mediante la fijación de la estructura social de los agentes, las capacidades de sus roles y las relaciones entre ellos. En definitiva, la arquitectura de un sistema multi-agente, estructurada en configuraciones semi-independientes.

Figura 1: estructura original de las Tecnologías del Acuerdo [1]

Argumentación y Negociación. Pueden ser vistos como los protocolos que definen la estructura de un acuerdo. Tiene que haber acuerdos a alto nivel (entre organizaciones) y también de bajo nivel (un pacto o acuerdo individual).

Confianza. Es la capa superior en la estructura. Se necesitan mecanismos de confianza que resuman la historia de los acuerdos y las ejecuciones subsecuentes de los mismos con el fin de construir relaciones a largo plazo [14].

Para mayores detalles acerca de estos conceptos, véase [4][5].

El enfoque propuesto provee los elementos requeridos para construir una arquitectura adaptativa, de modo que para definir un acuerdo emergente sólo se requiere la identificación de los patrones estructurales y el conjunto de protocolos inter-niveles. Pueden plantearse mayores refinamientos, por ejemplo considerar meta-elementos, o definir agentes específicos para realizar tareas de apoyo para el propio acuerdo (tales como sensores, observadores, controladores, planificadores, etc.), entre otros.

3.2 La plataforma

En esta subsección se presenta la arquitectura base para las tecnologías previamente discutidas, que fueron concebidas para ser soportadas por un SMA abierto.

Las actuales investigaciones en la plataforma se orientan a lograr una mayor capacidad y funcionalidad, aprovechando las características de los SMA. Es más, y desde este punto de vista, los servicios son específicamente utilizados para lograr una mayor interoperabilidad. La idea principal es exportar el *sistema de agentes* como un *sistema de servicios*. El ecosistema de servicios resultante será soportado tanto tecnológica como metodológicamente [15].

Estos conceptos se elaboran sobre la base de un sistema ya existente, la plataforma THOMAS [6]. Su diseño se resume en el gráfico siguiente (véase Figura 2).

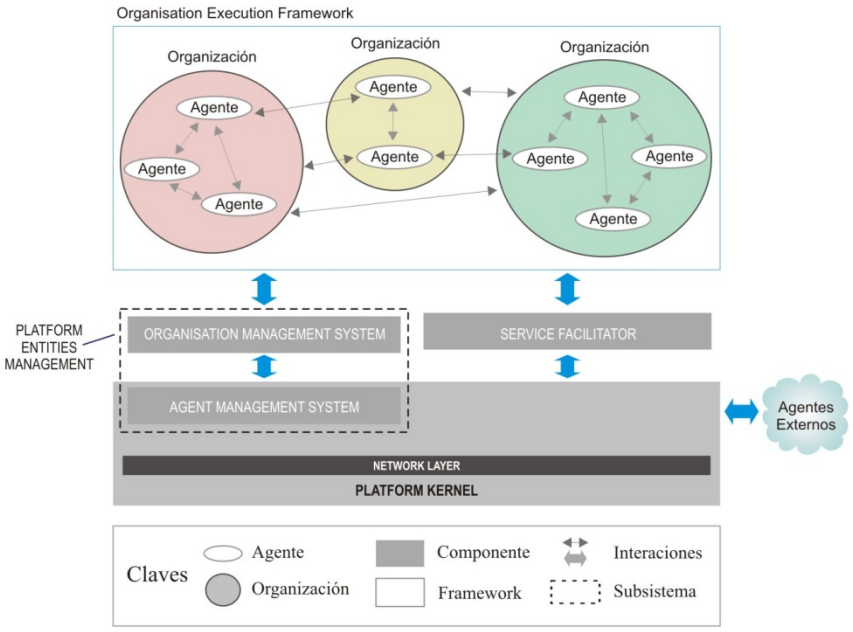

Figura 2: arquitectura técnica THOMAS (inspirado en [6])

La plataforma, incluido su *middleware*, está estructurada en tres niveles (aunque no son estrictamente capas) ortogonalmente soportados por cuatro componentes específicos que se incluyen como formando parte de tres diferentes subsistemas. A su vez, el subsistema *Platform Entities Management* está formado por capas, que se utilizan para proporcionar las capacidades a los diferentes niveles de la plataforma. Los tres niveles son:

- *Platform Kernel (PK)*. Es el *kernel* del middleware, incluye la capa de red y el componente Agent Management System (AMS). Provee las capacidades de una arquitectura compatible con FIPA [16]. A este nivel de descripción, la plataforma es un SMA abierto.
- *Service & Organisation Management. Es el nivel conceptual compuesto por los* componentes Organization Management System (OMS) y Service Facilitator (SF). No es estrictamente un subsistema, aunque provee las características y abstracciones relevantes para el Execution Framework.
- *Organisation Execution Framework.* Es el "espacio" donde las entidades computacionales se ejecutan y realizan sus tareas. Los agentes, sus organizaciones y los servicios que ofrecen están conceptualmente ubicados allí. Cada aplicación específica se concibe, diseña y ejecuta en este nivel de abstracción.

Los tres componentes principales mencionados de la plataforma son:

- *AMS*, que proporciona todas las capacidades y funciones necesarias para la gestión de los agentes;
- *OMS*, que hace lo propio con las organizaciones a la vez que mantiene unido al sistema en su conjunto;
- *SF*, que proporciona las capacidades y funciones necesarias para permitir que cierta selección de operaciones de una organización se comporte como un servicio unificado.

Más detalles acerca de la plataforma THOMAS pueden encontrarse en [6].

3.3 Evolución de la plataforma

Los progresos actuales sobre la plataforma se dan a nivel de los servicios con su adaptación a la especificación OSGi [17]. La idea principal es modularizar las aplicaciones en entidades más pequeñas denominadas *bundles*. Estas entidades pueden ser instaladas, actualizadas o eliminadas dinámicamente, proporcionando la posibilidad de cambiar el comportamiento del sistema sin tener que interrumpir su operación.

El *Service Tracker* se distingue entre los servicios que provee OSGi, especialmente para el enfoque propuesto: permite rastrear otros servicios registrados en la plataforma y también verificar que los que se han proveídos están o no disponibles. En un sistema basado en *bundles*, tanto productores como consumidores de servicios pueden aparecer y desaparecer dinámicamente. Para realizar el paso de mensajes, síncronos o asíncronos, entre dos entidades que pertenecen a distintos *bundles*, la especificación proporciona una herramienta: el patrón *Whiteboard*. Éste utiliza el registro de servicios de OSGi para mantener un registro de los *listeners* del sistema. Uno de los beneficios de esta herramienta es que el control del ciclo de vida de productores y consumidores de eventos se delega en la plataforma, que notifica a consumidores la desaparición de productores y viceversa.

Las actuales investigaciones, que se incluyen como parte del proyecto OVAMAH [18], se desarrollan ampliando los objetivos de la plataforma THOMAS. Además de proveer la tecnología necesaria para el desarrollo de organizaciones virtuales en entornos abiertos, permitirá dar *respuestas dinámicas* por medio de la adaptación y/o evolución de las propias organizaciones. Por ejemplo, agentes integrantes de una unidad organizacional podrán crear (o eliminar) otra unidad, afectando a los grupos del sistema; con respecto a las normas, podrán definir en qué momento es necesario agregarlas o eliminarlas; el tipo de relación social entre roles podrá cambiar en tiempo de ejecución,

tanto las condiciones para activarlas/desactivarlas como la cardinalidad entre roles; la topología del sistema (dada por las relaciones) podrá cambiarse (agregar/eliminar relaciones) también en tiempo de ejecución y luego verificarse que este cambio es consistente con los objetivos y el tipo organizacional; los servicios podrán ser asociados a nuevos roles que surjan en el sistema; etc.

4 Definición del acuerdo emergente

Cuando se utiliza un SMA abierto como solución de un problema complejo, a menudo se requiere cierta adaptación. Al mismo tiempo, la propia estructura debe ser flexible para lograr coordinación dentro del sistema.

El *acuerdo* entre entidades computacionales puede verse como un concepto adecuado para abordar la necesidad de una estructura adaptativa. El objetivo es descubrir una estructura apropiada, de modo que *emerja* como un acuerdo global. A continuación se definen conceptos y elementos que, construidos sobre las estructuras básicas descriptas en los apartados anteriores, harán posible definir una organización *emergente* sobre la base del *acuerdo*. Se pueden encontrar trabajos recientes relativos a las organizaciones dinámicas en [1][19][20].

4.1 Canalizando la emergencia: controles y protocolos

Cualesquiera que sean los objetivos, cualquier grupo de individuos puede ser ordenado según ciertas estructuras (por ejemplo una sociedad, una arquitectura, una jerarquía, etc.). Se puede provocar explícitamente la formación de estas estructuras mediante el uso de dos tipos diferentes de mecanismos, que tienen su fundamento en la limitación del rango de acciones válidas: *controles* y *protocolos*.

