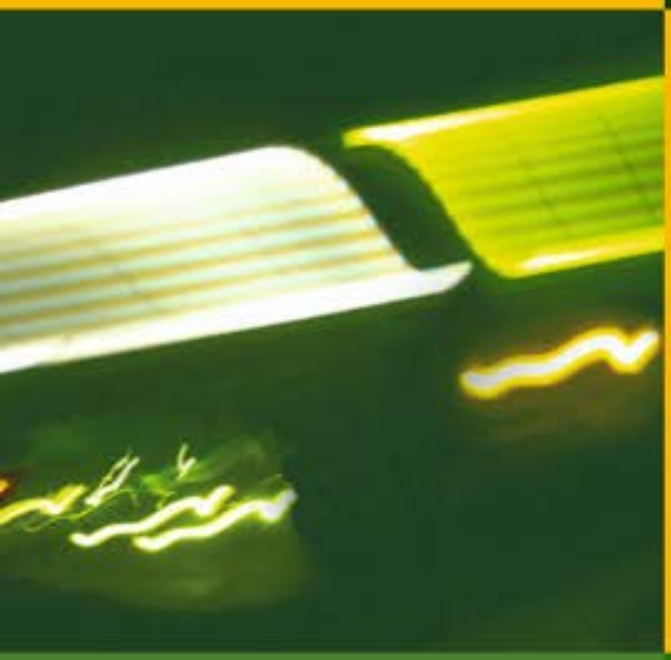


Ian Gorton

Essential Software Architecture

Second Edition

Springer

Arquitectura de Software esencial

Ian Gorton

Programas esenciales
Arquitectura

Segunda edicion

Springer

Ian Gorton
laboratorio Fellow
Pacífico Northwest National Laboratory PO Box
999 MSIN: K7-90

Richland, WA 99352 EE.UU.

ian.gorton@pnl.gov

ACM Computing Clasi fi cación (1998): D.2

ISBN 978-3-642-19175-6 e-ISBN 978-3-642-19176-3
DOI 10.1007 / 978-3-642-19176-3
Springer Heidelberg Dordrecht Londres Nueva York

Biblioteca del Congreso de control el número: 2011926871

Diseño de la cubierta: KuenkelLopka GmbH

Impreso en papel libre de ácido

Springer es parte de Springer Science + Business Media (www.springer.com)

Prefacio

Bienvenidos a la segunda edición de la arquitectura de software esencial. Está a 5 años desde la primera edición fue publicada, y en el mundo de la arquitectura de software, de 5 años es mucho tiempo. Por tanto, esta versión actualizada, con capítulos actualizados para capturar los nuevos avances en métodos y tecnologías, y relacionar experiencias relevantes de la práctica. Hay nueva arquitectura material de cobertura de la empresa, el desarrollo ágil, tecnologías de bus de servicios empresariales y servicios Web RESTful. Todos los capítulos tienen una lista actualizada y más extensa de la bibliografía recomendada, capturando a muchos de los mejores nuevos libros, documentos, sitios web y blogs, que yo sepa.

En particular, el completamente nuevo cap. 10 proporciona un estudio de caso sobre el diseño de la tecnología de Medici, que se extiende de un bus de servicios empresariales de código abierto con un modelo de programación basado en componentes. La tecnología Medici es de código abierto y libremente descargable (http://www.medici.pnl.gov), Por lo que es una herramienta muy adecuada para la enseñanza de los conceptos avanzados de middleware y la arquitectura que se describen en este texto.

En su corazón, sin embargo, esto sigue siendo un libro que tiene como objetivo impartir de manera sucinta un amplio alcance de los conocimientos de arquitectura de software relativo a los sistemas construidos a partir de tecnologías middleware convencionales. Esto incluye un espectro grande y diversa de los sistemas, que van sitios de comercio electrónico basado en fromWeb a SCienTI sistemas de análisis de datos financieros de alto rendimiento de gestión de datos c fi y.

Motivación

Lo que no ha cambiado en los últimos 5 años es que muchos proyectos con los que trabajo u opinión carecen de una noción explícita de un diseño arquitectónico. Requisitos funcionales suelen ser capturadas utilizando técnicas tradicionales o ágiles, estuvieron de acuerdo con las partes interesadas, y controlarán mediante métodos iterativos o cascada altamente tradicionales. Pero los problemas de arquitectura, el “cómo” de la aplicación logra su propósito, el “qué” sucede cuando las cosas cambian y evolucionan o fallan, son con frecuencia implícita (esto significa que están en la cabeza de alguien, tal vez) en el mejor. En el peor, simplemente no se abordan en forma alguna que pueda ser descrito en términos que no sean accidentales. Con frecuencia, cuando pido una visión general de la arquitectura de la aplicación y la conducción no funcional

requisitos en la primera reunión técnica, la gente comienza a dibujar en una pizarra. O me muestran el código y sumergirse en las iternals de la implementación basada en torno a su tecnología de moda preferido. Cualquiera de ellos es rara vez es una buena señal.

Los problemas y riesgos de las prácticas arquitectónicas pobres son bien conocidos y documentados dentro de la profesión de la ingeniería de software. Un gran cuerpo de conocimiento excelente de arquitectura es capturado en libros ampliamente accesibles, revistas e informes de los miembros del Instituto de Ingeniería de Software (SEI), Siemens y otras instituciones industriales y académicos de renombre.

Aunque el foco de gran parte de esta literatura es altamente sistemas técnicos, tales como la aviónica, fl ight de simulación y de conmutación de telecomunicaciones, este libro se inclina más a la corriente principal del mundo de las aplicaciones de software. En cierto sentido, se cierra la brecha entre las necesidades de la gran mayoría de los profesionales de software y el cuerpo actual de los conocimientos en arquitectura de software. Específicamente:

- Proporciona discusiones clara y concisa sobre los temas, técnicas y métodos que están en el centro de estudios de arquitectura de sonido.
- Se describe y analiza el componente de propósito general y middleware tecnogías que respaldan muchos de los patrones arquitectónicos fundamentales utilizados en aplicaciones.
- Se espera que los cambios en las tecnologías y prácticas pueden afectar a la próxima generación de sistemas de información empresarial.
- Utiliza sistemas de información conocidos como ejemplos, tomados del autor de tecnología de software. Estos son adecuados para los arquitectos existentes y el experiencias en los sistemas de información de banca, el comercio electrónico y gobierno.
- Proporciona muchos punteros y referencias a los trabajos existentes sobre la arquitectura de software.

Si usted trabaja como un arquitecto o diseñador mayor, o si desea 1 día, este libro debería ser de valor para usted. Y si usted es un estudiante que está estudiando ingeniería de software y necesita una visión general del campo de la arquitectura de software, este libro debería ser una fuente de primera accesible y útil de información. Ciertamente no le dirá todo lo que necesita saber - que va a tomar mucho más que puede ser incluido en un libro de tal longitud modesta. Pero pretende transmitir la esencia del pensamiento arquitectónico, prácticas y tecnologías de apoyo, y posicionar al lector a profundizar en las áreas que son pertinentes a su vida e intereses profesionales.

contorno

El libro está estructurado en tres secciones básicas. La primera es de carácter introductorio, y accesible por un lector relativamente no técnico que quieran una visión general de la arquitectura de software.

La segunda sección es la más técnica en la naturaleza. En él se describen las habilidades esenciales y conocimientos técnicos que necesita un arquitecto de TI.

La tercera tiene visión de futuro. Seis capítulos cada introducen un área emergente de la práctica o la

diseñadores, así como las personas que han leído las dos primeras secciones, y que deseen adquirir conocimientos sobre el futuro de las influencias de su profesión.

Más específicamente:

- **Los capítulos 1 - 3: Estos capítulos proporcionan el material introductorio para el resto de** el libro, y el área de la arquitectura de software en sí. Capítulo 1 analiza los elementos clave de la arquitectura de software, y describe el papel de un arquitecto de software. El capítulo 2 presenta los requisitos para un problema de estudio de casos, un diseño para el que se presenta en el Cap. 9. Esto demuestra el tipo de problema y la descripción asociada que un arquitecto de software normalmente trabaja. Capítulo 3 analiza los elementos de algunos atributos de calidad clave como la escalabilidad, el rendimiento y la disponibilidad. Arquitectos pasan mucho tiempo frente a los requisitos de atributos de calidad para las aplicaciones. Por lo tanto es esencial que estos atributos de calidad se conocen bien, ya que son elementos fundamentales de los conocimientos de un arquitecto.

- **Los capítulos 4 - 10: Estos capítulos son la columna vertebral técnica del libro. Capítulo 4** introduce una gama de tecnologías middleware fundamentales que los arquitectos aprovechan comúnmente en soluciones de aplicación. Capítulo 5 está dedicado a describir servicios Web, incluyendo tanto SOAP y los enfoques basados en REST. Capítulo 6 se basa en los capítulos anteriores para explicar plataformas middleware avanzadas, como las tecnologías de bus de servicios empresariales. El capítulo 7 presenta un proceso iterativo de arquitectura de software en tres etapas que se pueden adaptar para ser tan ágil como un proyecto requiere. En él se describen las tareas esenciales y documentos que implican un arquitecto. Capítulo 8 analiza la documentación de la arquitectura, y se centra en las nuevas notaciones disponibles en la versión de UML 2.0. Capítulo 9 reúne la información en los primeros 6 capítulos, mostrando cómo las tecnologías de middleware se puede utilizar para hacer frente a los requisitos de atributos de calidad para el estudio de caso. También demuestra el uso de la plantilla de la documentación descrita en el cap. 8 para la descripción de una arquitectura de la aplicación. Capítulo 10 proporciona otro caso práctico que describe el diseño de la fuente abierta Medici Integration Framework, que es un API especializado para la creación de aplicaciones estructuradas como las tuberías de componentes.

- **Los capítulos 11 - 15: Estos capítulos se centran en cada una técnica emergente o tecno-** gía que probablemente influyen en el futuro de los arquitectos de software. Estos incluyen líneas de productos de software, la arquitectura dirigida por modelos, la arquitectura orientada a aspectos y la Web Semántica. Cada capítulo presenta los elementos esenciales del procedimiento o tecnología, se describe el estado de la técnica y especula acerca de cómo puede afectar a las habilidades y prácticas requeridas de un arquitecto de software creciente adopción. Cada capítulo se refiere también su enfoque a una extensión del estudio de caso ICDE en el Cap. 9.

Richland, WA, EE.UU. Ian Gorton

De diciembre de 2010

Expresiones de gratitud

En primer lugar, gracias a los contribuyentes de los capítulos que han ayudado a proporcionar el contenido de las líneas de producto de software (Marcos grapas), la programación orientada a aspectos (Jenny Liu), basada en modelos de desarrollo (Liming Zhu), servicios Web (Paul Green de campo) y la Web Semántica (Judi Thomson). Adam Wynne también coautor del capítulo sobre Medici. Sus esfuerzos colectivos y la paciencia son muy apreciadas.

datos de contacto de los autores que han contribuido son los siguientes: Dr. Mark Staples, Australia Nacional de TIC, e-mail: mark.staples@nicta.com.au Dr. Liming Zhu, Australia Nacional de TIC, e-mail: liming.zhu@nicta.com.au Dr. Yan Liu, Pacífico noroeste Nacional de Laboratorio, EE.UU., correo electrónico: jenny.liu@nicta.com.au Adam Wynne, Pacífico noroeste Nacional de Laboratorio, EE.UU., correo electrónico: @ adam.wynne pnl.gov

Paul Green campo, Escuela de TI, CSIRO, Australia, correo electrónico: paul.green fi eld@csiro.au Dr. Judi McCuaig, Universidad de Guelph, Canadá, e-mail: judi@cis.uguelph.ca También me gustaría agradecer a todos en Springer quien ha ayudado a hacer realidad este libro, especialmente el editor, Ralf Gerstner.

También me gustaría agradecer a los muchos talentos de software arquitectos, ingenieros e investigadores que he trabajado de cerca con los recientemente y / o que han ayudado a dar forma a mi pensamiento y experiencia a través de discusiones geek largas y entretenidas. Sin ningún orden en particular, estos son Anna Liu, Paul Green campo, de Shiping Chen, Paul Brebner, Jenny Liu, John Colton, Karen Schchardt, Gary Negro, Dave Thurman, Jereme Haack, Sven Overhage, John Grundy, Muhammad Ali Babar, Justin Almquist, Rik poco campo, Kevin Dorow, Steffen Becker, Ranata Johnson, Len Bass, Lei Hu, Jim Thomas, Deb Gració, Nihar Trivedi, Paula Cowley, JimWebber, Adrienne Andrew, Dan Adams, Dean Kuo, John Hoskins, Shuping Ran, Doug Palmer, Nick Cramer, Liming Zhu, Ralf Reussner, Marcos Hoza, Shijian Lu, Andrew Cowell, Tariq Al Naeem, Wendy Cowley y Alan Fekete.

Contenido

Capítulo 1

Entender la arquitectura del software

1.1 ¿Qué es la Arquitectura de Software?

Los últimos 15 años han visto un tremendo aumento de la prominencia de una subdisciplina de ingeniería de software conocida como arquitectura de software. Arquitecto Técnico y Arquitecto en jefe son los títulos de trabajo que ahora abundan en la industria del software. Hay una Asociación Internacional de Software Architects, [1] e incluso una cierta friki más rico conocido en la tierra solía tener "arquitecto" en su puesto de trabajo en su mejor momento. No puede ser una mala actuación, entonces?

Tengo la sospecha de que la "arquitectura" es uno de los más usados en exceso y menos entendida en términos de desarrollo de software círculos profesionales. Lo escucho mal utilizada regularmente en diversos foros tales como la revisión de proyectos y debates, presentaciones de trabajos académicos en conferencias y lanzamientos de productos. Usted sabe que un término se está convirtiendo poco a poco vacía cuando se convierte en parte de la lengua vernácula de la fuerza de ventas de la industria de software.

Este libro es acerca de la arquitectura de software. En particular, se trata de los problemas de diseño y tecnología clave a considerar en la construcción de sistemas del lado del servidor que procesan las solicitudes múltiples y simultáneas de los usuarios y / u otros sistemas de software. Su objetivo es describir de manera concisa los elementos esenciales del conocimiento y las habilidades clave que se requieren para ser un arquitecto de software en la industria de software y tecnología de la información (IT). La concisión es un objetivo clave. Por esta razón, de ninguna manera todo lo que un arquitecto tiene que saber serán cubiertos. Si desea o necesita saber más, cada capítulo le apuntan a los recursos valiosos y útiles adicionales que pueden conducir a mucho mayor iluminación.

Así, sin más preámbulos, vamos a tratar de fi gura lo que, al menos en mi opinión, la arquitectura de software que realmente es, y esto es importante, no lo es. El resto de este capítulo se abordará esta cuestión, así como brevemente la introducción de las tareas principales de un arquitecto, y la relación entre la arquitectura y la tecnología en aplicaciones de TI.

[1] http://www.iasahome.org/web/home/home

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3_1, # Springer-Verlag Berlin Heidelberg 2011

1.2 Definiciones de Arquitectura de Software

Tratando de definir un término como arquitectura de software es siempre una actividad potencialmente peligrosa. Realmente no hay ampliamente aceptada definición por la industria. Para entender la diversidad de puntos de vista, tienen un navegar por la lista mantenida por el Instituto de Ingeniería de Software.[2] Hay mucho. La lectura de estos me recuerda una cita anónima que escuché en un programa de radio satírica recientemente, lo que fue más o menos en la línea de “la razón debate académico es tan vigorosa es que hay tan poco en juego”.

No tengo intención de añadir a este debate. En su lugar, vamos a examinar tres de fi niciones. Como miembro de la IEEE, que, por supuesto, naturalmente, comienzan con el fi nición de adoptada por mi cuerpo profesional:

> La arquitectura es definida por la práctica recomendada como la organización fundamental de un sistema, encarnado en sus componentes, sus relaciones entre sí y con el medio ambiente y los principios que gobiernan su diseño y evolución.
>
> [ANSI / IEEE Std 1471-2000, Práctica recomendada para la descripción arquitectónica de los sistemas de software intensivo]

Esto sienta las bases para una comprensión de la disciplina. Arquitectura capta la estructura del sistema en términos de componentes y cómo interactúan. También las reglas de diseño de todo el sistema de fi nes y considera cómo un sistema puede cambiar.

A continuación, siempre vale la pena conseguir el último punto de vista de algunos de los principales pensadores en el campo.

> La arquitectura de software de un programa o sistema de computación es la estructura o estructuras del sistema, que comprenden elementos de software, las propiedades externamente visibles de esos elementos, y las relaciones entre ellos.
>
> [L.Bass, P.Clements, R.Kazman, Arquitectura de software en la práctica (2ª edición), Addison-Wesley 2003]

Esto construye un poco en el ANSI / IEEE definición anterior, especialmente en lo que hace el papel de abstracción (es decir, propiedades externamente visibles) en una arquitectura y múltiples vistas arquitectura (estructuras del sistema) explícitos. Compare esto con otro, de Garlan y Shaw de principios de trabajo influyentes:

> [Arquitectura de software va] más allá de los algoritmos y estructuras de datos de la computación; el diseño y la especificación de la estructura general del sistema surge como un nuevo tipo de problema. problemas estructurales incluyen la organización bruta y la estructura global de control; protocolos de comunicación, la sincronización, y de acceso a datos; asignación de funcionalidad a elementos de diseño; distribución física; composición de elementos de diseño; de escala y rendimiento; y la selección entre alternativas de diseño.
>
> [RE. Garlan, M. Shaw, Una introducción a la arquitectura de software, Los avances en Ingeniería de Software e Ingeniería del Conocimiento, Volumen I, Mundial de la Ciencia, 1993]

Es interesante fijarse en ellos, ya que hay mucho en común. Incluyo el tercer principalmente ya que es más explícito acerca de ciertas cuestiones, como la escalabilidad y

---

2 http://www.sei.cmu.edu/architecture/de fi nitions.html

distribución, que están implícitos en las dos primeras. En cualquier caso, el análisis de éstos un poco hace que sea posible extraer algunas de las características fundamentales de arquitecturas de software. Estos, junto con algunos enfoques clave, se describen a continuación.

1.2.1 Arquitectura De multas Estructura

Gran parte del tiempo de un arquitecto tiene que ver con cómo particionar con sensatez una aplicación en un conjunto de componentes interrelacionados, módulos, objetos o cualquier unidad de partición de software que funcione para usted. [3] Diferentes requisitos de aplicación y limitaciones serán de definir el significado exacto de "sensatez" en la frase anterior - una arquitectura debe estar diseñado para cumplir con los requisitos especí fi cas y las limitaciones de la aplicación que se destina.

Por ejemplo, un requisito para un sistema de gestión de la información puede ser que la aplicación se distribuye a través de múltiples sitios, y una restricción es que cierta funcionalidad y los datos deben residir en cada sitio. O bien, la funcionalidad de una aplicación debe ser accesible desde un navegador web. Todos éstos imponen algunas limitaciones estructurales (sitio-específico, el servidor web alojado), y al mismo tiempo se abren vías para una considerable creatividad en el diseño de la partición de la funcionalidad a través de una colección de componentes relacionados.

En la partición de una aplicación, el arquitecto asigna responsabilidades a cada componente constituyente. Estas responsabilidades definen las tareas de un componente es un indicio fiable para llevar a cabo dentro de la aplicación. De esta manera, cada componente juega un papel fi específica en la aplicación, y el conjunto de componentes en general que comprende la arquitectura colabora para proporcionar la funcionalidad requerida.

Responsabilidad impulsada por el diseño (véase Wirfs-Brock en Lectura adicional) es una técnica de la orientación a objetos que puede ser utilizado de manera efectiva para ayudar a de fi ne los componentes clave en una arquitectura. Proporciona un método basado en herramientas informales y técnicas que hacen hincapié en el modelado del comportamiento usando objetos, responsabilidades y colaboraciones. He encontrado este gran ayuda en los proyectos anteriores para los componentes de estructuración a nivel arquitectónico.

Una cuestión estructural clave para casi todas las aplicaciones es minimizar las dependencias entre los componentes, la creación de una arquitectura de acoplamiento flexible de un conjunto de componentes altamente cohesivos. Existe una dependencia entre los componentes cuando un cambio en una potencial fuerza un cambio en los demás. Mediante la eliminación de dependencias innecesarias, los cambios son localizada y no se propagan a lo largo de una arquitectura (ver Fig. 1.1).