En cuanto a los primeros, los *controles*, pueden tanto *imponer* como *prohibir* interacciones específicas (a menudo, en la forma de conexiones arquitectónicas). Las estructuras auto-adaptativas, siendo típicamente centralizadas [11], muestran muchos ejemplos clásicos de este tipo: la mayoría de ellos manifiestan lazos explícitos de control, inspirados en los *reguladores* de la teoría de control clásica.

Por otro lado, los *protocolos*, que pueden *permitir* o *canalizar* comportamientos, se basan en el consenso y en acuerdos. Pueden ser descritos, genéricamente, como una manera de controlar las estructuras descentralizadas (o incluso distribuidas) [21]. Es decir, con los protocolos cada agente sabe cómo interactuar con el resto; es necesario

ponerse de acuerdo con ellos para poder comunicarse, y al mismo tiempo *regulan* el desarrollo de la estructura de interacción.

Estos dos mecanismos definen un amplio espectro de regulaciones. Las organizaciones de agentes y sus arquitecturas son simultáneamente regulados por controles unitarios, atómicos (como las *normas*, límites, *bloqueos*, lazos de control o *restricciones*), y protocolos múltiples, conectivos (como los *concentradores, puentes, canales* o *espacios*).

En arquitectura de software son conocidos muchos patrones y soluciones basados en controles implícitos y restricciones normativas. Sin embargo, el enfoque propuesto intenta establecer una solución basada en el *consenso*: también se trata de un enfoque a nivel arquitectónico, ya que la descripción de las interacciones pertenece a este nivel.

Es importante señalar que el propósito de estos mecanismos es "descubrir" una estructura adecuada de controles y protocolos para que pueda emerger una estructura global (es decir, la definición de diversas formas de arquitectura). Estos elementos harán posible definir las principales estructuras internas a fin de obtener organizaciones basadas en acuerdos.

Una vez que una estructura primaria pueda ser definida, un grupo elemental surge como una organización preliminar. Ésta será referenciada como una *iniciativa,* y su estructura será explicada en la siguiente subsección.

4.2 Definición de un acuerdo emergente: la *Iniciativa*

Como ya se ha señalado, un conjunto de controles y protocolos puede ser utilizado para generar dinámicamente una organización preliminar dentro de un grupo de individuos (agentes, en este artículo, aunque también podrían ser componentes genéricos).

En esta propuesta se define y utiliza un conjunto de controles y protocolos para generar cierta estructura (por tanto, varios de ellos son considerados como *controles generativos* y *protocolos generativos*). Esta estructura da lugar a una organización que crece con la dinámica del entorno. La organización emergente es lo que llamaremos una *iniciativa*: todavía no está plenamente establecida, pero sigue evolucionando.

La *iniciativa* puede seguir creciendo y mutando debido a su naturaleza *adaptativa*, pero cuando se tiene algún tipo de estructura "estable", ya es posible llamarla *organización*. Esta estructura se logra cuando todos los participantes pueden llegar al *acuerdo* necesario con el fin de solucionar el problema o conseguir el objetivo principal

que provocó su unión. La organización resultante es conceptualmente similar a otras organizaciones en varios enfoques de SMA, incluso en THOMAS [6].

El párrafo anterior implica tres conceptos importantes [22]:

- *Una iniciativa*. Es un grupo preliminar de individuos (agentes) que se reúnen en una determinada estructura, generada por un conjunto de controles y protocolos, así como también por ciertos patrones asociativos;
- *Una organización*. Se trata de un grupo establecido, dinámicamente originado desde una iniciativa (aunque también hay organizaciones "estáticas", una vez que se crean, ambos tipos son funcionalmente equivalentes);
- *Un acuerdo*. Es el acto por el cual una iniciativa se convierte en una organización "estable". De hecho, esto puede verse como el consenso que se alcanza entre los individuos dentro de la "semilla" inicial del grupo.

El proceso puede ser visto como si el sistema se moviera a un nuevo estado, en el que la estructura del "pasado" es suplantada por una "nueva estructura emergente". Obviamente esta nueva estructura admite nuevos elementos debido al entorno dinámico, pero ahora uno de sus objetivos es reforzar su naturaleza y tender a la persistencia. Como las estructuras pueden ser cada vez más complejas, queda claro que para ciertas clases de problemas es necesario que los individuos se agrupen en organizaciones, y después, como ya se ha dicho, en organizaciones "estables" basadas en acuerdos.

Por ejemplo, en una situación de emergencia, varios coches policiales pueden llegar a un lugar determinado, pero ninguno de ellos es, a priori, el líder del grupo. Siguiendo un protocolo interno, pueden elegir a uno como tal, y este acuerdo genera una organización *preliminar*: incluso la jerarquía es un protocolo. Esto es lo que podría denominarse como un *protocolo generativo*; cuando los individuos siguen este tipo de protocolo, se definen patrones estructurales implícitos.

Una *iniciativa* puede generarse a partir de estos patrones, a los que llamaremos *patrones de adaptación* – téngase en cuenta que el término "patrones" es utilizado en un sentido arquitectónico. Son pre-diseñados a partir de los servicios necesarios por una iniciativa, y concebidos para el refinamiento semántico correspondiente. Algunos ya han sido identificados (véase Tabla 1): *Façade*, *Mediator*, *Surveyor*, entre otros.

4.3 Ciclo de vida de las estructuras auto-organizadas

Como ya se ha dicho, un grupo de individuos puede organizarse, en base a metas concretas, en ciertas estructuras utilizando controles y protocolos. Estos mecanismos

hacen posible definir las estructuras internas relevantes para obtener organizaciones basadas en acuerdos. Una vez que la estructura primaria está definida, un grupo "elemental" emerge como entidad preliminar: la *iniciativa*. Ésta cambiará con la dinámica del entorno hasta ser una organización "estable".

La Figura 3 resume brevemente el *ciclo de vida* de las estructuras auto-organizadas propuestas [23]. El ciclo puede comenzar con un simple *agente*, que está en condiciones de generar ciertas interacciones, y además tiene el potencial de exportar algunos de sus *servicios*. Inicialmente no pertenece a *organización* alguna cuando llega al sistema, sin embargo se ajusta a cierto número predefinido de *controles* y *protocolos*, que le guiarán en las interacciones y le permitirán mantener conversaciones estructuradas con otros agentes, formando grupos informales con ellos.

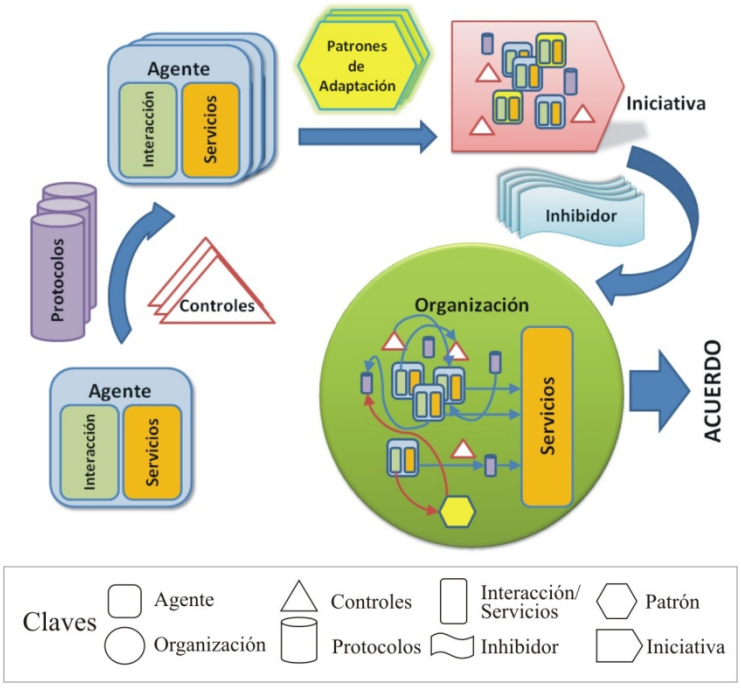

Figura 3: Ciclo de vida de una estructura auto-organizada [23]

Al ocurrir un cambio externo, el sistema debe reaccionar con un comportamiento *adaptativo*, y ésta es la funcionalidad que debe *disparar* la formación de las estructuras auto-organizadas (organizaciones). El sistema tiene a disposición cierto número de *patrones de adaptación* que pretenden, en conjunto, lograr la reacción deseada. Estos patrones son definiciones parciales de elementos y relaciones, y que incluyen la información suficiente para que un agente *aprenda* cómo realizar (o, siguiendo el protocolo, realice) cierto comportamiento. En resumen, guiados por un patrón de adaptación, ciertos agentes del grupo adquieren funciones específicas y comienzan a

forma una estructura: es lo que hemos llamado una *iniciativa*. Por supuesto, estas organizaciones pueden continuar evolucionando y participar, además, en otros acuerdos mayores [23].

5 Patrones de adaptación

Como ya se ha dicho, una *iniciativa* puede ser generada a partir de patrones, los llamados *patrones de adaptación*. Éstos son también patrones arquitectónicos: algunos de ellos ya han sido identificados, y reciben nombres como *Façade* o *Mediator*, entre otros [24].