---

[3] Componente aquí y en el resto de este libro se utiliza de manera muy informal para significar un "trozo" reconocible de software, y no en el sentido más estricto de la de fi nición de Szyperski C. (1998) Componente del programa: Más allá de programación orientada a objetos, Addison-Wesley

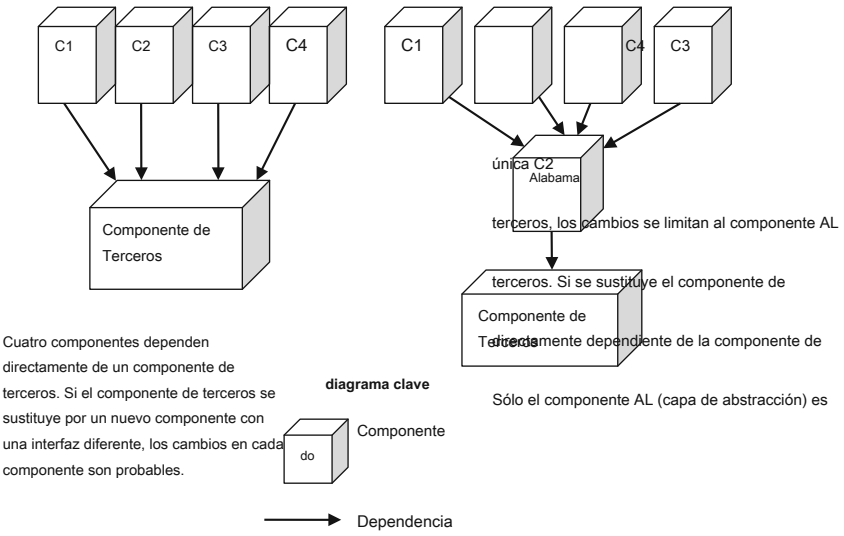


Fig. 1.1 Dos ejemplos de dependencias de los componentes

dependencias excesivas son simplemente una mala cosa. Ellos hacen que sea difícil de realizar cambios en los sistemas, más caro para probar los cambios, aumentan los tiempos de construcción, y hacen, desarrollo simultáneo basado en el equipo más difícil.

patrones ganaron". Por tanto, este libro va a utilizar patrones en lugar de estilos! 4

Comunicación de componentes 1.2.2 Arquitectura especí fi ca

Cuando una aplicación se divide en un conjunto de componentes, se hace necesario pensar en cómo estos componentes se comunican los datos e información de control. Los componentes de una aplicación pueden existir en el mismo espacio de direcciones, y comunicarse a través de llamadas a métodos sencillos. Ellos pueden ejecutar en diferentes hilos o procesos, y comunicarse a través de mecanismos de sincronización. O múltiples componentes pueden necesitar ser informada simultáneamente cuando se produce un evento en el entorno de la aplicación. Hay muchas posibilidades.

Un cuerpo de trabajo conocido colectivamente como patrones o estilos arquitectónicos [4] ha catalogado una serie de estructuras que faciliten utilizado con éxito ciertos tipos de comunicación de componentes [ver Patrones en Lectura adicional]. Estos patrones son esencialmente planos arquitectónicos reutilizables que describen la estructura y la interacción entre las colecciones de los componentes participantes.

Cada patrón tiene características que lo hacen apropiado para utilizar en tipos particulares satisfactorios de requisitos bien conocido. Por ejemplo, el patrón de cliente-servidor

[4] Patrones y estilos son esencialmente la misma cosa, sino como un líder autor arquitectura de software me dijo recientemente, "la gente

tiene varias características útiles, tales como las comunicaciones síncronas de petición-respuesta del cliente al servidor y servidores que soportan uno o más clientes a través de una interfaz publicada. De manera opcional, los clientes pueden establecer sesiones con servidores, que pueden mantener el estado de sus clientes conectados. arquitecturas cliente-servidor también deben proporcionar un mecanismo para que los clientes puedan ubicar los servidores, controlar los errores, y, opcionalmente, proporcionan seguridad en el acceso al servidor. Todas estas cuestiones se abordan en el patrón de arquitectura cliente-servidor.

El poder de los patrones de arquitectura deriva de su utilidad y capacidad de transmitir la información de diseño. Los patrones se ha comprobado que funcionan. Si se usa apropiadamente en una arquitectura, a aprovechar los conocimientos de diseño existentes mediante el uso de patrones.

Los sistemas grandes tienden a utilizar múltiples patrones, combinados de manera que satisfagan los requisitos de la arquitectura. Cuando una arquitectura se basa en patrones, también se hace fácil para los miembros del equipo para entender un diseño, como el patrón infiere estructura de componentes, las comunicaciones y los mecanismos abstractos que deben ser proporcionados. Cuando alguien me dice que su sistema se basa en una arquitectura cliente-servidor de tres niveles, sé de inmediato una cantidad considerable de su diseño. Este es un muy poderoso mecanismo de comunicación de hecho.

1.3 Arquitectura ocupa de los requisitos no funcionales

Los requisitos no funcionales son los que no aparecen en los casos de uso. En lugar de de fi nir qué la aplicación no, que tienen que ver con cómo la aplicación proporciona la funcionalidad requerida.

Hay tres áreas distintas de requisitos no funcionales:

- **Las limitaciones técnicas: Estos serán familiares para todos. Éstos limitan el diseño**

  opciones mediante la especificación de ciertas tecnologías que debe utilizar la aplicación. “Sólo tenemos los desarrolladores de Java, por lo que hay que desarrollar en Java”. “La base de datos existente se ejecuta en Windows XP”. Estos son por lo general no negociable.

- **restricciones de actividad: Estas opciones de diseño demasiado fuerza, sino para los negocios no es así,**

  razones técnicas. Por ejemplo, “Con el fin de ampliar nuestra base de clientes potenciales, hay que interactuar con herramientas XYZ”. Otro ejemplo es “El surtidor de nuestro middleware ha elevado los precios prohibitivos, por lo que nos estamos moviendo a una versión de código abierto”. La mayoría de las veces, estos también son negociables.

- **Atributos de calidad: Estas de fi ne los requisitos de una aplicación en términos de scal-**

  capacidad, disponibilidad, facilidad de cambio, la portabilidad, la facilidad de uso, rendimiento y así sucesivamente. Atributos de calidad abordar cuestiones de interés para los usuarios de aplicaciones, así como otras partes interesadas, como el propio equipo de proyecto o el promotor del proyecto. Capítulo 3 se analizan los atributos de calidad con cierto detalle.

por lo tanto, una arquitectura de aplicación debe abordar explícitamente estos aspectos del diseño. Los arquitectos deben entender los requisitos funcionales, y crear una plataforma que apoya estas y al mismo tiempo satisface los requisitos no funcionales.

1.3.1 Arquitectura es una abstracción

Una de las descripciones más útiles, pero a menudo no existentes, desde el punto de vista arquitectónico es algo que se conoce coloquialmente como una Marketecture. Se trata de una página, por lo general la representación informal de la estructura y las interacciones del sistema. Se muestra los principales componentes y sus relaciones y tiene unas cuantas etiquetas bien escogidas y cuadros de texto que retratan las filosofías de diseño incorporados en la arquitectura. UN Marketecture es un excelente vehículo para facilitar la discusión por las partes interesadas durante el diseño, la construcción, la revisión, y por supuesto el proceso de venta. Es fácil de entender y explicar y sirve como punto de partida para un análisis más profundo.

Un cuidadosamente diseñado Marketecture es particularmente útil, ya que es una descripción abstracta del sistema. En realidad, cualquier descripción arquitectónica debe emplear la abstracción con el fin de ser comprensible por los miembros del equipo y los actores del proyecto. Esto significa que los detalles innecesarios son suprimidas o ignoradas con el fin de centrar la atención y el análisis de las cuestiones arquitectónicas más destacadas. Esto normalmente se realiza mediante la descripción de los componentes en la arquitectura como cajas negras, especificando sólo su propiedades visibles externamente. Por supuesto, la descripción de la estructura y el comportamiento del sistema como colecciones de comunicación abstracciones de caja negra es normal para los profesionales que utilizan técnicas de diseño orientado a objetos.

Uno de los mecanismos más poderosos para describir una arquitectura es descomposición jerárquica. Los componentes que aparecen en un nivel de descripción se descomponen con más detalle en el acompañamiento de la documentación de diseño. Como ejemplo, la Fig. 1.2 representa una muy simple jerarquía de dos niveles usando una notación informal, con dos de los componentes en el diagrama de nivel superior descompuesto aún más.

Diferentes niveles de descripción en la jerarquía tienden a ser de interés para los diferentes desarrolladores en un proyecto. En la Fig. 1.2 , Es probable que los tres componentes en la descripción de alto nivel estarán diseñados y construidos por diferentes equipos que trabajan en el

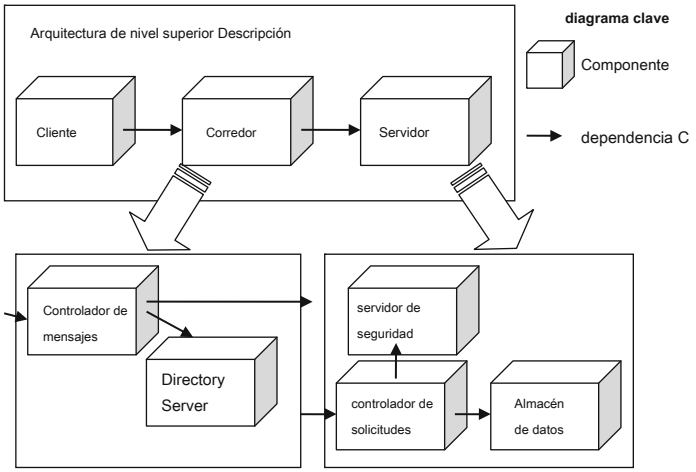


Fig. 1.2 Describiendo una arquitectura jerárquica 6

solicitud. La arquitectura particiones claramente las responsabilidades de cada equipo, de fi nir las dependencias entre ellos.

En este ejemplo hipotético, el arquitecto ha re fi ne el diseño de dos de los componentes, presumiblemente porque algunos requisitos no funcionales dictan que una mayor definición es necesario. Tal vez un servicio de seguridad existente debe ser utilizado, o la Corredor debe proporcionar una función de enrutamiento de mensajes fi específico que requiere un servicio de directorio que tiene un nivel conocido de la solicitud de rendimiento. En cualquier caso, este refinamiento fi más re crea una estructura que define y limita el diseño detallado de estos componentes.

La arquitectura simple en la Fig. 1.2 no se descompone la Cliente componente. Esto es, de nuevo, presumiblemente, debido a que la estructura interna y el comportamiento del cliente no es significativo en el logro de los requisitos no funcionales generales de la aplicación. Cómo Cliente obtiene la información que se envía a la Corredor no es un tema que preocupa al arquitecto, y en consecuencia el diseño detallado se deja abierta para el equipo de desarrollo del componente. Por supuesto, la Cliente componente podría ser el más complejo en la aplicación. Podría haber una arquitectura interna definida por su equipo de diseño, que cumpla con los objetivos especí fi cos de calidad para el Cliente componente. Estos son, sin embargo, preocupaciones localizadas. No es necesario que el arquitecto para complicar la arquitectura de la aplicación de estas cuestiones, ya que se pueden dejar con seguridad a la Cliente equipo de diseño para resolver. Este es un ejemplo de supresión de detalles innecesarios en la arquitectura.

1.3.2 Arquitectura Vistas

Una arquitectura de software representa un artefacto de diseño complejo. No es sorprendente entonces, como la mayoría de los artefactos complejos, hay una serie de formas de ver y entender la arquitectura. El término "arquitectura" puntos de vista se elevó a la prominencia en Philippe Krutchen de 1995 [5] documento sobre la 4 þ 1 Ver Modelo. Esto presenta una forma de describir y entender una arquitectura basada en los cuatro puntos de vista siguientes:

- Vista lógica: Esto describe los elementos signi fi cativos arquitectónico de la arquitectura y las relaciones entre ellos. La vista lógica capta esencialmente la estructura de la aplicación utilizando diagramas de clase o equivalentes.
- vista del proceso: Esta se centra en la descripción de la concurrencia y las comunicaciones elementos de una arquitectura. En las aplicaciones de TI, las principales preocupaciones están describiendo componentes multiproceso o replicados, y los mecanismos de comunicación síncrona o asíncrona utilizan.
- vista físico: Esto representa cómo los principales procesos y componentes son asignada en el hardware de las aplicaciones. Se podría mostrar, por ejemplo, cómo se distribuyen los servidores de bases de datos e Internet para una aplicación a través de una serie de máquinas de servidor.

[5] P.Krutchen, Blueprints-El valor arquitectónico "4 þ 1" Ver Modelo de arquitectura de software, IEEE Software, 12 (6) Nov.., 1995

- **vista de desarrollo: Esta captura la organización interna del software**

  componentes, por lo general, ya que se llevan a cabo en un entorno de desarrollo o herramienta de gestión con fi guración. Por ejemplo, la representación de una jerarquía de paquetes y la clase anidada para una aplicación Java representaría el punto de vista del desarrollo de una arquitectura.

Estos puntos de vista están unidas entre sí por los casos de uso fi cante arquitectónicamente significantes (a menudo llamados escenarios). Estos básicamente capturan los requisitos para la arquitectura y por lo tanto están relacionados con más de un punto de vista particular. Trabajando a través de los pasos de un caso de uso particular, la arquitectura se puede "probar", explicando cómo los elementos de diseño en la arquitectura responden al comportamiento requerido en el caso de uso. Vamos a estudiar cómo hacer esto "prueba de la arquitectura" en el Cap. 5.

Ya que el papel de Krutchen, ha habido mucho pensamiento, experiencia y desarrollo en el área de puntos de vista de la arquitectura. Sobre todo sobre todo es el trabajo de la SEI, coloquialmente conocido como los "Puntos de vista y más allá" enfoque (ver lecturas adicionales). Este recomienda capturar un modelo de arquitectura utilizando tres puntos de vista diferentes:

- **Módulo: Esta es una vista estructural de la arquitectura, que comprende el código**

  módulos tales como clases, paquetes y subsistemas en el diseño. También captura la descomposición módulo, herencia, asociaciones y agregaciones.
- **Componente y el conector: Este punto de vista se describen los aspectos del comportamiento de la**

  arquitectura. Los componentes son típicamente objetos, hilos, o procesos, y los conectores describen cómo interactúan los componentes. conectores comunes son los zócalos, middleware como CORBA o memoria compartida.
- **Asignación: Esta vista muestra cómo los procesos en la arquitectura se asignan a**

  hardware, y cómo se comunican usando redes y / o bases de datos. También captura una vista del código fuente en los sistemas de gestión de con fi guración, y que en el grupo de desarrollo tiene la responsabilidad de cada uno de los módulos.

La terminología utilizada en "Vistas y más allá" es fuertemente influenciada por la comunidad investigadora descripción de la arquitectura idioma (ADL). Esta comunidad ha sido influyente en el mundo de la arquitectura de software, pero ha tenido un impacto limitado en la tecnología de la información general. Así, mientras que este libro se concentrará en dos de estos puntos de vista, nos referiremos a ellos como el punto de vista estructural y la vista del comportamiento. perspicaces lectores deben ser capaces de trabajar a cabo el mapeo entre terminologías!

1.4 ¿Qué hace un arquitecto de software?

El ambiente que un arquitecto de software funciona en tiende para definir sus funciones y responsabilidades **exactas. Una buena descripción general del papel del arquitecto es mantenido por el SEI en su sitio web.** [6] **En lugar** de resumir esto, voy a describir brevemente, en ningún

[6] http://www.sei.cmu.edu/ata/arch_duties.html

en particular orden, cuatro habilidades esenciales para un arquitecto de software, independientemente de su entorno profesional.

- **Enlace: Arquitectos desempeñan muchas funciones de enlace. Han estado en contacto entre los clientes**
  o clientes de la aplicación y el equipo técnico, a menudo conjuntamente con los analistas de negocios y requerimientos. Trabajan de forma articulada entre los distintos equipos de ingeniería en un proyecto, ya que la arquitectura es fundamental para cada uno de éstos. Han estado en contacto con la gerencia, lo que justifica diseños, las decisiones y los costes. Han estado en contacto con la fuerza de ventas, para ayudar a promover un sistema a los compradores potenciales o inversores. Gran parte del tiempo, esta relación toma la forma de simplemente traducir y explicar terminología diferente entre los diferentes grupos de interés.

- **Ingeniería de software: Excelentes habilidades de diseño son las que consiguen un ingeniero de software**
  a la posición de arquitecto. Ellos son un requisito previo esencial para el papel. En términos más generales, sin embargo, los arquitectos deben promover las buenas prácticas de ingeniería de software. Sus diseños deben estar adecuadamente documentados y comunicados y sus planes deben ser explícitos y justificado. Ellos deben entender el impacto aguas abajo de sus decisiones, trabajando adecuadamente con los equipos de prueba de aplicaciones, documentación y liberación.

- **El conocimiento de tecnología: Los arquitectos tienen un profundo conocimiento de la tecnología**
  dominios que son relevantes para los tipos de aplicaciones que trabajan. Son influyentes en la evaluación y la elección de los componentes de terceros y tecnologías. Siguen los avances tecnológicos, y comprender cómo las nuevas normas, características y productos podrían ser explotados de manera útil en sus proyectos. Igual de importante, los buenos arquitectos saben lo que no saben, y pedir a los demás con mayor experiencia cuando necesitan información.

- **Gestión de riesgos: Los buenos arquitectos tienden a ser cautos. Ellos están constantemente**
  enumerar y evaluar los riesgos asociados con el diseño y la tecnología de decisiones que toman. Documentan y gestionar estos riesgos en conjunto con los patrocinadores y gestión de proyectos. Se desarrollan y abren las estrategias de mitigación de riesgos, que comunicarán a los equipos de ingeniería pertinentes. Ellos tratan de asegurarse de que no se produzcan desastres inesperados.

Busque estas habilidades en los arquitectos que trabaja con o emplea. Arquitectos juegan un papel central en el desarrollo de software, y deben ser polivalentes en software de ingeniería, la tecnología, la gestión y las comunicaciones.

1,5 arquitecturas y tecnologías

Los arquitectos deben tomar decisiones de diseño al principio del ciclo de vida de un proyecto. Muchos de éstos son difíciles, si no imposible, para validar y prueba hasta que las partes del sistema son en realidad construida. prototipado juiciosa de componentes arquitectónicos claves puede ayudar a aumentar la confianza en un enfoque de diseño, pero a veces es todavía difícil estar seguro del éxito de una opción de diseño particular, en un contexto de aplicación dada.

Debido a la dificultad de la validación de las decisiones de diseño inicial, arquitectos sensatez se basan en enfoques de eficacia probada para resolver ciertas clases de problemas. Este es uno de los grandes valores de los patrones arquitectónicos. Permiten a los arquitectos para reducir el riesgo mediante el aprovechamiento de diseños exitosos con atributos de ingeniería conocidos.

Los patrones son una representación abstracta de una arquitectura, en el sentido de que se pueden realizar en múltiples formas concretas. Por ejemplo, el patrón de publicación-suscripción arquitectura describe un mecanismo abstracto para débilmente acoplados, manyto-muchas comunicaciones entre los editores de mensajes y los abonados que deseen recibir mensajes. Sin embargo, no especifica cómo se gestionan las publicaciones y suscripciones, lo que la comunicación protocolos se utilizan, qué tipos de mensajes pueden ser enviados, y así sucesivamente. Estos son considerados todos los detalles de implementación.

Por desgracia, a pesar de las opiniones equivocadas de un número de académicos de informática, descripciones abstractas de las arquitecturas sin embargo, no se ejecutan en los ordenadores, ya sea directamente o por medio de la transformación rigurosa. Hasta que lo hagan, arquitecturas abstractas deben ser rei fi cado por los ingenieros de software como implementaciones de software concretos.

Afortunadamente, la industria del software ha llegado al rescate. patrones arquitectónicos utilizados ampliamente son compatibles con una variedad de marcos pre-construidos disponibles como las tecnologías comerciales y de código abierto. Por una cuestión de conveniencia, me referiré a ellos colectivamente como las tecnologías comerciales-off-the-shelf (COTS), a pesar de que estrictamente no es apropiado ya que muchos productos de código abierto de muy alta calidad se pueden utilizar libremente (a menudo con una pago por el apoyo modelo para la implementación de aplicaciones graves).

De todos modos, si un diseño requiere de publicación-suscripción de mensajería, o un intermediario de mensajes, o una arquitectura de tres capas, entonces las opciones de la tecnología disponible son muchas y variadas hecho. Este es un ejemplo de tecnologías de software que proporcionan las infraestructuras de software reutilizables, independientes de la aplicación que implementan demostrado enfoques arquitectónicos.

Como la fig. 1.3 representa, varias clases de tecnologías COTS se utilizan en la práctica para proporcionar implementaciones de patrones arquitectónicos envasados para su uso en sistemas de TI. Dentro de cada clase, existen productos que compiten comerciales y de código abierto. Aunque estos productos son superficialmente similares, tendrán diferentes conjuntos de características, ser implementado de manera diferente y tienen diferentes limitaciones en su uso.

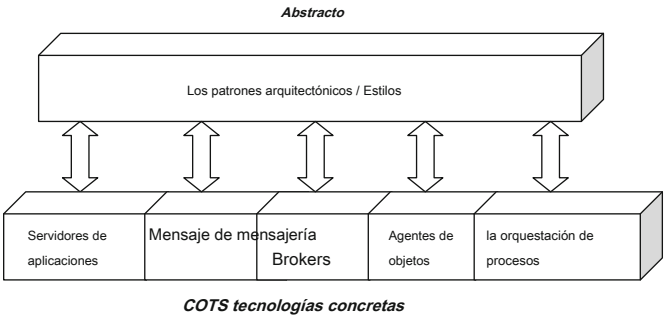


Fig. 1.3 Correspondencia entre los patrones arquitectónicos lógicas y tecnologías concretas 10

Los arquitectos están un poco a la vez bendecidos y maldecidos con esta diversidad de opciones de productos. La competencia entre los proveedores de productos impulsa la innovación, mejores conjuntos de características e implementaciones, y precios más bajos, sino que también representa una carga para el arquitecto para seleccionar un producto que tiene los atributos de calidad que satisfagan los requisitos de la aplicación. Todas las aplicaciones son diferentes en algunos aspectos, y no rara vez, o nunca, una una reducción de tamaño-ts-fi todos partido del producto. Diferentes implementaciones de tecnología COTS tienen diferentes conjuntos de puntos fuertes y débiles y los costes, y por lo tanto serán más adecuados para algunos tipos de aplicaciones que otros.

El DIF fi cultad para los arquitectos es en la comprensión de estos puntos fuertes y débiles al principio del ciclo de desarrollo de un proyecto, y la elección de un fi cación apropiada rei de los patrones arquitectónicos que necesitan. Por desgracia, esto no es una tarea fácil, y los riesgos y los costos asociados con la selección de una tecnología inadecuada son altos. La historia de la industria del software está llena de malas decisiones y proyectos fallidos posteriores. Para citar Eoin Woods, [7] y proporcionar otra extremadamente pragmática definición de la arquitectura de software:

> arquitectura de software es el conjunto de las decisiones de diseño que, si se hace de forma incorrecta, puede hacer que su proyecto sea cancelado.

Capítulos 4-6 proporcionan una descripción detallada y el análisis de estas tecnologías de infraestructura.

1.6 Arquitecto Título sopa

Escanear los anuncios puestos de trabajo. Verás principales arquitectos, arquitectos de productos, arquitectos técnicos, arquitectos de soluciones (Quiero colocar un anuncio parodia para un arquitecto problema), arquitectos de la empresa, y sin duda varios otros. He aquí un intento de dar algunas ideas generales en lo que significan:

- **Arquitecto en jefe:** Normalmente, un cargo superior y dirige un equipo de arquitectos dentro de una organización. Opera en un nivel general, muchas veces de organización, y coordina los esfuerzos a través de las líneas del sistema, aplicaciones y productos. Con mucha experiencia, con una rara combinación de un profundo conocimiento técnico y comercial.
- **Producto / Técnico / Arquitecto de soluciones:** Por lo general alguien que ha progresado a través de las filas técnicos y supervisa el diseño de la arquitectura de un sistema especí fi co o aplicación. Tienen un profundo conocimiento de cómo alguna pieza importante de software funciona realmente.
- **Arquitecto Empresarial:** Normalmente, una, más de rol de enfoque mucho menos técnica. arquitectos de la empresa utilizan diversos métodos de negocio y herramientas para comprender, documentar y planificar la estructura de los principales sistemas en una empresa.

El contenido de este libro es relevante para los dos primeros puntos anteriores, lo que requiere una sólida formación informática. Sin embargo, los arquitectos empresariales son algo

---

7 http://www.eoinwoods.info/

diferentes bestias. todo esto se vuelve muy confuso, especialmente cuando eres un arquitecto de software que trabaja en sistemas de la empresa.

En esencia, los arquitectos empresariales crear documentos, hojas de ruta, y los modelos que describen la organización lógica de las estrategias de negocio, métricas, capacidades de negocio, procesos de negocio, recursos de información, sistemas de negocio, y la infraestructura de redes dentro de la empresa.[8] Utilizan marcos para organizar todos estos documentos y modelos, siendo las más populares TOGAF[9] y el Marco Zachman.[10]

Ahora bien, si soy honesto, las anteriores capturas más o menos todo lo que sé acerca de la arquitectura de la empresa, a pesar de haber participado durante un corto tiempo en un esfuerzo arquitectura de la empresa! Soy un friki de corazón, y nunca he visto ninguna necesidad de la informática y el conocimiento de ingeniería de software en la arquitectura de la empresa. La mayoría de los arquitectos de la empresa que conozco tienen grados de sistemas comerciales o de información. Se refieren a la manera de “alinear la estrategia y la planificación con objetivos comerciales de la empresa”, “desarrollar políticas, normas y directrices para la selección de TI”, y “determinar la gobernabilidad”. Todas las preocupaciones muy altas e importantes, y no me refiero a ser despectivo, pero estos no son mis principales intereses. Las tareas de un arquitecto de la empresa, sin duda no se basan en unas pocas décadas de la informática acumulada y la teoría y la práctica de ingeniería de software.

Si eres curioso acerca de arquitectura de la empresa, hay algunas buenas referencias al final de este capítulo. Disfrutar.

1.7 Resumen

arquitectura de software es un bastante bien definida y comprendida disciplina de diseño. Sin embargo, sólo porque nosotros sabemos lo que es más o menos lo que hay que hacer, esto no quiere decir que sea mecánica o fácil. Diseño y evaluación de una arquitectura para un sistema complejo es un ejercicio creativo, que requiere conocimiento, la experiencia y la disciplina. Las di fi cultades se ven exacerbados por la naturaleza del ciclo de vida temprana de gran parte del trabajo de un arquitecto. En mi opinión, la siguiente cita de Philippe Krutchen resume el papel de un arquitecto a la perfección:

> La vida de un arquitecto de software es un largo (y, a veces dolorosa) sucesión de decisiones subóptimas realizado en parte en la oscuridad

El resto de este libro se describen los métodos y técnicas que pueden ayudar a arrojar al menos algo de luz sobre las decisiones de diseño arquitectónico. Gran parte de esta luz proviene de la comprensión y el aprovechamiento de los principios de diseño y tecnologías que han demostrado que funcionan en el pasado. Armado con este conocimiento, usted será capaz de

---

8 http://en.wikipedia.org/wiki/Enterprise_Architecture

9 http://www.opengroup.org/togaf/

10 http://www.zachmaninternational.com/index.php/the-zachman-framework

abordar los problemas complejos de arquitectura con más confianza, y después de un tiempo, tal vez incluso un poco garbo.

1.8 Lectura adicional

Hay un montón de buenos libros, informes y documentos disponibles en el mundo de la arquitectura de software. A continuación se presentan algunos que recomiendo especialmente. Estos se expanden en la información y los mensajes cubierto en este capítulo.

1.8.1 Arquitectura general

En términos de de fi nir el paisaje de la arquitectura de software y describir sus experiencias de proyectos, sobre todo con proyectos de defensa, es difícil ir más allá de los siguientes libros de los miembros del Instituto de Ingeniería de Software.

L. Bass, P. Clements, R Kazman. Arquitectura de software en la práctica, Segundo Edición. Addison-Wesley, 2003.

P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford. Arquitecturas de software de documentación: Vistas y más allá. 2 Dakota del Norte Edición, Addison-Wesley, 2010.

P. Clements, R. Kazman, M. Klein. La evaluación de arquitecturas de software: Métodos y casos prácticos. Addison-Wesley, 2002.

Para una descripción del “Estilo de descomposición”, véase Documentación de Arquitectura de Software, página 53. Y para una excelente discusión de la usos relación y sus consecuencias, ver el mismo libro, página 68.

Los siguientes son también merece la pena leer:

Nick Rozanski, Eion Woods, Software de Sistemas de Arquitectura: Trabajar con Stake- Uso de los titulares de puntos de vista y perspectivas, Addison-Wesley 2005 Richard N. Taylor, Nenad Medvidovic, Eric Dashofy, Arquitectura de software: Fundaciones, Teoría y Práctica, John Wiley and Sons, 2009

El artículo de Martin Fowler en el papel de un arquitecto es una lectura interesante.

Martin Fowler, ¿Quién necesita un arquitecto? IEEE Software, julio-agosto de 2003.

1.8.2 Requisitos de Arquitectura

El libro original que describe los casos de uso es:

I. Jacobson, M. Christerson, P. Jonsson, G. Overgaard. Software Orientado a Objetos Ingeniería: Un enfoque basado en casos de uso. Addison-Wesley, 1992.

diseño Responsabilidad impulsada es una técnica muy útil para la asignación de funcionalidad a componentes y subsistemas en una arquitectura. Lo siguiente debería ser de obligada lectura para los arquitectos.

R. Wirfs-Brock, A. McKean. Diseño del objeto: Roles, responsabilidades y colaboraciones. Addison-Wesley, 2002.

1.8.3 Patrones de Arquitectura

Hay una serie de libros de fi ne en los patrones de arquitectura. El trabajo de Buschmann es una excelente introducción.

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal ,. Patrón-Oriented Arquitectura de Software, Volumen 1: un sistema de modelos. John Wiley & Sons, 1996.

D. Schmidt, M. Stal, H. Rohnert, F. Buschmann. Patrón-Oriented Software arquitecture, Volumen 2, Patrones para concurrentes en red y objetos. John Wiley & Sons, 2000.

Dos libros recientes que se centran más en las pautas de sistemas de la empresa, especialmente integraciones de aplicaciones empresariales, están bien vale la pena leer.

M. Fowler. Los patrones de arquitectura de aplicación empresarial. Addison-Wesley, 2002.

G. Hohpe, B. Woolf. Los patrones de integración empresarial: diseñar, construir y desplegar soluciones de mensajería. Addison-Wesley, 2003.

1.8.4 Las comparaciones Tecnología

Una serie de documentos que surgió del proyecto de Evaluación de Tecnología Middleware (MTE) dan una buena introducción a los problemas y complejidades de las comparaciones de tecnología.

P. Tran, J. Gosper, I. Gorton. Evaluar el rendimiento sostenido de COTS-Sistemas de mensajería basados. en pruebas de software, la verificación y fiabilidad, vol 13, pp 229-240, Wiley and Sons, 2003.

I. Gorton, A. Liu. Evaluación del desempeño de los componentes alternativos Arquitecturas JavaBean de las aplicaciones empresariales, en IEEE Internet Computing, vol.7, no. 3, páginas 18-23, 2003.

A. Liu, I. Gorton. Acelerar COTS Middleware tecnología de adquisición: la Proceso i-mate. en IEEE Software, páginas 72-79, volumen 20, núm. 2, marzo / abril 2003.

1.8.5 Arquitectura Empresarial

En mi humilde opinión, hay algunos libros serio poco profundas escrito sobre la arquitectura empresarial. Sobreviví a través de las partes principales de este libro, por lo que lo recomendaría como un punto de partida.

James McGovern, Scott Ambler, Michael Stevens, James Linn, Elias Jo y Vikas Sharan, La Guía Práctica de la arquitectura de la empresa, Addison-Wesley, 2003.

Otra buena en general, libro práctico es:

Marc Lankhorst, Arquitectura Empresarial en el Trabajo, Springer-Verlag, 2009

Estoy seguro de que hay alegría que se tenía en el 700 þ páginas de la última TOGAF versión 9.0 libro (editorial Van Haren, ISBN: 9789087532307), pero al igual que Joyce Ulises, Sospecho que es una alegría que nunca voy a tener la paciencia para saborear. Si el Zachman Framework es más su cosa, hay un par de libros electrónicos, que se parecen a simple vista informativo:

http://www.zachmaninternational.com/index.php/ea-articles/25-editions

Capitulo 2
Al presentar el Estudio de Caso

2.1 Información general

En este capítulo se presenta el estudio de caso que se utilizará en los capítulos siguientes para ilustrar algunos de los principios de diseño en este libro. [1] Muy básicamente, la aplicación es un sistema de software multiusuario con una base de datos que se utiliza para compartir información entre los usuarios y herramientas inteligentes que tienen como objetivo ayudar al usuario a completar sus tareas de trabajo con mayor eficacia. Un diagrama de contexto informal se representa en la Fig. 2.1 .

El sistema cuenta con componentes de software que se ejecutan en la estación de trabajo de cada usuario, y un software distribuido compartida “back-end” que hace posible que las herramientas inteligentes de terceros para recopilar datos de, y comunicarse con, varios usuarios con el fin de ofrecer ayuda con su tarea . Es este software de back-end distribuida compartida que este caso de estudio se concentrará en, ya que es la zona donde surge la complejidad arquitectónica. También ilustra muchos de los problemas comunes de calidad que deben ser abordados por multiusuario, aplicaciones distribuidas.

2.2 El Sistema ICDE

La captura de información y difusión para el Medio Ambiente (ICDE) es parte de un conjunto de sistemas de software para la prestación de asistencia inteligente para profesionales como analistas fi nancieros, investigadores científicos y analistas de inteligencia. Con este fin, ICDE captura y almacena automáticamente los datos que registra una serie de acciones llevadas a cabo por un usuario al operar una estación de trabajo. Por ejemplo, cuando

[1] El proyecto de estudio de caso se basa en un sistema real en el que trabajé. Algunos licencia creativa ha sido explotada para simplificar los requisitos funcionales, por lo que estos no abrumar al lector con detalles innecesarios. Además, los eventos, detalles técnicos y el contexto descritas no siempre se ajustan a la realidad, ya que la realidad puede ser demasiado complicado para fines de ilustración.

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3_2, # Springer-Verlag Berlin Heidelberg 2011

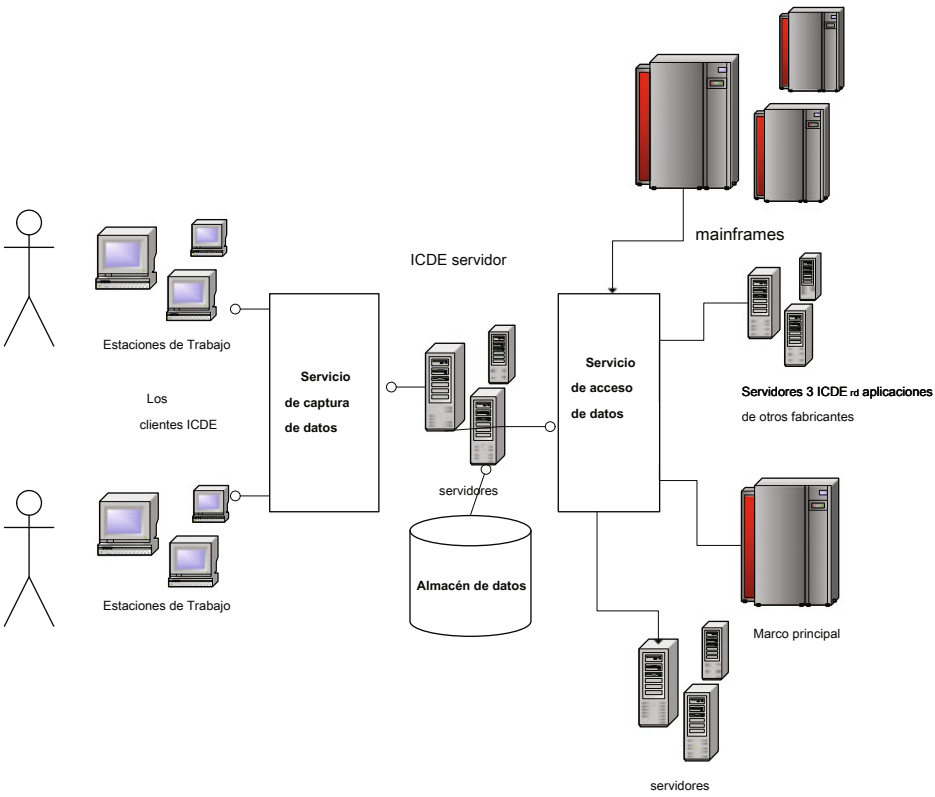


Fig. 2.1 diagrama de contexto ICDE 18

un usuario realiza una búsqueda en Google, el sistema ICDE transparentemente almacenar en una base de datos:

- La cadena de búsqueda consulta
- Las copias de las páginas web devueltos por Google que los usuarios se muestra en su navegador

Estos datos se pueden recuperar posteriormente de la base de datos del ICDE y usada por las herramientas de software de terceros que tratan de ofrecer ayuda inteligente para el usuario. Estas herramientas podrían interpretar una secuencia de entradas del usuario, y tratar de fi nd información adicional para ayudar al usuario con su tarea actual. Otras herramientas pueden rastrear los enlaces en los resultados de búsqueda devueltos que el usuario no hace clic en, tratando de encajar detalles nd potencialmente útiles que el usuario pasa por alto.

Un diagrama de casos de uso para el sistema ICDE se muestra en la Fig. 2.2 . Las tres principales casos de uso incorporar la captura de las acciones del usuario, la consulta de datos del almacén de datos, y la interacción de los terceros herramientas de otros fabricantes con el usuario.

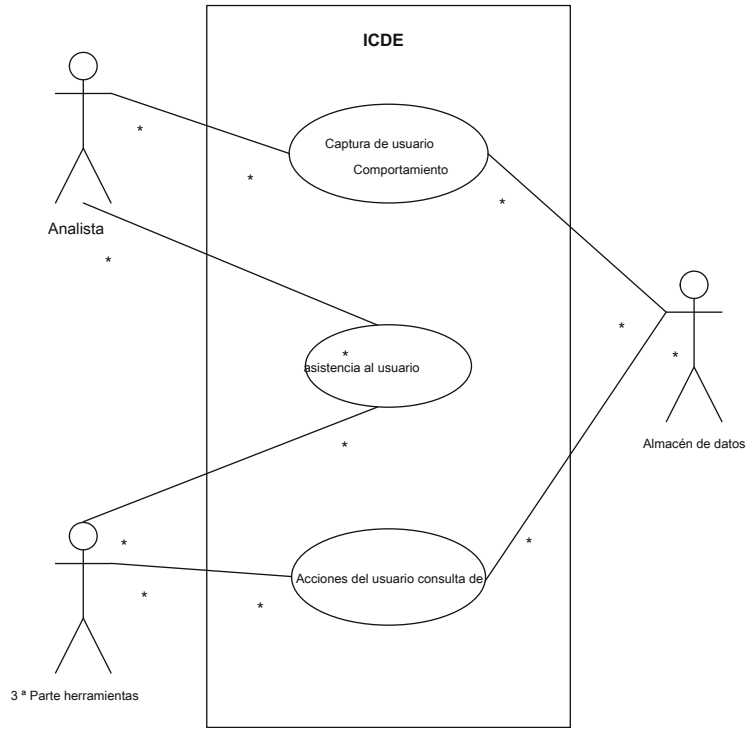


Fig. 2.2 casos de uso del sistema ICDE

Contexto 2.3 Proyecto

Pocos proyectos reales son los esfuerzos de campo efecto invernadero, lo que permite al equipo de diseño comienza con una hoja limpia y sobre todo sin restricciones de papel. El sistema ICDE ciertamente no es uno de ellos.

Una versión de producción inicial (v1.0) del ICDE se llevó a cabo por un pequeño equipo de desarrollo. Su principal objetivo era poner en práctica la Acciones de captura de usuario caso de uso. Esto creó el componente de cliente que se ejecuta en cada estación de trabajo del usuario, y condujo al diseño e implementación del almacén de datos. Esto era importante como almacén de datos era una parte integral del resto de la funcionalidad del sistema, y su diseño tenía que ser adecuada para soportar la alta tasa de transacciones que un gran número de usuarios podría llegar a generar.

v1.0 ICDE solamente se desplegó en un pequeño ensayo de usuario que involucra unos pocos usuarios. Este despliegue probado con éxito la funcionalidad del software cliente y demostrar los conceptos de captura y almacenamiento de datos. El diseño de v1.0 se basa en una arquitectura simple de dos niveles, con todos los componentes que se ejecutan en la estación de trabajo del usuario. Este diseño se muestra como un diagrama de componentes UML en la Fig. 2.3 . Los componentes de cliente de recolección y análisis fueron escritos en Java y tener acceso al almacén de datos

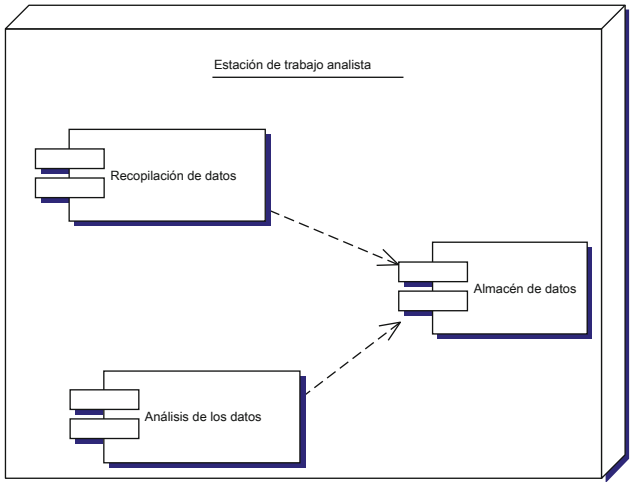


Fig. 2.3 ICDE versión de arquitectura 1,0 aplicación

(Servidor) directamente utilizando el JDBC [2] API. La aplicación ICDE completa ejecutado en Microsoft Windows XP.