El sistema está concebido como una *arquitectura orientada a servicios*, por lo tanto, metodológicamente, las primeras organizaciones "estables" deber ser concebidas como proveedoras de ciertos servicios de alto nivel. Estos servicios también serán usados como punto de partida para la definición funcional de las organizaciones evolutivas – incluso en su etapa inicial tratan de proporcionar esa funcionalidad.

Nombre	Descripción y funcionamiento.
Façade	Algún agente ha de representar a la propia organización para poder interactuar fácilmente con una que aún carece de estructura definida. Este agente será la *fachada* que redirige cualquier comunicación recibida; no es necesario que sea también un supervisor.
Mediator	Durante el proceso de emergencia, la organización aún no está establecida y los servicios de datos posiblemente no estén funcionando. Algún agente debe actuar como mediador haciendo posible el acceso a fuentes de datos, al menos indirectamente, y también realizar las traducciones necesarias, incluyéndose distintas clases de traducción *semántica*.
Surveyor	Al menos un agente debe controlar el crecimiento de la propia iniciativa durante el proceso de emergencia. Asimismo puede decidir el momento de introducir nuevos elementos y también cuándo se estabiliza el grupo. El *surveyor* tiene acceso a la biblioteca de patrones – decide cuándo cierto patrón es el adecuado, y debe ser activado.

Tabla 1: Algunos patrones de adaptación: patrones de diseño a nivel arquitectónico

Como ejemplo sencillo se ha seleccionado otro patrón de adaptación: *Gathering*. A continuación, y de acuerdo con [25], se proporciona un extracto de la plantilla de este patrón (véase Figura 4). La plantilla es similar en estilo y espíritu a los usados por [26], pero algunos campos han sido modificados para denotar la dinámica de los sistemas adaptativos. Téngase en cuenta que varios campos, y sus descripciones, se han omitido o simplificado debido a restricciones de espacio.

El patrón describe la situación donde varios agentes se encuentran cerca de un punto de encuentro (*venue*). Cuando ocurre un evento en este punto, los agentes se encuentran capacitados para interactuar directamente entre sí: en resumen, este evento inicial puede disparar la creación de una *iniciativa*.

Como ocurre en otros patrones, la plantilla debe complementarse con algún modelo de interacción – en los patrones de adaptación los *protocolos* son de particular importancia: esta es la razón por la que se usará una especificación en *cálculo-*π [27]. Se utiliza este cálculo debido a su gran expresividad; pero no es necesario utilizarlo para la codificación de los patrones: de hecho, se podrían utilizar otros mecanismos (o incluso formalismos) a estos efectos.

Esta especificación esencialmente utiliza el cálculo-π poliádico estándar, con algunas y convenientes alteraciones. Se utiliza la notación CSP en lugar de la original, por lo que también se reserva el símbolo asterisco (*) para representar la operación de *replicación*. También se asume una construcción (;) para una composición secuencial, y se incluye además el condicional (if A then B). Ambos se codifican convencionalmente de una manera estándar [27]. Se agregan también dos funciones simples, fácilmente implementables, por conveniencia algorítmica: **near** (detecta proximidad) y **count**. En la Figura 5 puede leerse un extracto de la especificación.

- Nombre del Patrón. ***Gathering*** **(Reunión)**
- Clasificación. Monitorización, Creacional.
- Propósito. Monitorizar el "espacio" que contiene elementos computacionales (agentes), canalizar los datos observados, facilitar la interacción y, en última instancia, proporcionar coordinación.
- Contexto. [*Describe el contexto de aplicación del patrón.*] *Gathering* puede usarse cuando aún no se ha formado una *iniciativa*, es decir, cuando los agentes pueblan el "espacio" pero todavía no se han definido los servicios de alto nivel, proporcionando una organización.
- Motivación. El sistema necesita coordinarse como un ecosistema de servicios, en el que los agentes puedan entrar o salir del entorno dinámicamente. Esto justifica un comportamiento que no podría

Figura 4: Plantilla del patrón de adaptación Gathering (extracto)

```
AgentX (sys_where) ::= …
        + (ν hi) sys_where!⟨hi⟩. sys_where?(where). τ. where?(event).
            (ν me, ret) ∗[ event!⟨me, ret⟩. ret?(you). τ. (me!⟨data⟩ |
you!⟨info⟩) ];
                AgentX⟨sys_where⟩
        + …

Venue (sys_place) ::= …
        + (ν here, r) sys_place!⟨here, r⟩. τ.
            ∗[ r?(hi). if count(r) ≥ 2 then (ν event) here!⟨event⟩.
                ∗[ event?(ag_x, ragx). (
                  ∗[ event?(ag_y, ragy). ragx!⟨ag_y⟩ ] | event!⟨ag_x, ragx⟩
) ] ];
                   Venue ⟨sys_place⟩
        + …

SubSystem (sys_place, sys_where) ::=
```

Figura 5: Extracto de especificación en cálculo-π para agentes cerca del punto de encuentro (venue)

El proceso *AgentX* describe la manera en que un agente estándar debe comportarse en el patrón. En primer lugar, es consciente de su existencia (*hi*) y pregunta al sistema dónde se encuentra (*sys_where*). Si está cerca de un punto de encuentro (*venue*) se le envía su nombre (*where*). Cuando algo (*event*) sucede allí, el agente envía su propio nombre (*me*) y un canal de respuesta (*ret*). Por este canal, recibe el nombre de otro agente (*you*). Por lo tanto, los dos agentes pueden ahora interactuar directamente, y no volver a usar el *venue*. Ésta última parte es un sub-proceso repetitivo (*[*A*]), que se repite indefinida y concurrentemente – lo que significa que el agente puede comunicarse con muchos otros, dos a dos.

El proceso *Venue* describe la influencia del "punto de encuentro" en los agentes. Primero informa al sistema de su existencia (*here*), y provee un canal de respuesta (*r*), en el que los agentes eventualmente podrán registrarse. Una vez que haya dos o más agentes, el evento se dispara y se retransmite dentro de un proceso repetitivo. El canal *event* es utilizado para recibir el nombre (*ag_x*) y devolver el canal a todos los agentes cercanos, encontrar una pareja potencial (*ag_y*) y comunicarse con ella, dentro de un proceso paralelo. Por supuesto hay otras maneras más sofisticadas para definir un protocolo de difusión (*broadcast*): éste es solo un ejemplo. El proceso *SubSystem* obviamente actúa como conector, esto es, define el comportamiento de la porción de *middleware* que aglutina al conjunto.

Como comentario final, puede verse que cada proceso AgentX es un *shifter* [23]: esto es, un "agente que cambia" – está cambiando su forma, es decir, su capacidad para interactuar con otros. En el mismo contexto, el *Venue* se define como un *changent*: esto es, un "agente de cambio" – que es el agente que *induce el cambio* sobre el resto de los elementos del patrón.

6 Conclusiones y trabajos futuros

Este trabajo ha explorado conceptos estructurales como base de un enfoque arquitectónico para proporcionar auto-adaptación a sistemas software. El concepto de *iniciativa* puede ser considerado como un punto de partida para proveer mecanismos que permitan el cambio de los patrones de composición y de los tipos de elementos dentro de tales sistemas.

El dinamismo necesario puede ser soportado por un acuerdo emergente – una estructura arquitectónica evolutiva, basada en la combinación predefinida de controles y protocolos: éstos son gestionados en el contexto del marco orientado a servicios, basado en agentes y centrado en organizaciones definidos en las *Tecnologías del Acuerdo*, provisto por la implementación en la plataforma THOMAS, el proyecto OVAMAH y servicios compatibles con el estándar OSGi.

La idea clave es crear un contexto arquitectónico en el que los agentes son *coordinados* y *reorganizados* por su inclusión en estructuras preliminares –los *patrones de adaptación*– y luego compuestos en organizaciones "estables".

Los patrones propuestos pueden ser entendidos como las estructuras necesarias para lograr una auto-adaptación real. Los protocolos que definen los patrones se pueden representar formalmente, como se puede ver en la especificación en cálculo-π incluida. De hecho, aún cuando el enfoque propuesto parece prometedor, estos son únicamente los primeros pasos: el potencial de crecimiento es significativo.

En trabajos futuros se desarrollarán e implementarán variantes de este enfoque en el marco de una nueva plataforma de agentes basada en THOMAS (denominada AT-MAS) a fin de lograr su refinamiento. Los conceptos todavía están evolucionando, pero aún en esta fase los fragmentos del enfoque ya han demostrado su utilidad, y su poder expresivo. Los resultados sugieren que la meta final, esto es, definir una auténtica *arquitectura adaptativa*, es viable, dado que la infraestructura que se ha creado puede crecer simplemente añadiendo nuevos patrones de adaptación.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia e Innovación de España, para los proyectos "Tecnologías del Acuerdo" (CONSOLIDER CSD2007-0022, INGENIO 2010), "OVAMAH" (TIN2009-13839-C03-02) y "MULTIPLE" (TIN2009-13838); y el "European Union RTD Framework Programme", a través de la Acción COST Agreement Technologies (COST Action IC0801).

Referencias

[1] Hübner, J.F., Boissier, O., Kitio, R., y Ricci, A., "Instrumenting multi-agent organisations with organisational artifacts and agents: Giving the organisational power back to the agents", *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 20, nº 3, pp. 369-400, 2010.

[2] Kramer, J. y Magee, J., "Self-Managed Systems: an Architectural Challenge", *Future of Software Engineering (FOSE@ ICSE'2007)*, pp. 259-268, IEEE, 2007.

[3] Agreement Technologies (AT) Project, www.agreement-technologies.org (último acceso 23/01/2012).

[4] Ossowski, S., "Coordination in Multi-Agent Systems: Towards a Technology of Agreement", *MATES, LNCS 5244*, pp. 2-12, Springer, 2008.

[5] Sierra, C., Botti, V.J., y Ossowski, S., "Agreement Computing", *KI*, vol. 25, nº 1, pp. 57-61, 2011.

[6] Argente, E., Botti, V., Carrascosa, C., Giret, A., Julian, V., y Rebollo, M., *An Abstract Architecture for Virtual Organisations: The THOMAS Project*, Technical report, DSIC, Universidad Politécnica de Valencia, 2008.

[7] DeLoach, S., "Moving multi-agent systems from research to practice", *International Journal of Agent-Oriented Software Engineering*, vol. 3, nº 4, pp. 378-382, 2009.

[8] Weyns, D., Helleboogh, A., y Holvoet, T., "How to get multi-agent systems accepted in industry?", *International Journal of Agent-Oriented Software Engineering*, vol. 3, nº 4, pp. 383-390, 2009.

[9] Ciancarini, P., "Coordination models and languages as software integrators", *ACM Computing Surveys*, vol. 28, nº 2, pp. 300-302, 1996.

[10] Papadopoulos, G.A., y Arbab, F., "Coordination models and languages". En Zelkowitz, M.V. (Ed.), *The Engineering of Large Systems*, vol. 46, pp. 329-400. Academic Press, 1998.

[11] Andersson, J., de Lemos, R., Malek, S., y Weyns, D., "Modelling Dimensions of Self-Adaptive Software Systems", *Software Engineering for Self-Adaptive Systems*, LNCS 5525, pp. 27-47, Springer, 2009.

[12] Casadei, M., y Viroli, M., "Applying self-organising coordination to the emergent tuple organisation in distributed networks". En Brueckner, S. et al. (Eds.), *2nd IEEE International Conference on Self-Adaptive and Self-Organising Systems – SASO*, 2008.

[13] Atienza, M., Schorlemmer, M., "I-SSA - Interaction-situated Semantic Alignment", *Proceedings International Conference on Cooperative Information Systems*, 2008.

[14] Sierra, C., Debenham, J., "Information-Based Agency", *Proceedings International Joint Conferences on AI (IJCAI-2007)*, pp. 1513-1518, AAAI Press, 2007.

[15] Pérez-Sotelo, J.S., Cuesta C.E., y Ossowski S., "Agreement Technologies for Adaptive, Service-Oriented Multi-Agent Systems", *Proceedings II Workshop on Agreement Technologies (WAT 2009) – XIII Conferencia de la Asociación Española para la Inteligencia Artificial CAEPIA2009*, vol. 635, CEUR Workshop Proceedings, 2009.

[16] FIPA Abstract Architecture Specification. *Technical Report SC00001L, Foundation for Intelligent Physical Agents*, FIPA TC Architecture, 2002.

[17] OSGi Alliance, www.osgi.org (último acceso 23/01/2012)

[18] OVAMAH - Organizaciones Virtuales Adaptativas: Técnicas y Mecanismos de Descripción y Adaptación, gti-ia.dsic.upv.es:8080/ovamah (último acceso 23/01/2012)

[19] Kolp, M., Giorgini, P., y Mylopoulos, J., "Multi-Agent Architectures as Organizational Structures", *Autonomous Agents and Multi-Agent Systems (AAMAS)*, vol. 13, nº 1, pp. 3-25, 2006.

[20] Weyns, D., Haesevoets, R., Helleboogh, A., Holvoet, T., y Joosen W., "The MACODO middleware for context-driven dynamic agent organizations", *Journal ACM Transactions on Autonomous and Adaptive Systems*, vol. 5, nº 1, pp. 1-28, 2010.

[21] Galloway, A.R., *Protocol: How Control Exists after Decentralization*, MIT Press, 2004.

[22] Pérez-Sotelo, J.S., Cuesta C.E., y Ossowski S., "Towards Adaptive Service Ecosystems with Agreement Technologies", *Proceedings I Workshop Adaptation in*

serVice EcosYsTems and ArchiTectures (AVYTAT2010) - OnTheMove Federated Conferences 2010, LNCS 6428, pp. 77-87, Springer, 2010.

[23] Cuesta, C.E., Pérez-Sotelo, J. S., y Ossowski, S., "Self-Organising Adaptive Structures: the Shifter Experience", *European Research Consortium for Informatics and Mathematics - ERCIM News*, nº 85, pp. 35-36, 2011.

[24] Billhardt, H., Centeno R., Fernandez A., Hermoso R., Ortiz R., Ossowski S., Perez-Sotelo J.S., Vasirani M., y Cuesta C.E., "Organisational Structures in Next-Generation Distributed Systems: Towards a Technology of Agreement", *Multiagent and Grid Systems: An International Journal*, vol. 7, nº 2-3, pp. 109–125, 2011.

[25] Ramírez, A. J., y Cheng, B. H. C., "Design Patterns for Developing Dynamically Adaptive Systems", *Proceedings ICSE2010-SEAMS*, pp. 49-58, 2010.

[26] Gamma, E., Helm, R., Johnson, R., y Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, 1994.

[27] Sangiorgi, D., Walker, D., "The Pi-calculus: a Theory of Mobile Processes". *Cambridge University Press*, 2003.

Análisis estático de código en el ciclo de desarrollo de software de seguridad crítica

Eduardo Lluna
Instituto Tecnológico de Informática (ITI)
elluna@iti.es

Resumen

El software es un elemento clave de los actuales sistemas de control incluidos los de seguridad crítica, en los que un fallo puede causar daños irreparables a personas o el entorno. Puesto que el software ni envejece ni se estropea, la calidad de éste dependerá principalmente de los defectos que se introduzcan en la fase de codificación. Por lo tanto cualquier técnica que permita eliminar estos defectos en la fase de creación permitirá aumentar la calidad a un coste más reducido. Las técnicas de Análisis Estático realizan esa función permitiendo localizar defectos sin ejecutar el código. Existen diversas técnicas y no siempre se pueden aplicar todas por razones de coste y tiempo. En este artículo se presenta una selección de las técnicas de análisis estático mínimas requeridas para un sistema de seguridad crítica en base a una norma y, puesto que estas técnicas son más eficientemente aplicadas por herramientas automáticas, se presenta un proceso de selección de estas herramientas en función de requisitos del proyecto.

Palabras clave: Calidad del software, Seguridad crítica, Análisis estático, EN-50128.

Static code analysis in the development cycle of safety critical software

Abstract

The software is a key element of control systems, including safety-critical, where failure could cause irreparable damage to persons or the environment. Software does not get aged or broken so its quality will largely depend on the number of defects introduced in the coding phase. Therefore any technique to avoid defects in the coding phase will increase software quality at a lower cost. Static Analysis techniques perform this function locating defects on the code without running it. There are several techniques but all of them cannot always be applied due to cost and time reasons. This article presents a minimum selection of static analysis techniques required for a safety critical system according a norm and, since these techniques are more efficiently applied by automated tools, a tool selection process based on project requirements is presented.

Key words: Software quality, Safety critical, Static analysis, EN-50128.

Lluna, E. "Análisis estático de código en el ciclo de desarrollo de software de seguridad crítica", REICIS, vol. 7, no.3, 2011, pp. 24-**. Recibido: 22-11-2011; revisado:1-12-2011; aceptado: 19-12-2011.

1. Introducción

Hoy en día el software es un elemento clave en los sistemas de control, incluidos los de seguridad crítica, aquellos cuyo fallo puede causar daños irreparables a personas, bienes o medio ambiente. Esta dependencia ha hecho que el nivel de fiabilidad requerido para este tipo de software sea muy elevado. La forma de conseguir un software de calidad suficiente es realizando desarrollos guiados por una serie de normas y estándares que fuercen el uso de prácticas seguras y controles de forma que se minimicen las posibilidades de error y permitan un seguimiento de la evolución del mismo. Estas normas dependen del área de aplicación final y afectan tanto al hardware como el software del sistema.

Este artículo presenta la técnica del Análisis Estático de código y selecciona un conjunto mínimo de técnicas específicas que puedan ser aplicadas en el contexto de los sistemas de seguridad crítica. Esta técnica, que permite detectar defectos en el código desarrollado sin necesidad de ejecutarlo, aparece recomendada en la mayoría de las normas de desarrollo de este tipo de sistemas. El hecho de basarse en estas normas, las cuales a su vez se basan en la experiencia y lecciones aprendidas a lo largo de muchos años de actividad, nos facilita el proceso de elección mediante la adopción de una serie de criterios predefinidos. El uso del Análisis Estático de código es altamente recomendable no sólo en el desarrollo de sistemas de seguridad crítica sino de cualquier tipo.