El papel de cada componente es el siguiente:

- Recopilación de datos: El componente de recogida comprende una serie de vagamente

  procesos acoplados que se ejecutan en una estación de trabajo cliente que hacen un seguimiento de forma transparente las actividades relevantes del usuario y almacenarlos en el Almacén de datos. Los eventos capturados se relacionan con accesos a Internet, los documentos que se abren y buscan, las modificaciones realizadas a los documentos, y alguna información básica acerca de ventanas cuando el usuario abre y cierra las aplicaciones en el escritorio. Cada evento tiene numerosos atributos asociados a ella, dependiendo del tipo de evento. Por ejemplo, un doble clic del ratón tiene (X, y) coordinar atributos, y un evento de activación ventana tiene el nombre de la aplicación como un atributo asociado.

- Almacén de datos: Este componente comprende un comercial-off-the-shelf (COTS)

  base de datos relacional. La información de eventos almacena bases de datos relacionales en varias tablas para capturar las actividades de los usuarios, con marcas de tiempo añadido para que el orden de los acontecimientos puede ser reconstruida. Los objetos grandes, tales como imágenes de las páginas web y documentos binarios se almacenan como campos Binary Large Object (BLOB) que utilizan las instalaciones de bases de datos nativas.

- Análisis de los datos: Una interfaz gráfica de usuario (GUI) herramienta basada admite un conjunto de

  Las consultas sobre el almacén de datos. Esto era útil para propósitos de prueba, y para dar la herramienta creadores de terceros un vistazo inicial a los datos que se de ser capturado, y era por lo tanto disponible para ellos para el análisis.

[2] Java Database Connectivity. 20

2.4 Objetivos de negocio

v2.0 ICDE tenían objetivos mucho más ambiciosos. Después de haber comprobado que el sistema funcionó bien en pruebas de implementaciones, los promotores del proyecto tenía dos objetivos de negocio importantes para la próxima versión. Éstas eran:

- Animar a los terceros desarrolladores de herramientas de fiesta para escribir aplicaciones para el mantenimiento del ICDE.

  Por ejemplo, en finanzas, un desarrollador independiente puede construir un “asesor de la” que vigila las acciones que un analista está mirando en su navegador y les informa de cualquier evento en las noticias que podrían afectar el valor de las acciones.
- Promover el concepto ICDE y herramientas a los clientes potenciales, con el fin de mejorar

  su entorno de trabajo analítico.

Es evidente que estos dos objetivos se centran en el fomento de un negocio en crecimiento alrededor de la tecnología de la ICDE, mediante la creación de un mercado atractivo para herramientas de terceros y un entorno de asesoramiento avanzada para los usuarios en una variedad de dominios de aplicación. El logro de estos objetivos exige que los planes técnicos y comerciales detallados para ser elaborados y seguido a través. Desde un punto de vista puramente técnico, dejando de lado las actividades tales como ventas y comercialización, los siguientes objetivos principales Se identificaron - **véase la tabla 2.1 :**

Con el fin de atraer a los desarrolladores de herramientas de terceros, es esencial que el medio ambiente tiene una interfaz de programación de aplicaciones potente y fácil de usar (API) que se puede acceder desde cualquier plataforma de sistema operativo que un desarrollador decide utilizar. Esto daría a los desarrolladores de herramientas fl exibilidad en la elección de su plataforma de despliegue, y hacer la portabilidad herramientas existentes más simple. Las encuestas de las herramientas existentes también plantearon la cuestión de que las herramientas analíticas poderosas máquinas pueden requerir racimo de gama alta que se ejecuta. De ahí que necesitarían la capacidad de comunicarse con los despliegues ICDE a través de redes de área local (y eventualmente de ancho).

Otra encuesta de clientes ICDE probables mostró que las organizaciones de usuarios potenciales tenían grupos de 10-150 analistas. En consecuencia, era importante que el software podría ampliarse fácilmente para soportar dichos números. También debe haber ninguna característica de diseño inherentes que inhiben la tecnología de apoyo a los despliegues más grandes que pueden aparecer en el futuro.

Tabla 2.1 ICDE v2.0 objetivo objetivos de negocio

Negocio	Apoyando objetivo técnico
Anime herramienta de terceros desarrolladores	el acceso mediante programación simple y fiable para almacenar datos para terceros herramientas
	Heterogénea (es decir, no Windows) de soporte de plataforma para correr herramientas de terceros
	Permitir herramientas de terceros para comunicarse con los usuarios de un ICDE máquina remota
Promover el concepto del ICDE a los usuarios	Escalar los componentes de recogida de datos y almacenamiento de datos para soportar hasta 150 usuarios en un solo sitio
	implementación de bajo costo para cada usuario de estación de trabajo ICDE

Igualmente importante, para mantener el costo base de un despliegue tan bajo como sea posible, tecnologías COTS caros deben evitarse siempre que sea posible. Esto a su vez hará que el producto sea más atractivo en términos de precio para los clientes.

2.5 Limitaciones

Los objetivos técnicos eran ambiciosos, y requerirían una arquitectura diferente para apoyar el acceso y las comunicaciones de datos distribuidos. Por esta razón, se decidió concentrar los esfuerzos en esta nueva arquitectura, y dejar el cliente, incluyendo las herramientas de interfaz gráfica de usuario y la captura de datos y estable. Los cambios sólo se harían al cliente para que pueda comunicarse con la nueva arquitectura de gestión de datos y noti fi cación que este proyecto sería diseñar. Por esta razón, el diseño del lado del cliente no se trata en este caso de estudio.

Un horizonte de tiempo de 12 meses se fijó para v2.0 ICDE. Una libertad provisional después de 6 meses fue para la adopción y la elaboración de tecnologías para hacer la API, y creación de datos sintéticos para 22 al mismo tiempo que estaba siendo v2.0 ICDE productizado y mejorado.

Así como tener un horario fi jo, el presupuesto de desarrollo fue también fijo. Esto significa que los recursos disponibles para el desarrollo limitaría las características que podrían ser incluidos en la versión v2.0. Estas limitaciones presupuestarias también influido las posibles opciones de implementación, dado que el número de desarrolladores, sus habilidades y tiempo disponible esencialmente se fija.

objetivo proporcionar los conocimientos básicos necesarios en el diseño de arquitecturas para cumplir con los

2.6 Resumen

La aplicación ICDE hace un interesante caso de estudio para una arquitectura de software. Requiere la arquitectura de una aplicación existente para ser extendido y mejorado para crear una plataforma para nuevas funciones y capacidades. Las limitaciones de tiempo y presupuesto restringen las posibles opciones. Sin duda, una remodelación de la tienda ICDE cliente v1.0 y los datos existentes es completamente fuera de la cuestión.

En el Cap. 9, el diseño para el back-end ICDE se elaborará y se explican. Los siguientes capítulos tienen como

Capítulo 3
Atributos de Calidad de Software

3.1 Atributos de Calidad

Gran parte de la vida de una arquitectura de software se dedica a diseñar sistemas de software para cumplir un conjunto de requisitos de atributos de calidad. atributos de calidad de software en general incluyen la escalabilidad, seguridad, rendimiento y fiabilidad. Estos a menudo se llaman informalmente "-ilities" de una aplicación (aunque, por supuesto, algunos, como el rendimiento, no bastante fi esta especificación léxica de cationes).

requisitos de atributos de calidad son parte de los requisitos no funcionales de una aplicación, que capturan las múltiples facetas de cómo se consiguen los requisitos funcionales de una aplicación. Todo pero la aplicación más trivial tendrá requisitos no funcionales que se pueden expresar en términos de requisitos de atributos de calidad.

Para que tenga sentido, requisitos de atributos de calidad debe ser específico acerca de cómo una aplicación debe alcanzar una determinada necesidad. Un problema común que encuentro regularmente en los documentos de arquitectura es una declaración general como "La aplicación debe ser escalable".

Esto es demasiado imprecisa y realmente no sirve de mucho a nadie. Como se discute más adelante en este capítulo, los requisitos de escalabilidad son muchos y variados, y cada uno se relaciona con diferentes características de aplicación. Por lo tanto, debe esta escala hipotética aplicación para manejar el aumento de conexiones de usuario simultáneas? O el aumento de los volúmenes de datos? O la implementación de una base de usuarios más grande? O la totalidad de estos?

De fi nir cuál de estas medidas de escalabilidad debe ser soportado por el sistema es crucial desde un punto de vista arquitectónico, como soluciones para cada difieren. Por lo tanto es de vital importancia para de la calidad del hormigón definir atributos de requisitos, tales como:

> Debe ser posible escalar el despliegue de una cantidad inicial de 100 escritorios de los usuarios dispersos geográficamente a 10.000 sin un aumento en el esfuerzo / costo de instalación y con fi guración.

Este es precisa y significativa. Como arquitecto, esto me señala un camino a un conjunto de soluciones y tecnologías concretas que facilitan la instalación cero esfuerzo y despliegue.

Tenga en cuenta sin embargo, que muchos de los atributos de calidad son en realidad un poco difícil para validar y probar. En este ejemplo, sería poco probable que en las pruebas para la primera

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3_3, # Springer-Verlag Berlin Heidelberg 2011

liberar, un caso de prueba sería instalar y con fi gura la aplicación en 10.000 ordenadores de sobremesa. No puedo ver a un jefe de proyecto firmando en esa prueba alguna manera.

Aquí es donde el sentido común y la experiencia vienen en. La solución adoptada, obviamente, debe funcionar para el despliegue de 100 usuarios inicial. Sobre la base de los mecanismos exactos utilizados en la solución (tal vez de descarga de Internet, software de gestión de escritorio corporativo, etc), podemos entonces sólo analizarlo a lo mejor de nuestra capacidad para evaluar si el requisito de escalabilidad de concreto se pueden cumplir. Si no hay AWS fl obvias o problemas, es probablemente seguro asumir la solución se escala. Pero va a escalar a 10.000? Como siempre con el software, sólo hay una manera de estar absolutamente, 100% seguro, ya que "es toda la charla hasta las pistas de código". [1]

Hay muchos atributos de calidad en general, y la descripción de todos en detalle por sí sola podría llenar un libro o dos. Lo que sigue es una descripción de algunos de la cualidad más relevante atributos para aplicaciones generales de TI, y un poco de debate sobre los mecanismos de arquitectura que son ampliamente utilizados para proporcionar soluciones para los atributos de calidad exigidos. Estos le darán un buen lugar para empezar cuando se piensa en las cualidades de una aplicación que se está trabajando debe poseer.

3.2 Rendimiento

Aunque para muchas aplicaciones, el rendimiento no es un problema muy grande, que obtiene la mayor parte de la atención en la comunidad atributo de calidad lleno de gente. Sospecho que esto se debe a que es una de las cualidades de una aplicación que a menudo pueden ser fácilmente cuanti fi car y validado. Cualquiera que sea la razón, cuando las cuestiones de rendimiento, De Verdad sí importa. Las aplicaciones que funcionan mal en algún aspecto crítico de su comportamiento son candidatos probables para convertirse en animales atropellados en la carretera de ingeniería de software.

Un requisito de la calidad del desempeño de fi ne una medida que nos indica la cantidad de trabajo de una aplicación debe realizar en un tiempo determinado, y / o plazos que se deben cumplir para un correcto funcionamiento. Pocas aplicaciones de TI tienen duro en tiempo real restricciones como las que se encuentran en aviónica o robótica sistemas, en la que si alguna salida se produce un milisegundo o tres demasiado tarde, las cosas realmente desagradables e indeseables puede suceder (voy a dejar que el lector utilice su imaginación aquí). Pero las aplicaciones que necesitan para procesar cientos, a veces miles y decenas de miles de transacciones por segundo se encuentran en muchas de las grandes organizaciones, especialmente en el mundo de la finanzas, las telecomunicaciones y el gobierno.

El rendimiento por lo general se manifiesta en las siguientes medidas.

3.2.1 rendimiento

El rendimiento es una medida de la cantidad de trabajo que debe realizar una solicitud por unidad de tiempo. El trabajo se mide típicamente en transacciones por segundo (TPS), o mensajes

---

[1] Ward Cunningham a su fi nido! 24

procesadas por segundo (mps). Por ejemplo, una aplicación de banca en línea podría tener para garantizar que puede ejecutar 1.000 TPS de clientes de banca por Internet. Un sistema de gestión de inventario de un gran almacén que tenga que procesar 50 mensajes por segundo de los socios comerciales que solicitan órdenes.

Es importante entender precisamente lo que se quiere decir con un requisito de caudal. Es que el rendimiento promedio durante un período determinado de tiempo (por ejemplo, un día hábil), o pico de rendimiento? Esta es una distinción crucial.

Un claro ejemplo de esto es una aplicación para hacer sus apuestas sobre acontecimientos tales como carreras de caballos. Para la mayoría de las veces, una aplicación de esta calaña hace muy poco trabajo (personas en su mayoría hacen sus apuestas justo antes de una carrera), y por lo tanto tiene un requisito de caudal promedio bajo y fácilmente alcanzable. Sin embargo, cada vez que hay un evento de carreras, tal vez todas las noches, el período más o menos 5 minutos antes de cada carrera ve miles de apuestas están colocados cada segundo. Si la aplicación no es capaz de procesar estas apuestas, ya que se colocan, entonces el negocio pierde dinero, y los usuarios se vuelven muy descontentos (y negando los jugadores la oportunidad de perder dinero no es una buena cosa para cualquier persona). Por lo tanto, para este escenario, la aplicación **debe estar diseñado para satisfacer anticipada pico rendimiento, no son promedio. De hecho, soporta solamente el** rendimiento promedio es probable que sea un “cambio en las carreras” error de diseño para un arquitecto.

3.2.2 Tiempo de respuesta

Esta es una medida de la latencia de una aplicación exposiciones en el procesamiento de una transacción de negocios. El tiempo de respuesta es más a menudo (pero no exclusivamente) asociado a la vez que una aplicación se necesita para responder a alguna entrada. Un tiempo de respuesta rápida permite a los usuarios trabajar con mayor eficacia, y por lo tanto es bueno para los negocios. Un excelente ejemplo es una aplicación de punto de venta que soporta un gran almacén. Cuando un elemento se escanea en la caja, una forma rápida, segundo o menos la respuesta del sistema con el precio del artículo, el cliente que puede ser servido rápidamente. Esto hace que el cliente y la tienda feliz, y eso es una buena cosa para todos los actores involucrados.

Una vez más, a menudo es importante distinguir entre los tiempos de respuesta garantizados y media. Algunas **aplicaciones pueden necesitar todas solicitudes de ser atendidas dentro de un límite de tiempo específico ed. Este es un** tiempo de respuesta garantizado. Otros pueden especificar un tiempo medio de respuesta, lo que permite latencias más grandes cuando la aplicación está muy ocupado. Es también muy extendida en el último caso para un requisito de tiempo de respuesta límite superior para ser especi fi. Por ejemplo, el 95% de todas las solicitudes debe ser procesado en menos de 4 s, y no hay peticiones debe tener más de 15 s.

3.2.3 Plazos

Seguramente todos hemos oído hablar del sistema de predicción del tiempo que tomó 36 horas para producir el pronóstico para el día siguiente! No estoy seguro si esto es apócrifa, pero es un excelente ejemplo de la necesidad de cumplir con un plazo de rendimiento. en los plazos

el mundo de TI se asocia comúnmente con los sistemas por lotes. Un sistema de pago de seguridad social debe completar en el tiempo para depositar los pagos por el demandante en sus cuentas en un día determinado. Si fi acabados finales, los solicitantes no se les paga cuando esperan, y esto puede causar trastornos y dolor severo, y no sólo para los reclamantes. En general, cualquier aplicación que tiene una ventana de tiempo limitada para completar tendrá un requisito plazo el rendimiento.

Estos tres atributos de rendimiento de todo pueden ser claramente especi fi y validado. Sin embargo, hay un error común a evitar. Se encuentra en la definición de una transacción, petición o el mensaje, todas las cuales se utilizó deliberadamente muy imprecisa en lo anterior. En esencia se trata de la definición de la carga de trabajo de una aplicación. La cantidad de procesamiento requerida para una transacción de negocios dada es una aplicación específica
medida. Incluso dentro de una aplicación, es probable que haya muchos tipos diferentes de solicitudes o transacciones, variando quizás de base de datos rápida las operaciones de lectura, a las actualizaciones complejas a múltiples bases de datos distribuidas.

Simplemente, no existe una medida genérica carga de trabajo, que depende enteramente de lo que el trabajo está haciendo la aplicación. Por lo tanto, al coincidir para cumplir con una medida de rendimiento dado, ser preciso sobre la carga de trabajo exacta o mezcla transacción, de fi nido en términos applicationspeci fi cas, que está firmando.

3.2.4 Rendimiento del Sistema de ICDE

3.2 Ejemplo: Rendimiento del Sistema de ICDE retrasado la interfaz gráfica de usuario 26

El rendimiento en el sistema ICDE es un atributo de calidad importante. Uno de los requisitos clave de rendimiento se refiere a la naturaleza interactiva del ICDE. Cuando los usuarios realizan sus tareas de trabajo, la parte cliente de la aplicación ICDE clave trampas y las acciones del ratón y los envía al servidor ICDE para su almacenamiento. En consecuencia, es muy importante que los usuarios ICDE no experimentan ningún retraso en el uso de sus aplicaciones mientras que el software ICDE atrapa y almacena eventos.

Atrapando usuario y la aplicación generada eventos en la interfaz gráfica de usuario se basa en el aprovechamiento de la interfaz de programación de aplicaciones del sistema fi co-plataforma específica (API). Las API proporcionan ganchos en los mecanismos de control de eventos interfaz gráfica de usuario y el sistema operativo subyacente. La implementación de esta funcionalidad es una preocupación aplicación cliente ICDE, y por lo tanto es responsabilidad del equipo cliente ICDE para asegurar que esto se lleva a cabo como fi ciente y rápido como sea posible.

Una vez que se atrapa un evento, el cliente ICDE debe llamar al servidor para almacenar el evento en el almacén de datos. Es vital, por tanto, que esta operación no contribuye cualquier retraso que el usuario pueda experimentar. Por esta razón, cuando se detecta un evento, se escribe en una cola en memoria en el cliente ICDE. Una vez que el evento se almacena en la cola, el hilo de la detección de eventos devuelve inmediatamente y espera para capturar el próximo evento. Esta es una operación muy rápida y por lo tanto presenta un retraso significativo. Otro hilo conductor en el fondo tira constantemente eventos de la cola y llama al servidor ICDE para almacenar los datos.

Esta solución dentro del cliente ICDE desacopla la captura y almacenamiento de eventos. Un retraso de escritura al

código. Desde la perspectiva del servidor ICDE, esto es crucial. El servidor debe por supuesto estar diseñado para almacenar eventos en el almacén de datos lo más rápido posible. Sin embargo, el diseño de servidor se puede garantizar que sólo habrá alguna vez una petición de un cliente por estación de trabajo de usuario en vuelo FL en cualquier instante, ya que sólo hay un hilo en cada cliente que envía el flujo de eventos de usuario en el servidor.

Así que para el servidor ICDE, sus requisitos de rendimiento clave eran fáciles de especificar. Debe proporcionar por debajo del segundo promedio de los tiempos de respuesta a las solicitudes del cliente ICDE.

3.3 Escalabilidad

Vamos a empezar con un representante de fi nición de escalabilidad [2]:

La recuperación de una solución a un problema funcionará cuando el tamaño del problema aumenta.