2. Desarrollo de sistemas de seguridad crítica

Las normas en uso para el desarrollo de software de sistemas de seguridad crítica dependen de la aplicación final. En el campo de la aeronáutica se usa en todo el mundo principalmente la norma DO-173B de la RCSA [1]. En Europa para el software de sistemas electrónicos en general la norma base es la IEC-61608-3 [2] a partir de la cual se derivan algunas otras normas para sistemas específicos como la EN-50128 [3] para sistemas ferroviarios, la norma IEC-61513 [4] para sistemas de centrales nucleares y la IEC-26262-6 [5] para el sector de la automoción. En temas de espacio, la NASA usa la norma propia NASA 8739.8 [6] mientras que otras agencias espaciales, incluida la ESA, usan la norma ECSS-ST-40C [7] de la ECSS para sus sistemas críticos.

Las normas mencionadas son sólo algunas de las existentes y, pese a la diversidad, todas comparten una estructura común forzando un ciclo de desarrollo y definiendo la

documentación requerida así como una serie de técnicas y buenas prácticas a seguir en cada una de las fases del desarrollo.

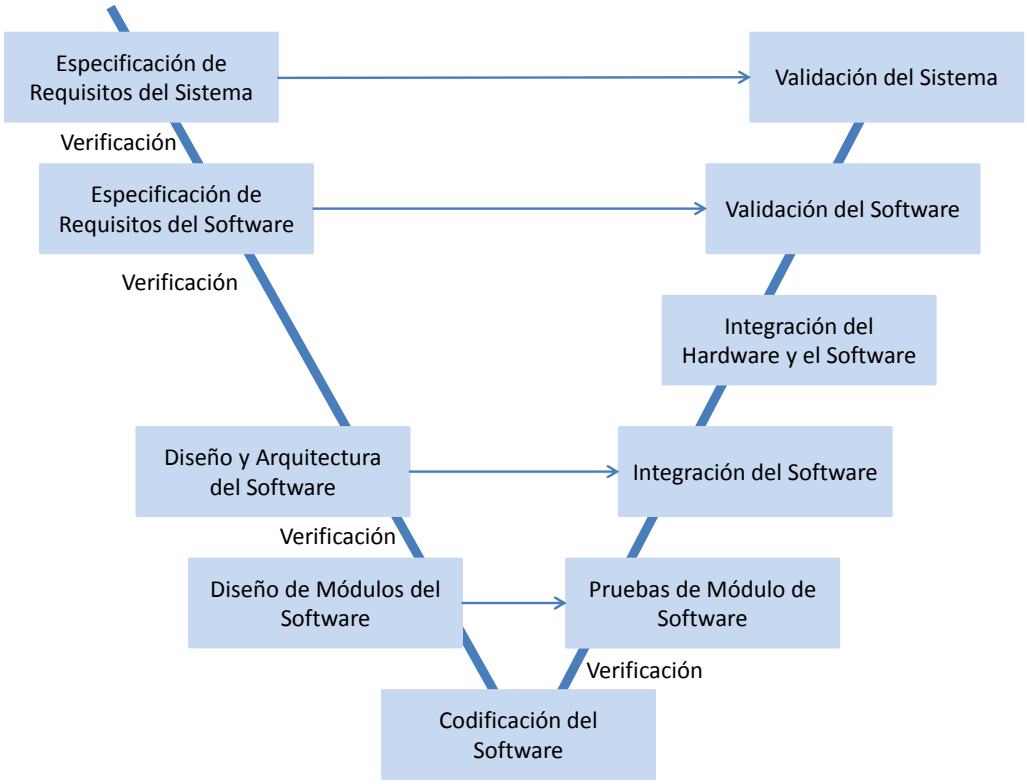

Figura 1: Ciclo de desarrollo del software definido por EN-50128.

Todos los sistemas no tienen la misma criticidad, por lo que las normas también definen una serie de niveles que van desde la ausencia de requisitos de seguridad hasta niveles máximos. Por ejemplo, la norma IEC-61608 define cuatro niveles de 1 a 4 de lo que llama *Safety Integrity Level* (SIL), siendo 1 el nivel mínimo de criticidad y 4 el máximo. Nivel 0 equivale a la ausencia de criticidad. Estos niveles de seguridad realmente se refieren a la seguridad del sistema como tal y se centran principalmente en el hardware definiendo tiempos medios antes de fallo. Desde el punto de vista del software estos niveles representan la peligrosidad o nivel de riesgo que produciría un fallo de software. En función del nivel de seguridad requerido las normas aplican unas técnicas u otras, por lo tanto un aspecto muy importante en el diseño es la elección adecuada del nivel de seguridad de la aplicación.

Vamos a centrarnos en la norma EN-50128 para aplicaciones ferroviarias la cual se basa en un ciclo de desarrollo en V (*V-model*) [8]. La Figura 1 muestra de forma esquemática este ciclo.

El ciclo de desarrollo incluye etapas habituales de especificación, diseño, codificación del software, integración de módulos, integración con el hardware y validación del software y del sistema. Pero un aspecto clave de la norma es el proceso de Verificación que se realiza siempre al terminar una de las fases y antes de comenzar la siguiente. Conviene tener clara la diferencia entre Validación y Verificación. De acuerdo con la definición de ambos términos que aparece en la norma ISO-9000 [9], la validación es la confirmación mediante la aportación de evidencia objetiva de que se han cumplido los requisitos para una utilización específica prevista, mientras que la verificación es la confirmación mediante la aportación de evidencia objetiva de que se han cumplido los requisitos especificados. Básicamente la validación tiene que ver con el uso final del sistema en una aplicación específica mientras que la verificación comprueba que en cada momento se ha hecho lo que se había dicho que se iba a hacer y los entregables de salida de una fase son los que se esperan en la siguiente.

3. Verificación del software

El software es un elemento del sistema que ni se estropea ni envejece. Eso significa que las posibles causas de fallo vendrán o bien de un fallo del hardware (sobre el que se ejecuta o de los periféricos que usa) o por un defecto en su desarrollo que hace que este no se comporte como se espera en función de las especificaciones en una situación particular. Para el caso de los fallos del hardware se emplean técnicas de tolerancia a fallos que principalmente se basan en redundancias de los componentes críticos y en las que no entraremos. La segunda fuente de fallos es más difícil de tratar y se basa en reducir al máximo posible, puesto que es prácticamente imposible erradicarlo al 100%, los defectos en el software. Es importante tener en mente que estamos asumiendo que la lógica especificada es correcta y que los problemas vienen de defectos que hacen que el software no siga la lógica definida. Si tenemos problemas en cuanto a la lógica del sistema (especificaciones de lo que debe hacer) nos encontramos en un caso de errores de diseño, los cuales deberían haberse detectado y subsanado en las etapas de diseño.

Por lo tanto es posible eliminar defectos del sistema verificando que el código desarrollado hace exactamente lo que se ha especificado que debe hacer. De esa forma en el uso real, cuando se den esas condiciones, hará lo que se espera de él.

Existen diferentes técnicas de verificación de software y la norma EN-50128 propone principalmente las siguientes: ensayos formales, ensayos probabilísticos, análisis estático y análisis dinámico. Para un nivel SIL4, el más estricto, todas estas

técnicas aparecen como Altamente Recomendables, mientras que para un nivel SIL1 sólo el análisis estático y el dinámico tienen esa consideración.

Aunque el uso de una única técnica de análisis no permite localizar todos los defectos de un sistema [13], en las siguientes secciones vamos a centrarnos en el análisis estático, ya que es una técnica que por sí misma es capaz de encontrar un número elevado de defectos en el software siendo una técnica relativamente sencilla de aplicar, por lo cual presenta un gran valor y es recomendada independientemente del nivel de seguridad.

4. Análisis Estático

El Análisis Estático es una evaluación del código generado para buscar defectos con la particularidad de que se realiza sin necesidad de ejecutar ese código.

Un aspecto importante para este análisis es el lenguaje de programación usado. La norma EN 50128 presenta una lista de lenguajes recomendables. En general son preferibles lenguajes altamente estructurados y muy restrictivos como el Ada, Modula-2 o Pascal, pero en la realidad, debido a la gran difusión del C/C++ en el sector industrial la mayoría de los desarrollos se realizan en este lenguaje y en Ada. La norma desaconseja el uso de C/C++ sin restricciones pero sí que permite el uso de estos lenguajes usando un subconjunto de los mismos y aplicando una serie de normas de codificación. El subconjunto más usado es el MISRA-C o MISRA-C++ [10] definido por la *Motor Industry Software Reliability Association* el cuál se considera seguro para aplicaciones de seguridad críticas.

En cuanto a las técnicas que pueden aplicarse para realizar este análisis la norma EN 50218 define un conjunto de éstas y entre las que marca como *Altamente Recomendables* para el nivel SIL4 se encuentran el análisis de valores extremos, el análisis del flujo de control y de datos, las revisiones de diseño y la ejecución simbólica del código. Estas técnicas tratan de revisar áreas que son conocidas como fuente de errores. Además hay que realizar una comprobación de las reglas de codificación que se adopten.