Esto es útil en un contexto arquitectónico. Nos dice que la escalabilidad es acerca de cómo un diseño puede hacer frente a algunos aspectos de los requisitos de la aplicación cada vez mayores en tamaño. Para llegar a ser un requisito atributo de calidad del hormigón, necesitamos entender exactamente lo que se espera conseguir más grande. Aquí hay unos ejemplos:

3.3.1 Solicitud de carga

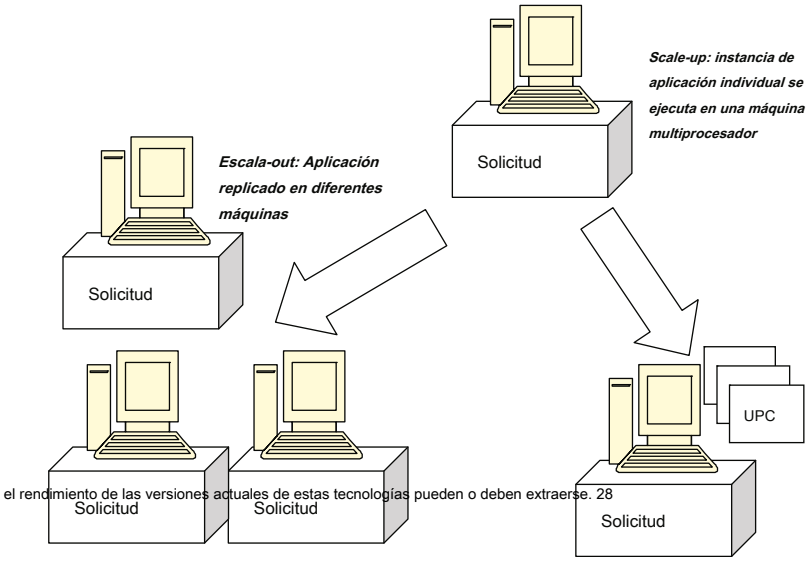
Basado en cierta mezcla Ned de fi de peticiones en una plataforma de hardware dada, una arquitectura para una aplicación de servidor puede estar diseñado para soportar 100 tps a carga pico, con una media de 1 s tiempo de respuesta. Si esta carga de solicitudes fueron creciendo en diez veces, puede el soporte de la arquitectura de este aumento de la carga?

En el mundo perfecto y sin capacidad de hardware adicional, como la carga aumenta, el rendimiento de aplicación debe permanecer constante (es decir, 100 tps), y tiempo de respuesta por la petición debe aumentar solamente linealmente (es decir, 10 s). Una solución escalable luego permitir la capacidad de procesamiento adicional para ser desplegado para aumentar el rendimiento y disminuir el tiempo de respuesta. Esta capacidad **adicional puede ser desplegado de dos maneras diferentes, uno mediante la adición de más CPU [3] (y la memoria** probable) a la máquina de las aplicaciones se ejecuta en (escalar), el otro de la distribución de la aplicación en **varios equipos (escalar). Esto se ilustra en la fig. 3.1 . Escala hasta funciona bien si es multiproceso una** aplicación, o múltiples instancias de proceso roscados individuales puede ejecutarse juntos en la misma máquina. Este último, por supuesto, consumir memoria adicional y recursos asociados, como son los procesos de vehículos pesados, hambrientos de recursos para lograr la concurrencia.

---

2 De http://www.hyperdictionary.com

3 La adición de una CPU más rápida nunca es una mala idea tampoco. Esto es especialmente cierto si una aplicación tiene componentes o cálculos que son inherentemente de un solo subproceso.



el rendimiento de las versiones actuales de estas tecnologías pueden o deben extraerse. 28

Fig. 3.1 Escalar frente ampliar

Escalar funciona bien si hay poco o, idealmente, ningún trabajo adicional necesario gestionar la distribución de las solicitudes entre las múltiples máquinas. El objetivo es mantener cada máquina igualmente ocupado, ya que la inversión en más de hardware se desperdicia si una máquina está totalmente cargado y otros ralentí de distancia. La distribución de carga de manera uniforme entre múltiples máquinas es conocida como el equilibrio de carga.

cedato, estos resultados son en la instantánea en el tiempo y son para propósitos ilustrativos. Absolutamente ninguna conclusión sobre

Es importante destacar que, para cualquiera de los enfoques, escalabilidad debe lograrse sin modificaciones a la arquitectura subyacente (aparte de los cambios fi guración inevitable Con Si se utilizan múltiples servidores). En realidad, como la carga aumenta, las aplicaciones exhibirán una disminución en el rendimiento y un aumento exponencial posterior en el tiempo de respuesta. Esto sucede por dos razones. En primer lugar, el aumento de las causas de ocupación aumentó la contención de recursos, tales como la CPU y la memoria por los procesos y los hilos en la arquitectura del servidor. En segundo lugar, cada solicitud consume algún recurso adicional (espacio de amortiguación, cerraduras, etc.) en la aplicación, y, finalmente, este recurso se agota y los límites de escalabilidad.

arquitecturas para aplicaciones empresariales JavaBean, en IEEE Internet Computing, vol.7, no. 3, páginas 18-23, 2003. Tenga en

Como ilustración, la fig. 3.2 muestra cómo seis versiones diferentes de la misma aplicación implementados utilizando diferentes servidores de aplicaciones JEE realizan a medida que aumenta la carga de 100 a 1.000 clientes. [4]

[4] El contexto completo para estas cifras se describe en: I.Gorton, A Liu, Evaluación del desempeño de los componentes alternativos

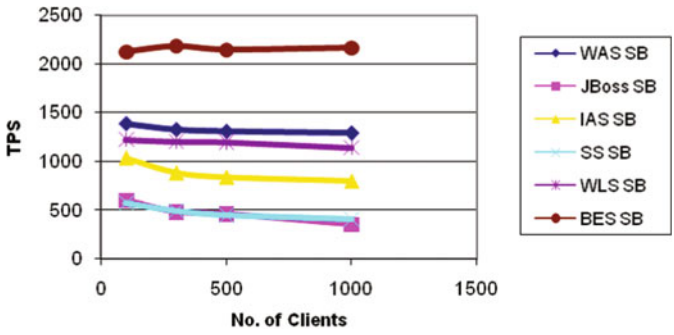


Fig. 3.2 Efectos del aumento de la carga petición del cliente en las plataformas JEE

3.3.2 conexiones simultáneas

Una arquitectura puede estar diseñado para soportar 1.000 usuarios concurrentes. ¿Cómo responde la arquitectura si este número crece de forma significativa? Si un usuario conectado consume algunos recursos, entonces es probable que haya un límite en el número de conexiones que puede ser apoyado de manera efectiva.

Me encontré con un ejemplo clásico de este problema al realizar una revisión de la arquitectura de un proveedor de servicios de Internet (ISP). Cada vez que un usuario conectado al servicio, la aplicación del ISP dio lugar a un nuevo proceso en su servidor que era responsable de distribuir publicidad dirigida al usuario. Esto funcionó muy bien, pero cada proceso consume considerables recursos de memoria y de procesamiento, incluso cuando el usuario conecta de forma sencilla y no hizo nada. Las pruebas revelaron rápidamente que las máquinas de servidor del ISP sólo podían apoyar alrededor de 2.000 conexiones antes de su memoria virtual se agotó y las máquinas de molido de manera efectiva a un alto en un frenesí de basura de disco. Esto hizo escalar las operaciones del ISP para apoyar a 100.000 usuarios de una proposición prohibitivamente caro, y, finalmente, a pesar de los esfuerzos desesperados de rediseño,

3.3.3 Tamaño de Datos

En pocas palabras, ¿cómo se comporta una aplicación como los datos que procesa los aumentos en el tamaño? Por ejemplo, una aplicación de intermediario de mensajes, tal vez una sala de chat, puede estar diseñada para procesar los mensajes de un tamaño medio esperado. ¿Qué tan bien reaccionará la arquitectura si el tamaño de los mensajes crece de forma significativa? En un tono ligeramente diferente, una solución de gestión de la información puede estar diseñada para buscar y recuperar datos de un repositorio de un tamaño fi cado. ¿Cómo se comportará la aplicación si el tamaño del repositorio crece, en términos de tamaño prima y / o el número de artículos? El último

se está convirtiendo en un problema tal que ha dado lugar a toda un área de investigación y desarrollo conocido como computación intensiva de datos. [5]

3.3.4 despliegue

¿De qué manera el esfuerzo involucrado en el despliegue o la modificación de una aplicación a una base creciente de usuarios crezca? Esto incluiría esfuerzo para su distribución, con fi guración y puesta al día con las nuevas versiones. Una solución ideal sería proporcionar mecanismos automatizados que pueden desplegarse de forma dinámica y gurar con fi una aplicación a un nuevo usuario, la captura de la información de registro en el proceso. Esta es, de hecho, exactamente cuántas aplicaciones se distribuye hoy en Internet.

3.3.5 Algunas reflexiones sobre Escalabilidad

El diseño de arquitecturas escalables no es fácil. En muchos casos, la necesidad de escalabilidad temprano en el diseño no es aparente y no se especi fi ca como parte de los requisitos de atributos de calidad. Se necesita un arquitecto inteligente y atento para garantizar enfoques inherentemente no escalable no se introducen como componentes arquitectónicos básicos. Incluso si la escalabilidad es un atributo de calidad requerida, validando que es fi satisfecho por una solución propuesta a menudo simplemente no es práctico en términos de programación o el costo. Es por eso que es importante para un arquitecto que se basan en diseños y tecnologías siempre que sea práctico de eficacia probada.

3.3.6 Escalabilidad para la aplicación ICDE

El principal requisito para la escalabilidad del sistema ICDE es apoyar el número de usuarios previstos en el mayor despliegue ICDE anticipada. Los requisitos especifican esto como aproximadamente 150 usuarios. Por consiguiente, la aplicación de servidor ICDE debe ser capaz de manejar una carga pico de 150 solicitudes simultáneas de los clientes ICDE.

3.4 capacidad fi Modi

Todos los arquitectos de software capaces saben que, junto con la muerte y los impuestos, modificaciones a un sistema de software durante su vida útil son simplemente un hecho de la vida. Es por eso que teniendo en cuenta posibles cambios a la aplicación es una buena práctica durante

---

[5] Una buena descripción de los computación intensiva de datos y algunos enfoques interesantes es la edición especial de la IEEE Computer partir de abril de 2008 - http://www2.computer.org/portal/web/csdl/ revistas / equipo # 3

formulación arquitectura. Cuanto más flexibilidad que puede ser integrado en un diseño inicial, a continuación, los cambios subsiguientes menos doloroso y costoso será. Esa es la teoría de todos modos.

El atributo de calidad capacidad modificado es una medida de lo fácil que puede ser para cambiar una aplicación para atender a los nuevos requerimientos funcionales y no funcionales. Observe el uso de “may” en la frase anterior. La predicción de la capacidad modificado requiere una estimar de esfuerzo y / o el costo de hacer un cambio. Sólo se sabe con certeza lo que un cambio va a costar después de que se ha hecho. A continuación, encontramos lo bueno que era su estimación.

Modi fi mide la capacidad sólo son relevantes en el contexto de una solución arquitectónica dada. Esta solución debe expresarse al menos estructuralmente como una colección de componentes, las relaciones de los componentes y una descripción de cómo los componentes interactúan con el medio ambiente. A continuación, evaluar la capacidad modificada requiere el arquitecto para hacer valer los escenarios de cambio probables que captan cómo pueden evolucionar los requisitos. A veces, estos serán conocidos con un grado razonable de certeza. De hecho, los cambios pueden incluso ser especificados en el plan de proyecto para las versiones posteriores. Gran parte del tiempo, sin embargo, tendrá que ser solicitada a las partes interesadas de aplicación, y extraídos de la experiencia del arquitecto posibles modificaciones. Hay infinitamente de un elemento de una bola de cristal involucrados.

escenarios de cambio ilustrativos son:

- Proporcionar acceso a la aplicación a través rewalls fi, además de existente
  “Detrás del cortafuego” de acceso.
- Incorporar nuevas características de auto check-in check-out.
- El proveedor de software de reconocimiento del habla COTS se retira del negocio y necesitamos
  para reemplazar este componente.
- La aplicación tiene que ser portado a Linux para el Microsoft Windows
  plataforma.

Para cada escenario de cambio, el impacto del cambio previsto en la arquitectura se puede evaluar. Este impacto no suele ser fácil de cuantificar, ya que más a menudo que no no existe la solución que se está evaluando. En muchos casos, lo mejor que se puede lograr es un análisis de impacto convincente de los componentes de la arquitectura que se necesita modi fi cación, o una demostración de cómo la solución puede adaptarse a la modi fi cación sin cambios.

Por último, sobre la base de costo, tamaño o del esfuerzo estimaciones de los componentes afectados, algunas útiles cuanti fi cación del coste de un cambio se puede realizar. Los cambios aislados a los componentes individuales o subsistemas débilmente acoplados son probable que sea menos caro de fabricar que los que causan efectos de la ondulación a través de la arquitectura. Si un cambio probable parece difícil y complejo de hacer, esto puede destacar una debilidad en la arquitectura que pudiera justificar un nuevo examen y rediseño.

Una palabra de precaución debe ser emitida aquí. Si bien débilmente acoplados, fácilmente modi fi arquitecturas capaces son generalmente “una buena cosa”, el diseño para la capacidad modificada necesita ser pensada cuidadosamente. Altamente arquitecturas modulares pueden llegar a ser excesivamente compleja, incurrirá en gastos adicionales de rendimiento y requieren significativamente más diseño y esfuerzo de construcción. Esto puede ser justificado en algunos sistemas que deben estar altamente con fi gurable tal vez en la implementación o en tiempo de ejecución, pero a menudo no lo es.

Usted probablemente ha escuchado algunos sistemas descritos como “sobre ingeniería”, lo que significa esencialmente invertir más esfuerzo en un sistema que se justifica. Esto se hace a menudo porque los arquitectos creen que saben las necesidades futuras de su sistema, y decide que lo mejor es hacer un diseño más flexible o sofisticados, por lo que puede adaptarse a las necesidades previstas. Eso suena razonable, pero requiere una bola de cristal fiable. Si las predicciones son erróneas, mucho tiempo y dinero se puede desperdiciar.

Hace poco estaba en el periférico de un proyecto de este tipo. La ventaja técnica pasó 5 meses establecimiento de una arquitectura basada en la mensajería cuidadosamente diseñado basado en el patrón de inyección de dependencia. [6] El objetivo era hacer que esta arquitectura extremadamente robusto y crear modelos de datos flexibles para la mensajería y el almacén de datos subyacente. Con estos en su lugar, la teoría era que la arquitectura podría ser reutilizado una y otra vez con el mínimo esfuerzo, y sería sencillo para inyectar nuevos componentes de procesamiento debido a la flexibilidad que ofrece la inyección de dependencias.

La palabra teoría en la frase anterior fue cuidadosamente elegido sin embargo. Los actores del sistema se impacientó, preguntándose por qué tanto esfuerzo estaba siendo gastado en una solución tan sofisticada, y pidieron ver algún progreso demostrable. La ventaja técnica resistió, insistiendo en que su equipo no debe ser desviada y continuó a abrazar a largo plazo beneficios de la arquitectura. Al igual que esta solución inicial fue a punto de finalizar, los participantes perdieron la paciencia y reemplazaron la ventaja técnica con alguien que estaba promoviendo una solución mucho más simple, servidor Web basado como su fi ciente.

Este fue un caso clásico de manipulación excesiva. Mientras que la solución original era elegante y podría haber cosechado grandes beneficios en el largo plazo, tales argumentos son esencialmente imposible ganar a menos que pueda mostrar evidencia demostrable, concreta de este a lo largo del camino. La adopción de enfoques ágiles es la clave del éxito aquí. Hubiera sido sensata para construir una versión inicial de la arquitectura de núcleo en unas pocas semanas y demostrar esto abordar una historia / usuario caso de uso que fue significativo para el actor. La manifestación habría implicado algunos elementos prototípicos de la arquitectura, no sería probado completamente, y sin duda se requiere un cierto código de usar y tirar para implementar el caso de uso - por desgracia todas las cosas desagradables a la dirección técnica. El éxito aunque hubiera construido confianza con las partes interesadas en la solución técnica,

La clave entonces es no dejar que la pureza de diseño conducir un diseño. Por el contrario, concentrándose en los requisitos conocidos y evolucionando y refactorización la arquitectura a través de iteraciones regulares, mientras que la producción de código que se ejecuta, tiene mucho sentido en casi todas las circunstancias. Como parte de este proceso, se puede analizar continuamente su diseño para ver qué mejoras futuras que pueda adaptarse (o no). Trabajando en estrecha colaboración con las partes interesadas puede ayudar a generar altamente probables necesidades futuras, y eliminar las que parece muy poco probable. Deja que estas impulsan la estrategia de arquitectura por todos los medios, pero nunca pierden de vista los requisitos conocidos y los resultados a corto plazo.

---

6 http://martinfowler.com/articles/injection.html

3.4.1 capacidad fi Modi para la aplicación ICDE

Modi fi capacidad para la aplicación ICDE es un culto di fi a uno especificar. Un requisito probable sería que la gama de eventos atrapados y almacenados por el cliente ICDE que ser ampliado. Esto tendría implicaciones en el diseño tanto del cliente ICDE y el almacén de servidor ICDE y datos.

Otra sería de herramientas de terceros para quieren comunicar nuevos tipos de mensajes. Esto tendría implicaciones en los mecanismos de intercambio de mensajes que el servidor ICDE apoyo. Por lo tanto estos dos escenarios capacidad modi fi podrían utilizarse para probar el diseño resultante para la facilidad de modi fi cación.

3.5 Seguridad

La seguridad es un tema técnico complejo que sólo puede ser entendido tanto superficialmente aquí. A nivel arquitectónico, la seguridad se reduce a la comprensión de los requisitos de seguridad precisas para una aplicación, y la elaboración de mecanismos para apoyarlos. Los requisitos más comunes relacionados con la seguridad son:

- **Autenticación: Las aplicaciones pueden verificar la identidad de sus usuarios y otros**
  aplicaciones con las que se comunican.
- **Autorización: usuarios y aplicaciones autenticados tienen de fi nida derechos de acceso**
  a los recursos del sistema. Por ejemplo, algunos usuarios pueden tener acceso de sólo lectura a los datos de la aplicación, mientras que otros tienen de lectura-escritura.
- **encriptación: Los mensajes enviados a / de la aplicación están cifrados.**
- **Integridad: Esto asegura que el contenido de un mensaje no se alteran durante el transporte.**
- **El no rechazo: El remitente de un mensaje tiene prueba de la entrega y el receptor**
  se asegura la identidad del remitente. Esto no significa ni puede refutar posteriormente su participación en el intercambio de mensajes.

Hay tecnologías bien conocidas y ampliamente usados que soportan estos elementos de seguridad de las aplicaciones. El Secure Socket Layer (SSL) y las infraestructuras de clave pública (PKI) se utilizan comúnmente en las aplicaciones de Internet para proporcionar autenticación, cifrado y no repudio. Autenticación y autorización se apoya en tecnologías Java utilizando el JAAS (JAAS). Los sistemas operativos y bases de datos proporcionan seguridad basada inicio de sesión para la autenticación y autorización.

Con suerte que está obteniendo la imagen. Hay muchas maneras, de hecho, a veces demasiados, para apoyar a los atributos de seguridad requerido para una aplicación. Bases de datos quieren imponer su modelo de seguridad en el mundo. diseñadores .NET felizmente aprovechar las características de seguridad operativos Windows. Las aplicaciones Java pueden aprovechar JAAS sin grandes problemas. Si una aplicación sólo necesita ejecutar en uno de estos dominios de seguridad, a continuación, las soluciones son fácilmente disponibles. Si una aplicación consta de varios componentes que todos deseamos para administrar la seguridad, las soluciones adecuadas deben estar

diseñado que normalmente se localizan gestión de la seguridad en un solo componente que aprovecha la tecnología más apropiada para satisfacer los requisitos.

3.5.1 Seguridad para la aplicación ICDE

Autentificación de usuarios ICDE y herramientas de terceras partes es ICDE los principales requisitos de seguridad para
que tarda en recuperarse antes de que se restablezca el servicio completo. 34
el sistema ICDE. En v1.0, los usuarios proporcionan un nombre de usuario y contraseña que está autenticado por la base de datos. Esto les da acceso a los datos en el almacén de datos asociado a sus actividades. v2.0 ICDE tendrá que admitir la autenticación similar para los usuarios, y ampliar este para manejar herramientas de terceros. También, como herramientas de terceros pueden ser ejecutar de forma remota y acceder a los datos del ICDE sobre una red insegura, los datos en tránsito deben estar cifrados.
recuperables son cómo se detectan los fallos y recuperación comienza (preferiblemente de forma automática), y el tiempo

3.6 disponibilidad

resolver las transacciones que estaban vuelo fl in- cuando se produjo el error. temas interesantes para aplicaciones

La disponibilidad se relaciona con la fiabilidad de una aplicación. Si una aplicación no está disponible para su uso cuando sea necesario, entonces es poco probable que sea llenado cumplir sus requisitos funcionales. La disponibilidad es relativamente fácil de especificar y medir. En cuanto a las especi fi caciones, muchas aplicaciones de TI deben estar disponibles al menos durante las horas normales de trabajo. La mayoría de los sitios de Internet desean 100% de disponibilidad, ya que no existen
de base de datos falla, no está disponible hasta que se haya recuperado. Esto significa reiniciar la aplicación de servidor, y las horas regulares de línea. Para un sistema vivo, la disponibilidad puede ser medido por la proporción del tiempo que se requiere es utilizable.

Los fallos en las aplicaciones hacen que ellos no estén disponibles. Fracasos impacto en la fiabilidad de una aplicación, que normalmente se mide por el tiempo medio entre fallos. La longitud de tiempo cualquier período de
aplicación o sistema. Un sistema de base de datos es el ejemplo clásico de un sistema de reembolso. Cuando un servidor indisponibilidad tiene una duración está determinada por la cantidad de tiempo que se necesita para detectar el fallo y reiniciar el sistema. En consecuencia, las aplicaciones que requieren alta disponibilidad minimizar o eliminar preferiblemente puntos únicos de fallo, y mecanismos de instituto que detectan automáticamente fracaso y reinicie los componentes que han fallado.

capacidad de restablecer los niveles de rendimiento requeridos y recuperar datos afectados después de un fallo de

Replicación de los componentes es una estrategia de probada eficacia para alta disponibilidad. Cuando un componente replicado falla, la aplicación puede continuar con la ejecución usando réplicas que siguen funcionando. Esto puede conducir a una disminución del rendimiento, mientras que el componente que ha fallado es hacia abajo, pero la disponibilidad no se vea comprometida.

Recuperabilidad está estrechamente relacionado con la disponibilidad. Una aplicación es recuperable si tiene la

Durante el proceso de recuperación, la aplicación no está disponible, y por lo tanto el tiempo medio para recuperar es una medida importante a tener en cuenta.

3.6.1 Disponibilidad para la aplicación ICDE

Mientras que la alta disponibilidad para la aplicación ICDE es deseable, es sólo es crucial que estén disponibles durante las horas de oficina de la del medio ambiente o fi cina se despliega en. Esto deja un amplio margen de tiempo de inactividad para las necesidades tales como actualización del sistema, copia de seguridad y mantenimiento. Sin embargo, la solución debe incluir mecanismos tales como la replicación componente para asegurar tan cerca de 100 Disponibilidad% como sea posible durante el horario comercial.

3.7 Integración

La integración se refiere a la facilidad con la que una aplicación se puede incorporar de forma útil en un contexto de aplicación más amplio. El valor de una aplicación o componente de frecuencia se puede aumentar en gran medida si su funcionalidad o datos pueden ser utilizados de manera que el diseñador no anticipó originalmente. Las estrategias más comunes para proporcionar la integración son a través de la integración de datos o la prestación de un API.

La integración de datos implica el almacenamiento de los datos de una aplicación manipula de manera que otras aplicaciones pueden tener acceso. Esto puede ser tan simple como usar una base de datos relacional estándar para el almacenamiento de datos, o tal vez la implementación de mecanismos para extraer los datos en un formato conocido como XML o un texto separado por comas fi l que otras aplicaciones puedan ingerir.

Con la integración de datos, las formas en que se utilizan los datos (o abusado) por otras aplicaciones es bastante fuera del control del propietario original de datos. Esto es porque las reglas de integridad de datos y de negocios impuestas por la lógica de aplicación son por-pasado. La alternativa es que la interoperabilidad a través de una API (ver Fig. 3.3). En este caso, los datos en bruto de la aplicación posee se oculta detrás de un conjunto de funciones que facilitan el acceso controlado a los datos externa. De esta manera, las reglas de negocio y de seguridad se pueden hacer cumplir en la implementación de la API. La única forma de acceder a los datos y la integración con la aplicación es mediante el uso de la API suministrado.

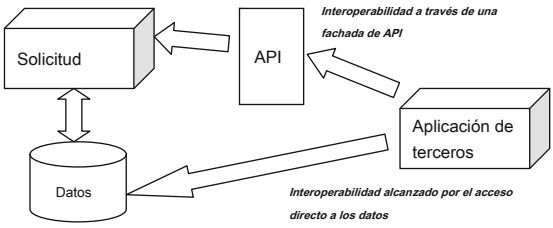


Fig. 3.3 opciones de integración

La elección de la estrategia de integración no es simple. La integración de datos es flexible y simple. Las aplicaciones escritas en cualquier lenguaje pueden procesar texto, o acceder a bases de datos relacionales con SQL. La construcción de un API requiere más esfuerzo, pero proporciona un entorno mucho más controlado, en cuanto a la exactitud y la seguridad, para la integración. También es mucho más robusto desde una perspectiva de integración, ya que los clientes API están aislados de muchos de los cambios en las estructuras de datos subyacentes. Que no se rompen cada vez que el formato es modi fi cado, como los formatos de datos no están directamente expuestos y accesibles. Como siempre, la mejor elección de la estrategia depende de lo que quiere lograr, y qué problemas existen.

3.7.1 Integración para la aplicación ICDE

más difícil". CAR Hoare. 36

Los requisitos de integración para ICDE giran en torno a la necesidad de apoyar las herramientas de análisis de terceros partido. Tiene que ser un bien de fi nida y el mecanismo entiende por herramientas de terceros para acceder a los datos en el almacén de datos ICDE. Como herramientas de terceros a menudo se ejecutará de forma remota desde un almacén de datos ICDE, la integración a nivel de datos, permitiendo que las herramientas de acceso directo al almacén de datos, parece poco probable que sea viable. Por lo tanto es probable que sea facilitado a través de una API soportado por la aplicación ICDE integración.

3.8 otros atributos de calidad

Hay muchos otros atributos de calidad que son importantes en diversos contextos de aplicación. Algunos de estos son:

- Portabilidad: Una aplicación puede ejecutarse fácilmente en un software diferente / deficiencias, y la otra forma es hacerlo tan complicado que no hay deficiencias obvias. El primer método es mucho plataforma de hardware a la que se ha desarrollado para? Portabilidad depende de las opciones de la tecnología de software utilizados para implementar la aplicación y las características de las plataformas que necesita para ejecutar. Fácilmente bases de código portátiles tendrán sus dependencias plataforma aislados y encapsulados en un pequeño conjunto de componentes que se pueden sustituir sin afectar al resto de la aplicación.

- la capacidad de prueba: ¿Qué tan fácil o di fi culto es una aplicación para probar? Las primeras decisiones de diseño puede afectar en gran medida la cantidad de casos de prueba que se requieren. Como regla general, cuanto un diseño más complejo, el culto más fi cultades que es completamente prueba. La simplicidad tiende a promover la facilidad de las pruebas. [7] Del mismo modo, escribiendo menos de su propio código mediante la incorporación de componentes sometidas a pruebas previas reduce el esfuerzo de prueba.
- la capacidad de soporte: Esta es una medida de cuán fácil es una aplicación para apoyar una vez que se se despliega. Soporte normalmente implica problemas de diagnóstico y la fi jación que

[7] "Hay dos formas de construir un diseño de software: Una forma es hacerlo tan simple que obviamente no hay

ocurrir durante el uso de la aplicación. sistemas soportables tienden a proporcionar instalaciones para el diagnóstico explícitos, tales como registros de errores de aplicaciones que registran las causas de los fracasos. También se construyen de forma modular para que los xes código fi se pueden implementar sin el uso de aplicaciones severamente incomodar.

3.9 compromisos de diseño

Si la vida de un arquitecto eran simples, diseño sería sólo tendrá por consecuencia las políticas y mecanismos de construcción en una arquitectura para satisfacer las características de calidad requeridas para una aplicación determinada. Elegir un atributo de calidad requerido, y proporcionar mecanismos para apoyarlo.

Por desgracia, este no es el caso. atributos de calidad no son ortogonales. Ellos interactúan de maneras sutiles, lo que significa un diseño que satisface el requisito atributo de una calidad puede tener un efecto perjudicial sobre la otra. Por ejemplo, un sistema de alta seguridad puede ser difícil o imposible de integrar en un entorno abierto. Una aplicación de alta disponibilidad puede disyuntiva menor rendimiento para una mayor disponibilidad. Una aplicación que requiere un alto rendimiento puede estar vinculada a una plataforma en particular, y por lo tanto no ser fácilmente portátil.

La comprensión de las compensaciones entre requisitos de atributos de calidad, y el diseño de una solución que hace compromisos sensatos es una de las partes más difíciles de la función de la arquitectura. Simplemente no es posible satisfacer todos los requisitos de la competencia. Es el trabajo del arquitecto de desentrañar estas tensiones, hacerlas explícitas a los actores del sistema, si es necesario priorizar, y explícitamente documentar las decisiones de diseño.

¿Le suena fácil? Si tan solo este fuera el caso. Es por eso que te pagan grandes cantidades de dinero.

3.10 Resumen

Los arquitectos deben gastar mucho esfuerzo atributos de calidad, precisamente, la comprensión, por lo que un diseño puede ser concebido para hacerles frente. Parte de la fi cultad cultly es que los atributos de calidad no siempre se expresan de manera explícita en los requisitos, o adecuadamente capturados por el equipo de ingeniería de requisitos. Es por eso que un arquitecto debe estar asociado con los requisitos de ejercicio de recopilación para el sistema, para que puedan formular las preguntas correctas para exponer y concretar las características de calidad que deben ser abordados.

Por supuesto, la comprensión de los requisitos de atributos de calidad es más que un requisito previo necesario para el diseño de una solución para satisfacerlas. Conflictivas atributos de calidad son una realidad en todas las aplicaciones de la complejidad, incluso mediocre. Creación de soluciones que elegir un punto en el espacio de diseño que satisface adecuadamente estos requisitos es muy difícil, tanto técnica como social. Esta última implica comunicaciones con las partes interesadas para discutir las tolerancias de diseño, el descubrimiento de escenarios

cuando ciertos requisitos de calidad pueden ser de forma segura relajado, y la comunicación clara de compromisos de diseño para que las partes interesadas comprendan lo que están firmando.

3.11 Lectura adicional

El amplio tema de los requisitos no funcionales está cubierto muy a fondo en:

L. Chung, B. Nixon, E. Yu, J. Mylopoulos, (editores). Requisitos no funcionales en Ingeniería de Software de la serie: La Serie Internacional Kluwer en Ingeniería de Software. Vol. 5, Editores Académicos Kluwer. 1999.

Una excelente referencia general sobre la seguridad y las técnicas y tecnologías de un arquitecto debe tener en cuenta es:

J. Ramachandran. Diseño de la seguridad Soluciones de arquitectura. Wiley & Sons, 2002.

Un enfoque interesante y práctico para evaluar la capacidad de modi fi de una arquitectura utilizando herramientas de la arquitectura de reconstrucción y las métricas de análisis de impacto se describe en:

I. Gorton, L. Zhu. Herramienta de soporte para la Reconstrucción Just-in-Time Arquitectura y Evaluación: un relato de experiencia. Conferencia Internacional sobre Ingeniería de Software (CISE)

Capítulo 4

Una introducción a Middleware arquitecturas y tecnologías

4.1 Introducción

No soy realmente un gran entusiasta de establecer analogías fuertes entre el papel de un arquitecto de software y la de un arquitecto edificio tradicional. Hay similitudes, pero también muchas diferencias profundas. [1] Pero vamos a ignorar esas diferencias para una segunda, con el fin de ilustrar el papel de middleware en la arquitectura de software.

Cuando un arquitecto diseña un edificio, crean dibujos, esencialmente un diseño que muestra, desde diversos ángulos, la estructura y las propiedades geométricas del edificio. Este diseño se basa en los requisitos de la construcción, tales como el espacio disponible, la función (o fi cina, iglesia, centro comercial, casa), estética deseada y cualidades funcionales y presupuesto. Estos dibujos son una representación abstracta del artefacto concreto previsto (sic).

Obviamente hay una gran cantidad de esfuerzo de diseño sigue siendo necesaria para convertir los dibujos de arquitectura en algo que la gente puede empezar a construir. Hay diseño detallado de las paredes, diseños de piso, escaleras, instalaciones eléctricas, de agua y tuberías para nombrar sólo unos pocos. Y como cada uno de estos elementos de un edificio está diseñado en detalle, se seleccionan los materiales y componentes para la construcción de cada uno adecuados.

Estos materiales y componentes son los bloques básicos de construcción para edificios. Han sido creadas para que puedan cumplir llenar las mismas necesidades esenciales en muchos tipos de edificios, ya sean de torres de oficina, estaciones de ferrocarril o casas de familias humildes.

Aunque quizás no es la analogía con más glamour, me gusta pensar en middleware como el equivalente de la tubería o tuberías o cableado para las aplicaciones de software. Las razones son las siguientes:

1 Middleware proporciona maneras probadas para conectar los diversos componentes de software en una aplicación para que puedan intercambiar información a través de mecanismos relativamente fáciles de usar. Middleware proporciona las tuberías para el envío de datos entre los componentes, y se puede utilizar en una amplia gama de diferentes dominios de aplicación.

---

1 El siguiente artículo trata de cuestiones: J. Baragry y K. Reed. ¿Por qué necesitamos una visión diferente de arquitectura de software. La Conferencia de Trabajo IEEE / IFIP en Arquitectura de Software (WICSA), Amsterdam, Países Bajos, 2001.

DOI 10.1007 / 978-3-642-19176-3_4, # Springer-Verlag Berlin Heidelberg 2011

- Middleware se puede utilizar para conectar juntos numerosos componentes en útil, bien
  topologías entendido. Las conexiones pueden ser de uno a uno, uno-a-muchos o muchos manyto.

- Desde la perspectiva del usuario de la aplicación, middleware es completamente oculto. usuarios
  interactuar con la aplicación, y no importa cómo la información se intercambia internamente.
  Mientras funciona, y funciona bien, es el middleware invisible
  infraestructura.
- Los usuarios de la aplicación única de tiempo son siempre conscientes del papel que juega es el middleware
  cuando falla. Esto es por supuesto muy parecido a la plomería real y sistemas de cableado.

No es probable que sea prudente para empujar la analogía de plomería más. Pero es de esperar que haya servido a su propósito. Middleware proporciona la infraestructura listos para usar para la conexión de componentes de software. Puede ser utilizado en una gran variedad de diferentes dominios de aplicación, ya que ha sido diseñado para ser gurable general y con fi para satisfacer las necesidades comunes de las aplicaciones de software.

4.2 Middleware Tecnología Clasi fi cación

Middleware consiguió su etiqueta, ya que fue concebido como una capa de software “plumbinglike” infraestructura que estaba sentado entre la aplicación y el sistema operativo, es decir, la mitad de las arquitecturas de aplicaciones. Por supuesto, en realidad, el middleware es mucho más compleja que la plomería o una simple capa aislante una aplicación de los servicios del sistema operativo subyacente.

Diferentes dominios de aplicación tienden a considerar diferentes tecnologías como middleware. Este libro trata de las aplicaciones informáticas de comunicación, y en ese dominio hay una colección bastante bien entendido que se conoce normalmente como middleware. Figura 4.1
proporciona una clasi fi cación de estas tecnologías, y los nombres de algunos productos / tecnologías que representan a cada categoría de ejemplo. Una breve explicación de las categorías están por debajo, y el resto de este capítulo y los dos siguientes van a describir en detalle cada uno:

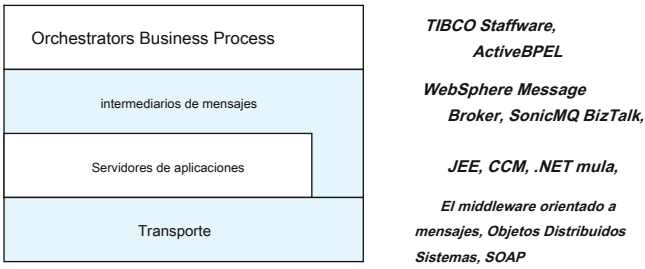


Fig. 4.1 La clasificación de las tecnologías de middleware 40

- La capa de transporte representa los tubos básicos para el envío de solicitudes y moviendo

  datos entre los componentes de software. Estos tubos proporcionan instalaciones y mecanismos que hacen que el intercambio de datos directo en arquitecturas de aplicaciones distribuidas simples.

- Los servidores de aplicaciones suelen ser construido en la parte superior de los servicios básicos de transporte.

  Ellos proporcionan capacidades adicionales, tales como los servicios de transacciones, seguridad y directorio. También apoyan un modelo de programación para crear aplicaciones basadas en servidor multiproceso que explotan estos servicios adicionales.

- intermediarios de mensajes explotan, ya sea un servicio básico de transporte y / o aplicación

  servidores y agregan un motor de procesamiento de mensajes especializado. Este motor proporciona funciones para la transformación rápida de mensajes y funciones de programación de alto nivel para de fi nir cómo intercambiar, manipular y enrutar los mensajes entre los diversos componentes de una aplicación.

- orchestrators de procesos de negocio (BPO) aumentar las características del corredor mensaje a

  el trabajo de apoyo fl ujo de aplicaciones de estilo. En tales aplicaciones, los procesos de negocio pueden tomar muchas horas o días en completarse debido a la necesidad de las personas para realizar ciertas tareas. BPO proporcionan las herramientas para describir este tipo de procesos de negocio, ejecutarlas y gestionar los estados intermedios, mientras que cada paso en el proceso es ejecutado.

4.3 Objetos Distribuidos

La tecnología de objetos distribuidos es un miembro de la familia venerable middleware. Mejor caracterizado por CORBA, [2] distribuido middleware basado en objetos ha estado en uso desde la década de 1990 anteriores. Como muchos lectores estarán familiarizados con CORBA y similares, sólo los conceptos básicos son brevemente cubierto en esta sección para la integridad.

Un simple escenario de un cliente que envía una solicitud a un servidor a través de un corredor de petición de objeto (ORB) se muestra en la Fig. 4.2 . En CORBA, criado objetos interfaces de apoyo que están especi fi usando IDL de CORBA (lenguaje de descripción de interfaz). IDL interfaz de fi ne los métodos que un objeto de servidor soporta, junto con el parámetro y tipos de retorno. Un ejemplo trivial IDL es:

```
ServerExample módulo {
        interfaz MyObject {

                isAlive cadena (); }; };
```

Este IDL interfaz define un objeto CORBA que soporta un solo método, isAlive, que devuelve una cadena y no toma ningún parámetro. Un compilador de IDL se utiliza para procesar la interfaz de fi niciones. El compilador genera un esqueleto objeto en

[2] Common Object Request Broker Architecture.

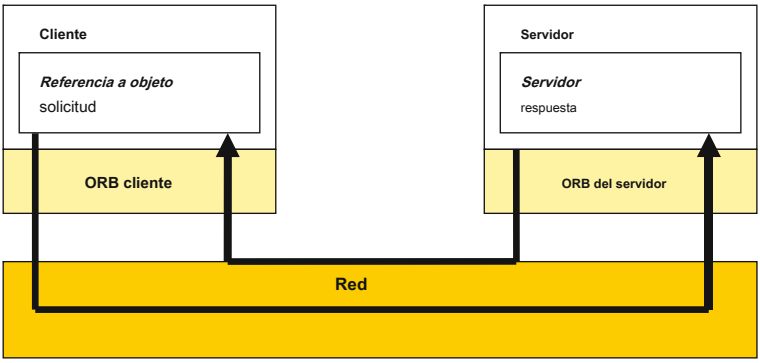


Fig. 4.2 objetos distribuido utilizando CORBA 42

un lenguaje de programación de destino (por lo general, pero no necesariamente, C ++ o Java). El esqueleto objeto proporciona los mecanismos para llamar a los métodos de implementación del servidor. El programador debe entonces escribir el código para implementar cada método criado en un lenguaje de programación nativo:

```
class MyServant extends _MyObjectImplBase {
  public String isAlive()   {
    return "\nLooks like it…\n";
    }
}
```

El proceso del servidor debe crear una instancia del servidor y hacer que sea exigible a través del ORB:

```
ORB orb = ORB.init(args, null);
MyServant objRef = new MyServant();
orb.connect(objRef);
```

Un proceso de cliente ahora puede inicializar un ORB cliente y obtener una referencia al servidor que reside dentro del proceso del servidor. Sirvientes suelen almacenar una referencia a sí mismos en un directorio. Clientes consultar el directorio utilizando un nombre lógico simple, y devuelve una referencia a un sirviente que incluye su ubicación en la red y la identidad proceso.

```
ORB orb = ORB.init(args, null);
// Lookup is a wrapper that actually access the CORBA Naming //
Service directory - details omitted for simplicity
MyServant servantRef = lookup(“Myservant”)
String reply = servantRef.isAlive();
```

La llamada sirviente se parece a una llamada sincrónica a un objeto local. Sin embargo, los mecanismos de ORB transmiten, o Mariscal, la solicitud y los parámetros asociados en toda la red al servidor. El código del método se ejecuta, y el resultado se calculan las referencias de vuelta al cliente espera.