El análisis de valores extremos busca defectos en el manejo de las variables en los extremos de su rango de validez o en valores típicamente propensos a error como el cero en el caso de las divisiones o el uso de punteros. Igualmente se comprueban los accesos a *arrays* y elementos con un límite, fuentes habituales de problemas. El análisis del flujo de control busca problemas en la estructura lógica del programa los cuales

pueden ser reflejo de defectos. Por ejemplo, no debe haber fragmentos de código inalcanzables, los cuales se pueden deber a defectos en decisiones que impiden entrar en ciertas partes. También hay que evitar fragmentos de código de complejidad innecesaria puesto ésta puede ocultar problemas y verificar que todos los bucles tienen condición de salida. El análisis del flujo de datos busca errores en las estructuras de datos y en los accesos a las mismas. Los tipos de problemas habituales que deben comprobarse son, por ejemplo, la lectura de variables que no han sido previamente escritas, lo cual puede llevar a comportamientos indeseados, que no haya lecturas o escrituras a una misma variable seguidas, lo cual puede significar que falta código entre accesos. En general hay que comprobar cualquier operación con datos que pueda ser susceptible de enmascarar un problema. Las revisiones de diseños son procesos formales en los que un grupo de revisores comprueban el código usando un conjunto de casos de ensayo que son probados de forma manual sobre el código. Existen normas que definen los procesos de revisión formal y uno de los más conocidos, y sugerido por la norma, son las inspecciones de Fagan [11]. La ejecución simbólica consiste en sustituir a lo largo del software las expresiones del lado derecho e izquierdo de todas las asignaciones manteniendo nombre de variables en lugar de valores de forma que se obtenga al final una expresión para cada una de las variables. Esa expresión resultante se compara con la especificación para ver si coincide. Este proceso normalmente es largo y muy complejo por lo que esta técnica se limita a código relativamente simple.

La tabla 1 muestra a modo de resumen para cada una de las técnicas mencionadas, a excepción de la revisión formal y la ejecución simbólica, los aspectos más importantes que hay que comprobar siguiendo las recomendaciones de la norma EN-50128 para un sistema SIL4.

Técnica	Actividad
Reglas de codificación	Las que vengan definidas en la norma que se aplique (por ejemplo MISRA-C/C++)
Análisis de valores extremos	División por cero
	Uso de punteros nulos
	Uso del mayor valor posible de una variable
	Uso del menor valor posible de una variable
	Accesos fuera de rango en *arrays*
	Uso correcto de listas y *arrays* vacios
	Comprobación de rangos de parámetros en las funciones.

Análisis del flujo de control	Código accesible
	No existe código anudado (simplificable)
	Todos los bucles tienen condición de salida alcanzable
Análisis del flujo de datos	No se leen variables antes de ser iniciadas (salvo volátiles)
	No se escriben variables más de una vez antes de ser leídas (salvo volátiles)
	No se escriben variables que luego no se leen (salvo volátiles)

Tabla 1. Técnicas de Análisis Estático recomendadas por la norma EN 50128.

5. Métricas

Las métricas son indicadores cuantitativos del grado en que un componente, en este caso el software, posee un atributo dado. Al ser un indicador cuantitativo, un valor numérico, permite una comprobación fácil de si el valor está dentro de unos rangos y por lo tanto, de definir de forma clara un criterio de aceptación. Normalmente las métricas no identifican directamente defectos pero, por ejemplo, las métricas de complejidad del código, pueden dar una idea de las probabilidades de que haya más o menos defectos ocultos puesto que a mayor complejidad mayor probabilidad de que hayan defectos ocultos. Para cada métrica hay que definir también un rango de valores que se considera aceptables, los cuales dependerán de la aplicación y del nivel de seguridad requerido. Existen muchas métricas definidas y en uso pero para un sistema de seguridad crítica las más interesantes, por los motivos mencionados, son las de medida de la complejidad del código.

Métrica	Descripción	Rango
DCOM	Densidad de comentarios	>0.2
STCYC	Complejidad ciclomática	1-5
LEVEL	Nivel de anidamiento de funciones	0-4
CALLING	Número de veces que una función es llamada	0-5
CALL	Número de funciones llamadas por una función	0-5
PARAM	Número de parámetros de una función	0-5
RETURN	Número de puntos de retorno de una función	0-1
STNRA	Número de funciones con recursividad	0
NGTO	Número de instrucciones goto	0

Tabla 2. Métricas aplicables.

La tabla 2 muestra un conjunto de métricas a aplicar en sistemas críticos basada en la propuesta del *Hertseller Initiative Software* [12] para sistemas de automoción. Las

métricas seleccionadas miden principalmente la complejidad del código puesto que es el factor principal de cara a la presencia de defectos pero también incluye algunas relacionadas con el uso de técnicas de programación difíciles de seguir y depurar como la recursividad o de estructuras que incrementan la desorganización del código como el *goto*. La tabla también incluye los rangos válidos propuestos para cada métrica en sistemas con nivel SIL4.

El mantenimiento de las métricas dentro de los valores aceptados, si bien no evitan los errores, nos dan una indicación directa del nivel de complejidad y por lo tanto de la probabilidad de tener errores.

6. Proceso de selección de herramientas

El análisis estático puede realizarse de forma manual, de hecho, las inspecciones formales se realizan de esa forma. Aspectos como la comprobación de reglas de codificación y los análisis de valores extremos, flujo de control y datos se comprueban de forma más eficiente mediante herramientas de software que evitan que se cometan errores y aceleran el proceso.

Existen en el mercado una gran variedad de herramientas de análisis estático, tanto de código libre como comerciales pero para sistemas de seguridad crítica es necesario que estén certificadas, por lo que el número de estas se reduce considerablemente y por otra parte el coste de las mismas crece de forma importante. Por lo tanto, en base al proyecto en el que se vaya a aplicar, la elección de la herramienta adecuada es muy importante puesto que por una parte afectará a la cantidad de pruebas que haya que realizar de forma manual, lo cual impacta en la duración del proyecto, y por otra, al tener un coste elevado pueden llegar a ser una parte importante del presupuesto del proyecto.

Para realizar la selección de la herramienta de análisis se ha definido un proceso que se muestra en la Figura 2.

Lo primero y fundamental es tener perfectamente definido el uso de la herramienta, lo que incluye el lenguaje de programación, el nivel de seguridad requerido, las reglas de codificación y el conjunto de técnicas de análisis requeridas. En un proyecto de seguridad crítica la norma de aplicación marcará las pautas a seguir en estas decisiones. Una vez está clara esta información se prepara un plan de evaluación el cual incluye la definición de unas métricas que permitirán tomar la decisión final. En nuestro caso las métricas son el porcentaje de cobertura de comprobación de las reglas

de codificación y el porcentaje de cobertura de las técnicas de análisis incluidas en la tabla 1.

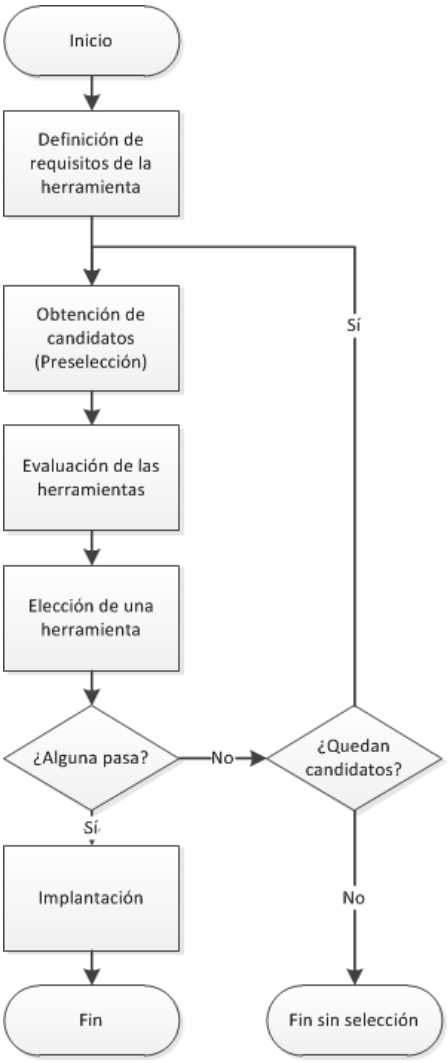

Figura 2: Proceso de selección de la herramienta

Métrica	Descripción	Rango
PTAE	Porcentaje de técnicas de Análisis Estático soportadas	>75%
PTAD	Porcentaje de técnicas de Análisis Dinámico soportadas	>75%
PRMC	Porcentaje de reglas MISRA C++ comprobadas	>75%
PMAE	Porcentaje de métricas de Análisis Estático calculadas	>80%
PMAD	Porcentaje de métricas de Análisis Dinámico calculadas	>80%
MCCY	Cálculo de complejidad ciclomática	Sí

Tabla 3. Métricas para la elección de la herramienta.

Para cada métrica también se definirá un rango de valores aceptables y el criterio de selección consistirá en seleccionar aquella herramienta que obtenga mayores valores en las métricas pero teniendo en cuenta que estas también tienen un valor mínimo de

aceptación y si no se llega a ese valor la herramienta en cuestión no puede ser tenida en cuenta. Estas métricas y sus valores se muestran en la tabla 3.

El siguiente paso es la selección de las herramientas que aparentemente pueden cumplir los requisitos básicos (preselección) y sobre las que se realizará la evaluación. Por ejemplo, sólo por el lenguaje de programación o por la certificación de seguridad requerida hay herramientas que se descartarán directamente. Por motivos de coste, principalmente en tiempo, lo normal es que no sea posible evaluar todas las herramientas que podrían servir al pasar la preselección, por lo que algunos candidatos potenciales pueden quedar sin evaluar. En esta preselección, cuando hay muchos candidatos y claramente hemos de quedarnos sólo con unos pocos, pueden aplicarse criterios de selección más o menos subjetivos en base a prestigio de marcas o experiencia previa. Es importante poder realizar la evaluación sobre una versión real de la herramienta puesto que hay una serie de factores 'no medibles' que pueden influir, principalmente ligados a la usabilidad de esta, por lo que es necesario en las herramientas comerciales obtener una licencia de evaluación. Puesto que el coste de las herramientas suele ser alto, es habitual que los fabricantes accedan a proporcionar versiones de evaluación de duración limitada.

Una vez conseguidas las herramientas se efectúa el proceso de evaluación mediante la ejecución del plan de pruebas definido con cada una de las herramientas preseleccionadas y se calculan las métricas. Esta etapa es la que más tiempo requiere puesto que las personas que realizan las pruebas deben de pasar por el periodo de aprendizaje inicial de las herramientas.

Finalizadas las pruebas y calculadas las métricas el proceso de selección es sencillo y se basa simplemente en la aplicación del criterio de selección previamente definido. Puesto que existen unos valores mínimos es posible que ninguna de las herramientas evaluadas supere la evaluación, en ese caso, si quedan candidatos potenciales no evaluados, se procederá a una nueva selección con estos. En el caso de que no queden candidatos, en principio no sería posible encontrar una herramienta con nuestros requisitos, por lo que se podría repetir el proceso de toma de decisión modificando el criterio de aceptación si decidimos que es preferible usar una herramienta con una cobertura limitada a no usar ninguna.

Con la herramienta seleccionada ya sólo queda la compra de la licencia final y la implantación de la misma. El proceso de implantación también puede llevar su tiempo puesto que todos los miembros del equipo que vayan a participar en el análisis deben

entrenarse en el uso de la nueva herramienta. Para ello es muy conveniente que en la fase de evaluación se vaya preparando una guía de cada herramienta que vaya recogiendo las dificultades que el evaluador ha ido encontrando en su propio proceso de aprendizaje. Si bien esto puede hacer que el proceso de evaluación tome más tiempo, permitirá acelerar la implantación y la difusión de la herramienta entre los miembros del equipo final.

Este proceso de selección se ha aplicado en un proyecto de seguridad crítica con nivel SIL4 desarrollado en C y C++ de acuerdo a la norma EN 50128. Los criterios de selección de la herramienta fueron los siguientes: certificada para SIL4, soporte del lenguaje C/C++, uso de reglas de codificación MISRA C/C++ y soporte de las técnicas de análisis estático mencionadas en la tabla 1. También se introdujeron una serie de requisitos para el análisis dinámico que no se ha incluido en este artículo. Con lo anterior se preparó el plan de evaluación y las métricas para tomar una decisión. El criterio de aceptación corresponde a los valores de la Tabla 3. La ejecución del plan consistió en realizar los análisis requeridos usando las herramienta bajo prueba a un proyecto interno bien conocido para comprobar si es posible obtener con ella los parámetros requeridos.

Se preseleccionaron cuatro herramientas comerciales que cumplían los requisitos previos y de las cuales fue posible obtener una licencia de evaluación. Se dedicó una única persona a la ejecución del plan de evaluación de forma que las evaluaciones se realizaron en serie. El desarrollo del plan de evaluación requirió 6 semanas de trabajo. Este periodo también puede ser considerado como de formación del evaluador en el manejo de esas herramientas y para la redacción de la documentación de uso de la misma. Esta documentación preparada mientras se realiza la evaluación es usada posteriormente para la formación de los demás miembros del grupo. Finalizada la evaluación, la selección de la herramienta más adecuada se realizó de forma objetiva e inmediata aplicando las métricas.

7. Conclusiones

En este artículo se ha visto el papel de la verificación en el ciclo de desarrollo en V típicamente usado en los sistemas de seguridad crítica en el caso particular de la norma EN 50128. En particular la verificación del código generado tiene una gran importancia puesto que las principales causas de fallo del software, descartadas las debidas a problemas con el hardware o el diseño, son defectos introducidos en la fase de creación

del mismo. Esta verificación es vital para eliminar la mayor parte de defectos en la fase más temprana posible y poder asegurar los niveles de seguridad y fiabilidad requeridos por la aplicación.

Dentro de las técnicas de verificación de código el análisis estático es una técnica relativamente sencilla y fácilmente automatizable, incluida como altamente recomendable en todas las normas de sistemas de seguridad crítica y que permite mejorar la calidad del software desde el momento mismo de su creación. Es conocido que el coste de reparación de un defecto crece conforme avanzamos en el ciclo de desarrollo, por lo que encontrar los defectos en la misma fase de la escritura del código es el momento menos costoso. Aunque una única técnica no garantiza el descubrir todos los problemas, la técnica tratada es una de las de más fácil adopción y que más ventajas aporta.

Existen multitud de herramientas que permiten realizar de forma automática este análisis de código, pero cuáles pueden ser usadas en un proyecto específico depende de ciertos parámetros del proyecto, como son el lenguaje de programación usado y el nivel de seguridad. Algunas de estas herramientas, sobre todo las que están certificadas para aplicaciones de seguridad crítica, son caras. Luego debido a la gran variedad y al coste de las mismas, la elección de la herramienta adecuada es importante para el éxito del proyecto. Se ha presentado un proceso de selección sencillo pero que permite realizar esta selección de una forma rigurosa.

Para sistemas que no sean de seguridad crítica, el análisis estático es también una herramienta importante para aumentar la calidad de los productos de software a un coste relativamente bajo por lo que debería estar incluida en cualquier ciclo de desarrollo de software.

Referencias

[1] RTCA, *DO-178B. Software Considerations in Airborne Systems and Equipment Certification*, RTCA, 1992.

[2] IEC, *IEC-61508-3 Seguridad funcional de los sistemas eléctricos, electrónicos y electrónicos-programables relacionados con la seguridad. Parte 3: Requisitos del Software (soporte lógico)*, IEC, 2004.

[3] AENOR, *UNE-EN-50128 Aplicaciones ferroviarias. Sistemas de comunicación, señalización y procesamiento. Software para sistemas de control y protección de ferrocarril*, AENOR, 2002.

[4] IEC, *IEC-61513 Nuclear power plants. Instrumentation and control for systems important to safety. General requirements for systems*, IEC, 2011.

[5] IEC, *IEC-26262-6 Road vehicles – Functional safety. Part 6. Product development. Software Level*, IEC, 2009.

[6] NASA. *NASA 8739.8 Software assurance standard*, NASA, 2004.

[7] ECSS. *ECSS-ST-40C Space Engineering. Software*, ECSS, 2009.

[8] Broekman, B. y Nothenboom, E., *Testing Embedded Software*, Addison-Wesley, 2003.

[9] AENOR. *UNE-EN ISO 9000 Sistemas de gestión de la calidad. Fundamentos y vocabulario*, AENOR, 2005.

[10] MISRA. *Motor Industry Software Reliability Association: Guidelines for the use of the C language in critical systems*. MISRA, 2004.

[11] Fagan, M.E., "Advances in Software Inspections", *IEEE Transactions on Software Engineering*, vol. 12, nº 7, pp 744-751, 1986.

[12] HIS, *Source Code Metrics*, Herfseller Initiative Software, 2008.

[13] Zheng, J., Nagappan, N., Hudepohl, J.P., Vouk, M.A., On the value of static analysis for fault detection in software. *IEEE Transactions on Software Engineering*, vol. 32, nº 4, pp 240-253, April 2006.

REICIS

ACTUALIDAD INVITADA

Perspectivas de la mejora de procesos de software

José Antonio Calvo-Manzano Villalón
Director de la Cátedra de Mejora de Procesos Software en el Espacio Iberoamericano (MPSEI)

La cátedra MPSEI

El software se ha convertido en un aspecto estratégico en cualquier tipo de negocio. Por ello, la mejora del proceso software se ha convertido en una ventaja estratégica en la mayoría de las organizaciones. Por tanto, la investigación en los distintos aspectos de mejora en universidades y centros de investigación, así como la participación de las empresas y de las administraciones y los organismos públicos, son fundamentales para la implantación con éxito de los proyectos de mejora de procesos.