Esta es una descripción muy simplista de la tecnología de objetos distribuidos. Hay mucho más detalle que debe ser abordado para construir sistemas reales, temas como excepciones, los funcionarios de localización y multihilo, por nombrar sólo algunos. Desde la perspectiva de un arquitecto, sin embargo, las siguientes son algunas de las preocupaciones esenciales de diseño que deben ser abordados en aplicaciones:

- **Las solicitudes a los servidores son llamadas remotas, y por lo tanto relativamente caro (lento) como**

  que atraviesan el ORB y de la red. Esto tiene un impacto en el rendimiento. Siempre es aconsejable para diseñar interfaces de modo que las llamadas remotas pueden reducirse al mínimo, y el rendimiento es mayor.

- **Al igual que cualquier aplicación distribuida, los servidores pueden ser intermitente o permanente**

  no disponible debido a la red o proceso, o fallo de la máquina. Las aplicaciones necesitan estrategias para hacer frente al fracaso y mecanismos para reiniciar servidores que han fallado.
- **Si un servidor tiene relación con el estado de una interacción con un cliente (por ejemplo, un cliente**

  objeto almacena el nombre / dirección), y el servidor falla, se pierde el estado. Mecanismos para la recuperación del estado en consecuencia deben ser diseñados.

4.4 middleware orientado a mensajes

Orientado a mensajes middleware (MOM) es una de las tecnologías clave para la construcción de sistemas empresariales a gran escala. Es el pegamento que une a las aplicaciones de otro modo independientes y autónomos y los convierte en un sistema único e integrado. Estas aplicaciones pueden ser construidas utilizando diversas tecnologías y ejecutarse en diferentes plataformas. Los usuarios no están obligados a reescribir sus aplicaciones existentes o realizar cambios sustanciales (y arriesgadas) sólo para que jueguen un papel en una aplicación en toda la empresa. Esto se logra mediante la colocación de una cola entre emisores y receptores, proporcionando un nivel de indirección durante las comunicaciones.

Cómo MOM se puede utilizar dentro de una organización se ilustra en la Fig. 4.3 . los
MOM crea una bus de software para la integración de aplicaciones de cosecha propia con herencia

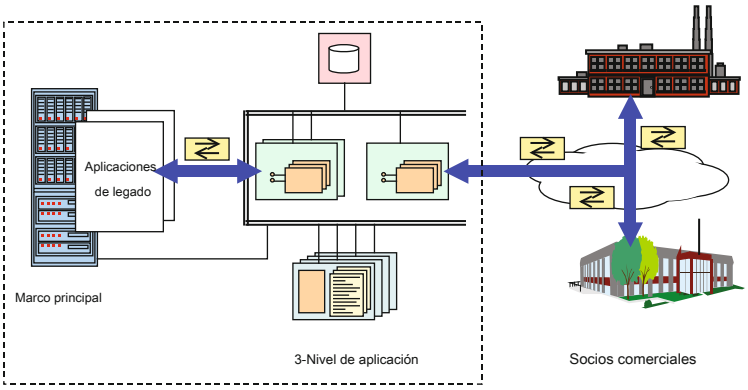


Fig. 4.3 Integración a través de la mensajería

aplicaciones y la conexión de aplicaciones locales con los sistemas de negocio proporcionada por los socios comerciales.

4.4.1 Fundamentos de MOM

MOM es una, la tecnología asíncrona inherentemente imprecisa. Esto significa que el emisor y el receptor de un mensaje no están estrechamente unidas, a diferencia de las tecnologías de middleware síncronos como CORBA. tecnologías de middleware síncronos tienen muchos puntos fuertes, pero pueden dar lugar a diseños frágiles si todos los componentes y los enlaces de red, siempre hay que estar trabajando al mismo tiempo para todo el sistema para operar con éxito.

Una infraestructura de mensajería desacopla emisores y receptores utilizando una cola de mensajes intermedia. El remitente puede enviar un mensaje a un receptor y saber que va a ser entregado finalmente, incluso si el enlace de red está inactivo o el receptor no está disponible. El remitente simplemente le dice a la tecnología MOM para entregar el mensaje y luego continúa con su trabajo. Remitentes no son conscientes de qué aplicación o proceso eventualmente procesa la solicitud. Figura 4.4 representa este envío-recepción mecanismo básico.

MOM se suele implementar como un servidor que puede manejar mensajes de múltiples clientes simultáneos.[3] Con el fin de desacoplar los emisores y receptores, el MOM proporciona colas de mensajes que los mensajes de remitentes y receptores en lugar eliminan los mensajes. Un servidor MOM puede crear y administrar los mensajes múltiples colas, y puede manejar múltiples mensajes que se envían desde las colas utilizando simultáneamente hilos organizados en una agrupación de hebras. Uno o más procesos pueden enviar mensajes a una cola de mensajes, y cada cola pueden tener uno o muchos receptores. Cada cola tiene un nombre que los emisores y receptores especifican cuando realizan enviar y recibir operaciones. Esta arquitectura se ilustra en la Fig. 4.5 . mensajes. En este tipo de aplicación, 'Enviar' y 'recibir' colas se mantienen en los mismos sistemas que se comunican. 44

Un servidor MOM tiene una serie de responsabilidades básicas. En primer lugar, se debe aceptar un mensaje de la aplicación de envío, y enviar un reconocimiento de que se ha recibido el mensaje. A continuación, se debe colocar el mensaje en el extremo de la cola que fue fi especificados por el remitente. Un remitente puede enviar muchos mensajes a una cola

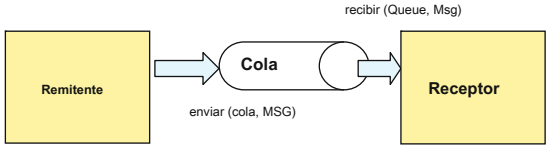
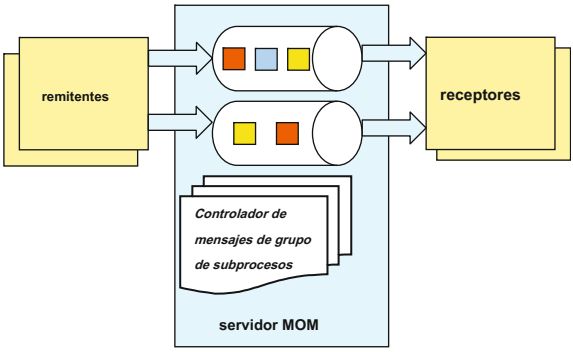


Fig. 4.4 fundamentos de MOM

[3] MOM también se puede implementar simplemente de un modo punto a punto sin un servidor centralizado cola de

Fig. 4.5 Anatomía de un servidor MOM



antes de que los receptores de eliminarlos. Por lo tanto el MOM debe estar preparado para mantener mensajes en una cola durante un período prolongado de tiempo.

Los mensajes se entregan a los receptores de una manera First-In-First-Out (FIFO), es decir, el orden en que llegan a la cola. Cuando un receptor solicita un mensaje, el mensaje a la cabeza de la cola se entrega al receptor, y tras la recepción exitosa, el mensaje se elimina de la cola.

El asíncrono, carácter desconectado de la tecnología de mensajería hace que sea una herramienta extremadamente útil para resolver muchos problemas de diseño de aplicaciones comunes. Estos incluyen escenarios en los que:

- El emisor no necesita una respuesta a un mensaje. Sólo quiere enviar el mensaje a otra aplicación y continuar con su propio trabajo. Esto se conoce como sendand-olvidar mensajería.
- El emisor no necesita una respuesta inmediata a un mensaje de solicitud. El receptor puede tardar varios minutos quizá para procesar una solicitud y el emisor puede estar haciendo un trabajo útil, mientras tanto, en lugar de esperar.
- El receptor, o la conexión de red entre el emisor y el receptor, puede no funcionar de forma continua. El remitente se basa en el MOM para entregar el mensaje siguiente cuando se establece una conexión. La capa MOM debe ser capaz de almacenar los mensajes para su posterior entrega, y, posiblemente, la recuperación de los mensajes no enviados después de fallos del sistema.

4.4.2 Explotación Características avanzadas MOM

Las características básicas de la tecnología MOM rara vez son su fi ciente en aplicaciones empresariales. sistemas de misión crítica necesitan garantías mucho más fuertes de la entrega de mensajes y el rendimiento que puede ser proporcionado por un servidor de base de MOM. Commercialoff-the-shelf (COTS) los productos de MOM por lo tanto suministran funciones avanzadas adicionales para aumentar la fiabilidad, facilidad de uso y la escalabilidad de los servidores de MOM. Estas características se explican en las siguientes secciones.

4.4.2.1 Entrega de mensajes

tecnologías de MOM están sobre la entrega de mensajes entre aplicaciones. En muchas aplicaciones de la empresa, esta entrega debe hacerse de forma fiable, dando las garantías remitente que el mensaje finalmente será procesada. Por ejemplo, una aplicación de procesamiento de una transacción de tarjeta de crédito puede colocar los detalles de la transacción en una cola para su posterior procesamiento, para añadir el total de la transacción a la cuenta del cliente. Si este mensaje se pierde debido al estrellarse servidor MOM - tales cosas ocurren - a continuación, el cliente puede ser feliz, pero la tienda donde se efectuó la compra y la compañía de tarjetas de crédito van a perder dinero. Tales escenarios, obviamente, no pueden tolerar la pérdida de mensajes, y deben asegurar la entrega fiable de mensajes.

la entrega de mensajes fiable, pero se produce a expensas del rendimiento. servidores de MOM normalmente ofrecen una gama de calidad de servicio (QoS) opciones que permiten un rendimiento del balance de arquitecto en contra de la posibilidad de perder mensajes. Tres niveles de garantía de entrega (o QoS) suelen estar disponibles, con los niveles más altos de fiabilidad siempre viene a costa de la reducción del rendimiento. Estas opciones de calidad de servicio son:

- **Mejor esfuerzo: El servidor MOM hará todo lo posible para entregar el mensaje. No entregado**

  mensajes sólo se mantienen en la memoria en el servidor y se pueden perder si un sistema falla antes de que se entregue un mensaje. las interrupciones de red o aplicaciones que reciben no disponibles también pueden provocar que los mensajes a tiempo y ser desechado.
- **Persistente: Las garantías de capa MOM para entregar mensajes a pesar del sistema y**

  fallos en la red. Los mensajes no entregados se registran en el disco, además de ser mantenidos en la memoria y por lo tanto **pueden ser recuperados y posteriormente entregadas después de un fallo del sistema. Esto se representa en la Fig. 4.6 . Los** mensajes se mantienen en un registro en disco para la cola hasta que haya sido entregada a un receptor.

- **transaccional: Los mensajes pueden ser agrupados en unidades de "todo o nada" para la entrega.**

  También, la entrega de mensajes puede ser coordinado con un ResourceManager externo tal como una base de datos. Más en la entrega transaccional se explica en las siguientes secciones.

Diversos estudios se han llevado a cabo para explorar las diferencias de rendimiento entre estos tres niveles de calidad de servicio. Todos ellos por su propia naturaleza son específico a una aplicación de referencia en particular, entorno de prueba y producto de MOM. Algunas conclusiones muy generales, se puede esperar para ver entre el 30 y el 80%

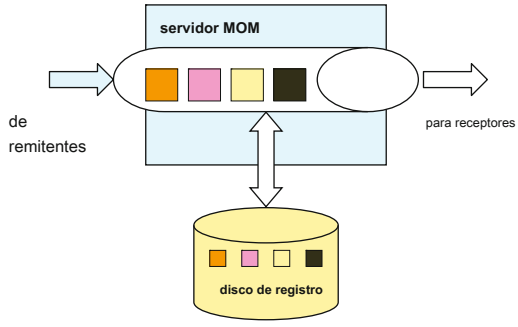


Fig. 4.6 la entrega de mensajes

reducción del rendimiento cuando se pasa de mejor esfuerzo a la mensajería persistente, dependiendo del tamaño del mensaje y la velocidad del disco. Transaccional será más lenta que persistente, pero a menudo no por mucho, ya que esto depende en gran medida de cómo están involucrados muchos participantes en las operaciones. Vea la sección de lectura adicional al final de este capítulo para algunos punteros a estos estudios.

4.4.2.2 Transacciones

mensajería transaccional normalmente se basa en mensajes persistentes. Se integra perfectamente con las operaciones de mensajería código de la aplicación, no permitiendo que los mensajes transaccionales que se enviarán hasta que la aplicación emisora se compromete la transacción de cerramiento. La funcionalidad básica transaccional MOM permite que las aplicaciones para la construcción de lotes de mensajes que se envían como una sola unidad atómica cuando la aplicación se compromete.

Los receptores también deben crear un ámbito de transacción y pedir para recibir lotes completos de los mensajes. Si la transacción se confirma por los receptores, estos mensajes transaccionales serán recibidos juntos en el orden en que fueron enviados, y luego se eliminan de la cola. Si el receptor se anula la transacción, anymessages ya leídos serán puestos de nuevo en la cola de espera, listo para el siguiente intento de manejar la misma transacción. Además, las transacciones consecutivas enviadas desde el mismo sistema a la misma cola llegarán en el orden en que se cometieron, y cada mensaje será entregado a la aplicación una sola vez para cada transacción confirmada.

mensajería transaccional también permite que envía y recibe mensajes que coordinarse con otras operaciones transaccionales, tales como actualizaciones de la base. Por ejemplo, una aplicación puede iniciar una transacción, enviar un mensaje, actualizar una base de datos y luego confirmar la transacción. La capa de MOM no hará que el mensaje disponible en la cola hasta que se confirma la transacción, ya sea asegurando que el mensaje se envía y la base de datos se actualiza, o que ambas operaciones se revierten y parece no haber sucedido.

Un ejemplo de seudocódigo de integrar las actualizaciones de mensajería y de base de datos se muestra en la Fig. 4.7 . El código de la aplicación remitente utiliza declaraciones de demarcación de transacción (la forma exacta varía entre los sistemas MOM) para especificar el ámbito de la transacción. Todas las declaraciones entre el empezar y cometer declaraciones de transacción son considerados como parte de la transacción. Tenga en cuenta que tenemos dos transacciones independientes, que se producen en este ejemplo. Las transacciones emisor y receptor están separados y se comprometen (o abortar) individualmente.

4.4.2.3 La agrupación

servidores de MOM son el mecanismo de intercambio de mensajes principal en muchas aplicaciones empresariales. Si un servidor MOM no está disponible debido a servidor o fallo de la máquina, a continuación, las aplicaciones no pueden comunicarse. No es sorprendente entonces, las tecnologías de MOM fuerza industrial hacen posible agrupar los servidores de MOM, Ejecución de instancias del servidor en múltiples máquinas (ver Fig. 4.8).

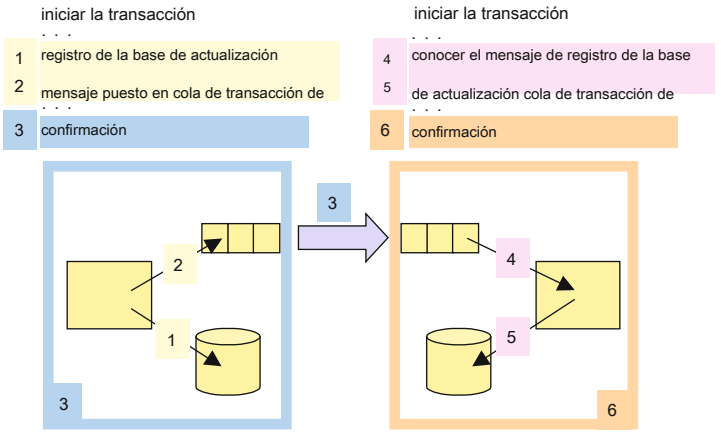
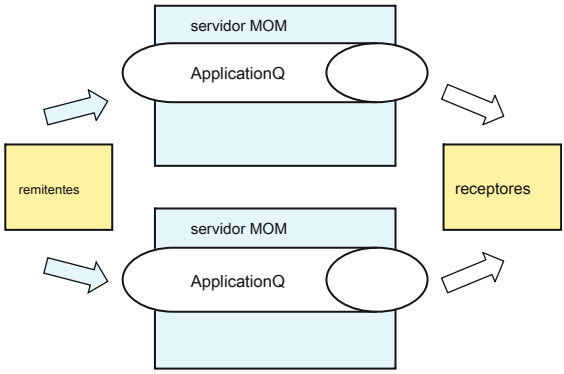


Fig. 4.7 mensajería transaccional

Fig. 4.8 La agrupación de servidores mamá para la fiabilidad y escalabilidad

agrupación. 48



Exactamente cómo funciona la agrupación depende del producto. Sin embargo, el esquema en la Fig. 4.8 es típico. Múltiples instancias de servidores de MOM son con fi gurada en un clúster lógico. Cada servidor es compatible con el mismo conjunto de colas, y la distribución de estas colas a través de servidores es transparente para los clientes de MOM. clientes de MOM se comportan exactamente igual que si había un servidor físico y cola instancia.

Cuando un cliente envía un mensaje, una de las instancias de la cola se selecciona y el mensaje almacenado en la cola. Del mismo modo, cuando un receptor solicita un mensaje, una de las instancias de la cola se selecciona y un mensaje eliminado. La implementación del servidor MOM agrupación es responsable de dirigir las peticiones de cliente a instancias de colas individuales. Esto se puede hacer de forma estática, cuando un cliente abre una conexión con el servidor, o dinámicamente, para cada solicitud. [4]

[4] Una aplicación que necesita para recibir los mensajes en el orden en que se envían no es adecuado para funcionar en este modo una

Un clúster tiene dos beneficios. En primer lugar, si falla un servidor MOM, las otras instancias de cola están todavía disponibles para los clientes a utilizar. Aplicaciones en consecuencia, pueden mantener la comunicación. En segundo lugar, la carga de solicitudes de los clientes se puede transmitir a través de los servidores individuales. Cada servidor sólo ve una fracción (idealmente 1 / [número de servidores] en el cluster) de la general tráfico c. Esto ayuda a distribuir la carga de mensajes a través de múltiples máquinas, y puede proporcionar mucho más alto rendimiento de la aplicación.

4.4.2.4 dos vías de mensajería

Aunque la tecnología MOM es inherentemente asíncrona y desacopla emisores y receptores, sino que también puede ser utilizado para las comunicaciones sincrónicas y la construcción de sistemas más fuertemente acoplados. En este caso, el remitente simplemente usa la capa de MOM para enviar un mensaje de solicitud a un receptor en una cola de solicitudes. El mensaje contiene el nombre de la cola a la que un mensaje de respuesta debe ser enviada. El remitente espera entonces hasta que el receptor envía de vuelta un mensaje de respuesta en una cola de respuestas, tal como se muestra en la Fig. 4.9 . Este estilo sincrónica de mensajería usingMOM se utiliza con frecuencia en sistemas de la empresa, en sustitución de la tecnología síncrona convencional tal como CORBA. Hay una serie de razones por qué los arquitectos pragmáticos podrían optar por utilizar la tecnología de mensajería de esta manera, incluyendo:

- tecnología de mensajería se puede utilizar con las aplicaciones existentes a bajo costo y con un riesgo mínimo. adaptadores están disponibles, o pueden ser fácilmente escrito a la interfaz entre las tecnologías y aplicaciones de mensajería de uso común. Las aplicaciones no tienen que ser reescritos o portado antes de que puedan ser integrados en un sistema mayor.

- tecnologías de mensajería tienden a estar disponible en una gama muy amplia de plataformas, por lo que es fácil la integración de aplicaciones heredadas o sistemas de empresas siendo dirigido por los socios comerciales.

- Las organizaciones que ya hayan comprado y adquirido experiencia en el uso, una tecnología de mensajería y pueden no necesitar las características adicionales de una tecnología de servidor de aplicaciones.