La Cátedra de Mejora de Procesos Software en el Espacio Iberoamericano (MPSEI) de la Universidad Politécnica de Madrid (UPM) nació en el año 2005 y está actualmente patrocinada por la Fundación Everis. Pretende ser un elemento estabilizador de la investigación en este campo así como un catalizador entre las investigaciones llevadas a cabo en la Universidad y las necesidades del mundo empresarial.

Consideramos importante la involucración de las empresas patrocinadoras de forma activa en los trabajos a desarrollar. Por ello, dentro de este dominio de actuación deberían intervenir tanto en la definición de las líneas de investigación (plasmadas, en la mayoría de los casos, en el desarrollo de tesis doctorales) como en la validación de los resultados (habitualmente reflejada en los casos de estudio de dichas sesis).

La cátedra MPSEI tiene como objetivo general la investigación, adaptación y difusión de las técnicas de mejora del proceso de software, la promoción de actividades docentes y de investigación interdisciplinar, la transferencia de conocimientos y de resultados tecnológicos, y la financiación de becas (principalmente a alumnos de doctorado/máster) en el área de las Tecnologías de la Información y de las Comunicaciones, destacando especialmente sus aspectos más innovadores en el ámbito de la mejora de procesos de software.

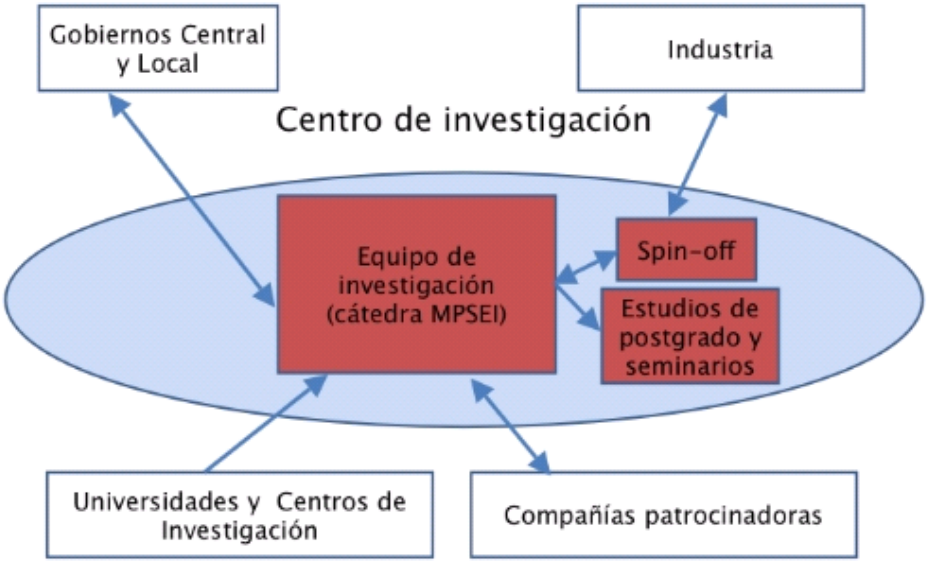

Figura 1. Entorno funcional

Para lograr este objetivo, se ha establecido el entorno funcional mostrado en la Figura 1, que podría ser útil para cualquier "cátedra" en general. Así, la cátedra MPSEI se relaciona con:

- la Universidad a efectos de impartir y actualizar la docencia reglada y los estudios de postgrado: en nuestro caso particular, la UPM);
- la Universidad: actualmente, tenemos un convenio de colaboración con la Universidad de Medellín en Colombia.
- centros de Investigación: actualmente, se está colaborando con el Software Engineering Institute: SEI de la Universidad de Carnegie Mellon; también, se va a firmar un convenio de colaboración con el Centro de Investigación en Matemáticas –CIMAT ubicado en Zacatecas, México para la colaboración en proyectos de investigación;
- las compañías patrocinadoras (en este caso particular, Everis) a efectos de priorizar y experimentar las investigaciones;
- la Administración para la realización de eventos, desarrollo de modelos, así como para la diseminación de conocimientos de interés para la Administración (actualmente, se está colaborando con el Instituto Nacional de Tecnologías de la Comunicación -INTECO)

- la Industria para recoger sus necesidades y llevar a cabo acciones que mejoren su competitividad y diseminen los resultados de las investigaciones de la cátedra.

Las actividades de la cátedra

Entre las actividades generales que se llevan a cabo dentro del ámbito de la Cátedra MPSEI, se encuentran:

- Proponer líneas de trabajo entre las prioridades y necesidades de las empresas (patrocinadoras o no) y las de la UPM. Estas líneas de trabajo constituyen las tesis doctorales de nuestros alumnos de doctorado.
- Facilitar la máxima difusión de los resultados de investigación y, en general, de todas las actividades realizadas en el ámbito de la Cátedra. Para ello, se publican resultados de las investigaciones en congresos y revistas relativas a mejora de procesos.

Así mismo, la cátedra participa en la organización de congresos como parte del comité organizador local. Actualmente, participamos en

- La Conferencia SEPG Europe 2012 (5-7 Junio) como "Local Host Committee"
- La Conferencia CISTI 2012 (20-23 Junio) como Comité Organizador

Al igual que en el año 2011, también participamos (junto con INTECO y Everis) en la organización del I Encuentro Hispano Mexicano de calidad del software, celebrado en Monterrey (México), para el año 2012 (octubre) se pretende organizar el II Encuentro en León (España).

Además, se han abordado las siguientes líneas de actividad

- Colaborar con otras Cátedras, grupos de investigación, departamentos, instituciones o centros que compartan objetivos similares.
- Actualmente, estamos colaborando con el SEI en la traducción al castellano de uno de los modelos de procesos más difundidos a nivel mundial, el CMMI-DEV v1.3. Para ello, se cuenta con el apoyo de Everis, INTECO y CIMAT como patrocinadores, de Accenture-Coritel e IECISA como colaboradores, y de CERASA (CEURA) como editor.

- Promover el patrocinio de becas entre los alumnos de doctorado/máster/ingeniería/grado de la UPM. En este caso, Everis está permitiendo que nuestros alumnos hagan prácticas reales en la empresa.

Líneas de investigación y futuro

Las líneas de investigación, desde el punto de vista de los procesos, que actualmente tienen un mayor interés son:

- Entornos ágiles: todo lo relacionado con las aproximaciones ágiles como, por ejemplo, SCRUM.
- Control estadístico de procesos: aspectos relativos a la aplicación del control estadístico a los procesos (por ejemplo, Lean, Six Sigma).
- Entornos multi-modelo: esta línea se relaciona con la aplicación de diferentes modelos en una organización, ya sea a nivel del modelo global (por ejemplo, utilizar CMMI-DEV para desarrollo, y CMMI-SVC o ITIL para la parte de servicios) o bien a nivel de un proceso particular (por ejemplo, a nivel de gestión de proyectos, aplicar la gestión de proyectos de acuerdo a PMBOK, PRINCE2, TSP, los procesos relativos a gestión de proyectos de CMMI-DEV, etc.).
- Aspectos organizativos y humanos: aspectos relativos a la gestión del cambio, trabajo en equipo (virtuales o no), capacidades de los equipos y evolución de los puestos de trabajo (qué capacidades debe tener cada uno de los miembros de un equipo y cómo obtener esa capacitación), nuevos canales de comunicación. Asimismo, incluimos aquí el "despliegue de procesos" por su estrecha relación con los factores humanos.
- Medianas, pequeñas y micro empresas (MPMEs): aplicación de los modelos de procesos, orientados generalmente a las grandes empresas, a las MPMEs.

En un futuro cercano, nos atrevemos a indicar la importancia que tendrá el desarrollo global de software (*global software development*) unido a otras líneas de investigación como por ejemplo las aproximaciones ágiles y los entornos multimodelos (o incluso modelos sectoriales).

Desde estas páginas, agradecer a todos los patrocinadores y colaboradores por su participación en la Cátedra MPSEI y alentar a otras Universidades y empresas a la

creación de nuevas Cátedras, ya en España o en países latinoamericanos, de forma que podamos crear una red de compartición del conocimiento. Esta red permitiría que tanto las Universidades/Centros de Investigación como las Administraciones y las empresas se vean beneficiadas de estas relaciones.

Perfil profesional

Jose A. Calvo-Manzano Villalón es doctor en Informática (2000). Actualmente es profesor contratado doctor en la Facultad de Informática de la Universidad Politécnica de Madrid, donde imparte su docencia en el área de Ingeniería de Software. Ha participado en más de 20 proyectos de investigación (a nivel europeo, administración pública española y con empresas privadas). Es autor de numerosas publicaciones y miembro de comités de programas de conferencias como EuroSPI, CISTI así como miembro del Consejo Científico de revistas como REICIS o RISTI. Actualmente es el director de la Cátedra de Mejora de Procesos Software en el Espacio Iberoamericano (MPSEI) de la UPM. Ha participado en la traducción al español de CMMI-DEV v1.2 y v1.3. También posee las certificaciones ITIL v2 y v3 Foundation, así como la certificación CMDB Foundation.