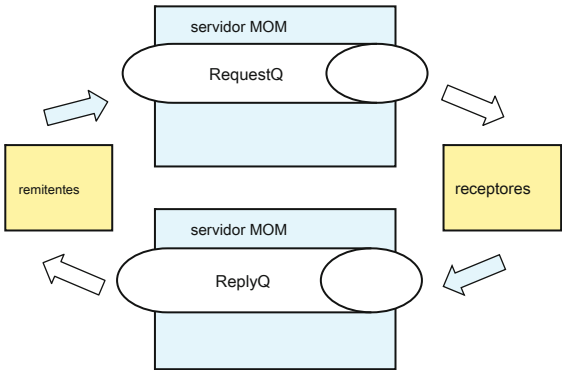


Fig. 4.9 mensajes de petición-respuesta

4.4.3 Publish-Subscribe

MOM es un método probado y eficaz para la construcción de sistemas de la empresa de forma flexible. Pero, como todo, tiene sus limitaciones. La principal es que MOM es inherentemente una tecnología de uno-a-uno. Un emisor envía un único mensaje a una sola cola, y un receptor recupera ese mensaje para la cola. No todos los problemas se resuelven tan fácilmente por un estilo de mensajería 1-1. Aquí es donde publicación-suscripción arquitecturas entran en escena.

Publicación-suscripción de mensajería extiende los mecanismos básicos de MOM para apoyar **1 a muchos, muchos a muchos, y muchos a 1 comunicaciones de estilo. Editores envían una sola copia de un mensaje dirigido a una llamada tema, o tema. Los temas son un nombre lógico para el equivalente de publicación-suscripción de** una cola en la tecnología básica de MOM. Los suscriptores escuchar los mensajes que se envían a los temas que les interesan. El servidor de publicación-suscripción luego distribuye cada mensaje enviado en un tema para cada suscriptor **que está escuchando en ese tema. Este esquema básico se representa en la Fig. 4.10 . En términos de la articulación** flexible, publicación-suscripción tiene algunas propiedades atractivas. Emisores y receptores se desacoplan, respectivamente, cada uno de los cuales desconocen aplicaciones recibirán un mensaje, y que en realidad enviado el mensaje. Cada tema también puede tener más de un editor, y los editores pueden aparecer y desaparecer de forma dinámica. Esto da una considerable flexibilidad sobre los regímenes con fi guración estática. Del mismo modo, los suscriptores pueden suscribirse y darse de baja de forma dinámica a un tema. Por lo tanto el aparato de abonado para un tema puede cambiar en cualquier momento, y esto es transparente para el código de aplicación.

En publicación-suscripción de tecnologías, la capa de mensajería tiene la responsabilidad de gestionar los temas, y los suscriptores knowingwhich está escuchando qué temas. También tiene la responsabilidad de entregar cada mensaje enviado a todos los suscriptores actuales activos. Los temas pueden ser persistentes o no persistente, con los mismos efectos sobre la entrega de mensajes fiable como en MOM básica de punto a punto (que se explica en la sección anterior). Los mensajes también pueden ser publicados con una configuración opcional “time-to-live”. Esto le dice al servidor de publicación-suscripción para intentar entregar un mensaje a todos los suscriptores activos para el período de tiempo de vida, y después de que borrar el mensaje de la cola.

El protocolo subyacente una tecnología MOM utiliza para la entrega de mensajes puede afectar profundamente el rendimiento. Por defecto, la mayoría utiliza punto a punto sencillo

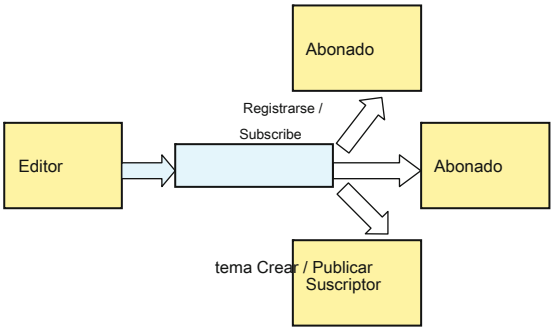


Fig. 4.10 Publicación-suscripción de

sockets TCP / IP. Las implementaciones de publicación-suscripción basado en la tecnología de mensajería punto a punto duplicado cada mensaje de operación de envío del servidor para cada suscriptor. Por el contrario, algunas tecnologías MOM son compatibles con los protocolos de difusión o multidifusión, que envían cada mensaje una sola vez en el alambre, y la capa de red se encarga de la entrega a múltiples destinos.

En la Fig. 4.11 , La arquitectura de multidifusión utiliza en editoras de TIBCO Rendezvous
La tecnología suscribirse se ilustra. Cada nodo en la red de publicación-suscripción ejecuta un proceso daemon conocido como RVD. Cuando se crea un nuevo tema, se le asigna una dirección IP multicast.

Cuando un editor envía un mensaje, su local, RVD daemon intercepta el mensaje y multidifusiones una única copia del mensaje en la red a la dirección asociada con el tema. Los demonios que escucha en la red recibe el mensaje, y cada uno comprueba si tiene alguna abonados locales al tema del mensaje en su nodo. Si es así, entrega el mensaje al abonado (s), de lo contrario se ignora el mensaje. Si un mensaje tiene abonados en una red remota, [5] un rvrd daemon intercepta el mensaje y envía una copia a cada red remota utilizando protocolos IP estándar. cada receptora rvrd daemon multicasts el mensaje a todos los abonados en su red local.

No es sorprendente que las soluciones basadas en multidifusión tienden a proporcionar un mejor rendimiento bruto y la escalabilidad de la mensajería mejor esfuerzo. No hace mucho tiempo, yo estaba

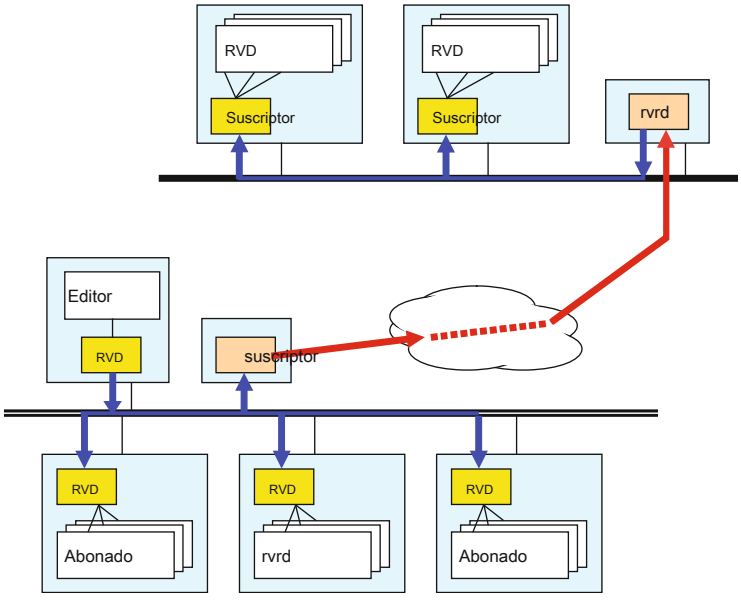
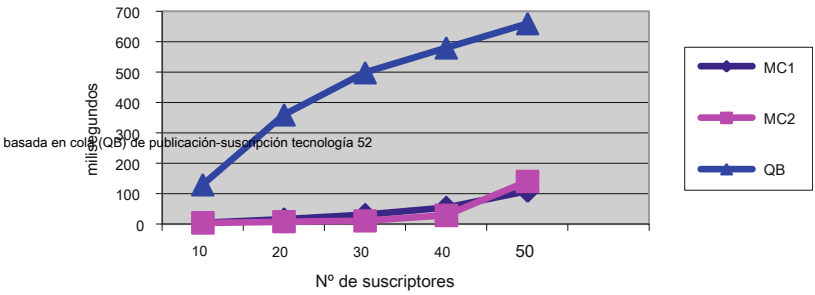


Fig. 4.11 la entrega de multidifusión para la publicación-suscripción

[5] Y la red de área extendida no soporta multidifusión.



basada en cola (QB) de publicación-suscripción tecnología 52

Fig. 4.12 Publicación-suscripción mejor rendimiento de mensajería esfuerzo: Comparación de las tecnologías de multidifusión 2 (MC1, MC2) con una

participa en un proyecto para cuantificar la diferencia entre el rendimiento esperado de multidifusión y soluciones de punto a punto. Hemos investigado esto escribiendo y la ejecución de algunos puntos de referencia para comparar el rendimiento relativo de las tres tecnologías, y la figura de publicación-suscripción. 4.12 muestra los resultados de referencia.

Se muestra el tiempo medio para la entrega de un único editor a entre 10 y 50 suscriptores concurrentes cuando el editor emite una ráfaga de mensajes tan rápido como sea posible. Los resultados muestran claramente que multicast publicación-suscripción es ideal para aplicaciones con exigencias de las latencias de mensaje de baja y por lo tanto, muy alto rendimiento.

4.4.3.1 comprensión de temas

Los temas son el equivalente de publicación-suscripción de colas. nombres de los temas son simplemente cadenas, y se especi fi can administrativamente o mediante programación cuando se crea el tema. Cada tema tiene un nombre lógico, que es fi especificados por todas las aplicaciones que deseen publicar o suscribirse usando el tema.

Algunas tecnologías de publicación-suscripción apoyan jerárquica tema de nombres. Los detalles exactos de cómo los mecanismos explican a continuación el trabajo dependen del producto, pero los conceptos son genéricos y funcionan de manera similar a través de implementaciones. Vamos a usar el ejemplo ligeramente facetious se muestra en la Fig. 4.13 de un árbol de temas de nomenclatura.

Cada cuadro representa un nombre de tema que puede ser utilizado para publicar mensajes. El nombre único para cada tema es una cadena totalmente cuali fi cada, con un "/" se utiliza como separador entre los niveles del árbol. Por ejemplo, los siguientes son los nombres de los temas válidos:

```
Sydney
Sydney/DevGroup
Sydney/DevGroup/Information
Sydney/DevGroup/Information/work
Sydney/DevGroup/Information/gossip
Sydney/SupportGroup
Sydney/SupportGroup/Information
Sydney/SupportGroup/Information/work
Sydney/SupportGroup/Information/gossip
```

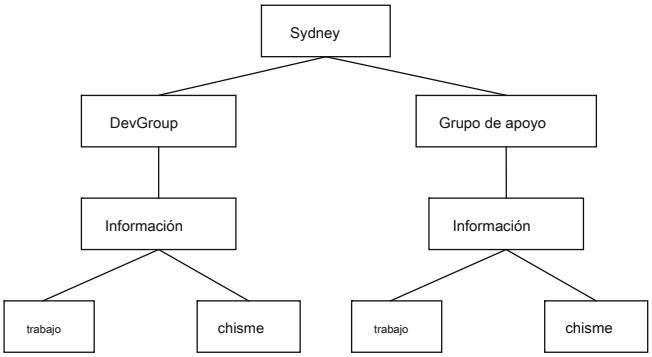


Fig. 4.13 Un ejemplo de nomenclatura jerárquica tema

nombres de los temas jerárquicos se vuelven realmente útil cuando se combina con comodines tema. En nuestro ejemplo, un "*" se utiliza como comodín que coincide con cero o más caracteres en un nombre de tema. Los suscriptores pueden utilizar comodines para recibir mensajes de más de un tema cuando se suscriban. Por ejemplo:

```
Sydney/*/Information
```

Esto coincide con los dos Sydney / DevGroup / Información y
Sydney / SupportGroup / Información. Del mismo modo, un abonado que especi fi ca en el tema siguiente:

```
Sydney/DevGroup/*/*
```

Esto recibir mensajes publicados en los tres temas dentro del Sydney / DevGroup rama de árbol. Como la suscripción de ramas enteras de un árbol de temas es muy útil, algunos productos son compatibles con una forma abreviada de lo anterior, el uso de otro carácter comodín como "**", es decir ,:

```
Sydney/DevGroup/**
```

Los comodines "**" también coincide con todos los temas que están en Sydney / DevGroup rama. un comodín, es de gran alcance, ya que es naturalmente extensible. Si se añaden nuevos temas en esta rama de la jerarquía de temas, los abonados no tienen que cambiar el nombre del tema en su solicitud de suscripción para recibir mensajes en los nuevos temas.

jerarquías nombre del tema cuidadosamente elaborados combinados con comodines hacen posible la creación de unas infraestructuras de mensajería flexibles muy fl. Considerar cómo las aplicaciones que desee suscribirse a varios temas, y organizan su diseño para apoyar estos.

4.5 Servidores de Aplicaciones

Hay muchos de fi niciones para servidores de aplicaciones, pero todos más o menos de acuerdo sobre los elementos principales. A saber, un servidor de aplicaciones es una tecnología de servidor basado en componente que se encuentra en el nivel medio de una arquitectura de N-capas, y proporciona distribuidos comunicaciones, seguridad, transacciones y persistencia. En esta sección, vamos a utilizar el Java Enterprise Edition [6] como nuestro ejemplo.

Los servidores de aplicaciones son ampliamente utilizados para construir aplicaciones orientados a Internet. Figura 4.14 muestra un diagrama de bloques de la arquitectura clásica de N-capas adoptada por muchos sitios web.

Una explicación de cada nivel es a continuación:

- Nivel de cliente: En una aplicación web, el nivel de cliente comprende típicamente un Internet navegador que envía solicitudes HTTP y descargas de páginas HTML desde un servidor web. Esta es la tecnología de los productos básicos, no es un elemento del servidor de aplicaciones.
- Capa Web: La capa web ejecuta un servidor web para manejar peticiones de clientes. Cuando un petición llega, el servidor web alojada invoca componentes de servidor web como los servlets, Java Server Pages (JSP) o Active Server Pages (ASP) en función de la Avor fl de servidor web que se utiliza. La solicitud fi cación de entrada es el componente web exacta para llamar. Este componente procesa los parámetros de la petición, y las utiliza para llamar a la capa de lógica de negocio para obtener la información requerida para satisfacer la solicitud. El componente Web presenta el resultado de la devolución al usuario como HTML a través del servidor web.
- Componentes de Negocio Nivel: Los componentes de negocio comprenden la actividad principal lógica para la aplicación. Los componentes de negocio se realizan mediante, por ejemplo, Enterprise JavaBeans (EJB) en JEE, componentes .NET o objetos CORBA. Los componentes de negocio reciben las peticiones de la capa web, y satisfacer las solicitudes por lo general mediante el acceso a una o más bases de datos, la devolución de los resultados a la capa web.

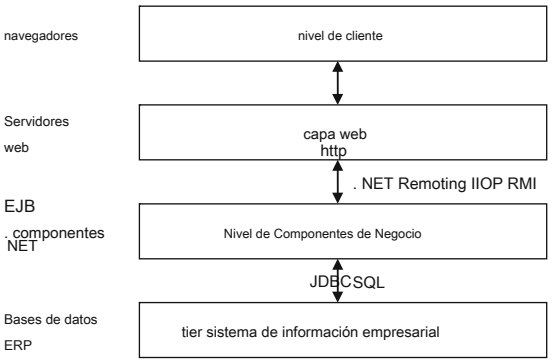


Fig. 4.14 arquitectura multicapa para aplicaciones web

[6] La plataforma era conocido como Java 2 Platform, Enterprise Edition o J2EE hasta que el nombre fue cambiado a Java EE en la

Un entorno de tiempo de ejecución conocido como envase acomoda los componentes. El contenedor proporciona una serie de servicios a los componentes que alberga. Estos variando en función del tipo de recipiente (por ejemplo, EJB, .NET, CORBA), pero incluir transacciones y de gestión de ciclo de vida del componente, gestión de estado; seguridad, multihilo y puesta en común de recursos. Los componentes especifican, en archivos externos a su código, el tipo de comportamiento que requieren del contenedor en tiempo de ejecución, y luego se basan en el recipiente para proporcionar los servicios. Esto libera al programador de la aplicación de estorbar la lógica de negocio con código para manejar el sistema y el medio ambiente.

- Sistemas de Información de la empresa Nivel: Este normalmente consiste en una o más bases de datos y aplicaciones de back-end como mainframes y otros sistemas heredados. Los componentes de negocio deben consultar e interactuar con estos almacenes de datos para procesar las solicitudes.

El núcleo de un servidor de aplicaciones es el contenedor de componente de negocio y el apoyo que proporciona para la implementación de la lógica de negocio utilizando un modelo de componente de software. A medida que los detalles varían entre las tecnologías de servidor de aplicaciones, vamos a ver en el modelo EJB utiliza ampliamente apoyada por JEE. Este es un ejemplo representativo de la tecnología de servidor de aplicaciones.

4.5.1 Enterprise JavaBeans

La arquitectura EJB define un modelo de programación estándar para la construcción de aplicaciones Java del lado del servidor. Un servidor de aplicaciones JEE compatible proporciona un contenedor EJB para gestionar la ejecución de los componentes de aplicación. En términos prácticos, el contenedor proporciona un proceso del sistema operativo (de hecho una máquina virtual Java) que aloja los componentes EJB. Figura 4.15 muestra la relación entre un servidor de aplicaciones, un recipiente y los servicios prestados. Cuando un cliente EJB invoca un componente de servidor, el contenedor asigna un hilo e invoca una instancia

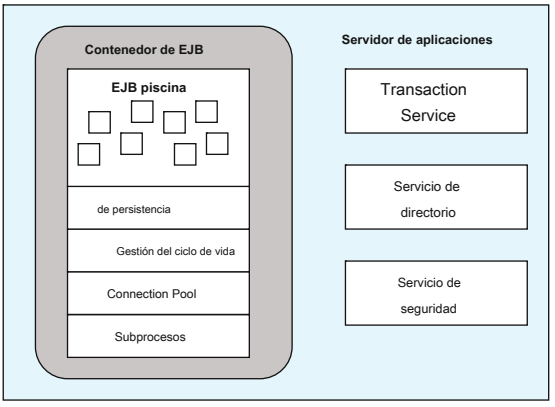


Fig. 4.15 servidor de aplicaciones JEE, contenedor EJB y servicios asociados

del componente EJB. El contenedor gestiona todos los recursos en nombre del componente y todas las interacciones entre los componentes y los sistemas externos.

4.5.2 Modelo de componentes EJB

El componente EJB modelo define la arquitectura básica de un componente EJB. Se especi fi ca la estructura de las interfaces de los componentes y los mecanismos por los que interactúa con su contenedor y con otros componentes.

La última fi cación especí EJB (parte del Java TM Platform, Enterprise Edition (Java EE) versión 5) la define dos tipos principales de componentes EJB, a saber, beans de sesión y beans controlados por mensajes. JEE anteriores especi fi caciones también de fi ne beans de entidad, pero éstos se han eliminado y sustituido por el más simple y más potente Java Persistence API [7] . Esto proporciona una facilidad de mapeo objeto / relacional para las aplicaciones Java que necesitan acceso a bases de datos relacionales desde el nivel del servidor (un requisito muy común, y uno más allá del alcance de este libro).

Los beans de sesión se utilizan normalmente para la ejecución de la lógica de negocio y de prestación de servicios para los clientes a llamar. Los beans de sesión corresponden al controlador en un modelview-controlador [8] la arquitectura, ya que encapsulan la lógica de negocio de una arquitectura a tres niveles. Los beans de sesión de fi ne una interfaz c fi aplicación específica que los clientes pueden utilizar para hacer peticiones. Los clientes envían una petición a un bean de sesión y el bloque hasta que el bean de sesión envía una respuesta.

De manera algo diferente a los beans de sesión, beans controlados por mensajes son componentes que procesan mensajes de forma asíncrona. Amessage frijol esencialmente actúa como un oyente para los mensajes que se envían desde un cliente Java Message Service (JMS). Tras la recepción de un mensaje, el frijol ejecuta su lógica de negocio y espera a que el siguiente mensaje que llega. No hay respuesta se envía al remitente del mensaje.

Además, hay dos tipos de beans de sesión, conocidos como apátrida beans de sesión y stateful beans de sesión. La diferencia entre estos se representa en la Fig. 4.16 . Un bean de sesión sin estado se define como no siendo conversacional con respecto a su proceso de llamada. Esto significa que no mantiene ninguna información de estado en nombre de cualquier cliente que llama. Un cliente obtendrá una referencia a un bean de sesión sin estado en un recipiente, y se puede utilizar esta referencia para hacer muchas llamadas en una instancia del bean. Sin embargo, entre cada invocación frijol sucesiva, un cliente no está garantizado para unirse a cualquier instancia del bean de sesión sin estado particular. El cliente delegados contenedor EJB llama a los apátridas beans de sesión en una que necesita base, por lo que el cliente nunca puede estar seguro de qué instancia de frijol que realmente van a hablar. Esto hace que sea de sentido para almacenar información relacionada con el estado del cliente en un bean de sesión sin estado. Desde la perspectiva del contenedor, todas las instancias de un bean de sesión sin estado son vistos como iguales y se pueden asignar a cualquier solicitud entrante.

---

[7] http://java.sun.com/javaee/reference/faq/persistence.jsp

[8] Ver http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

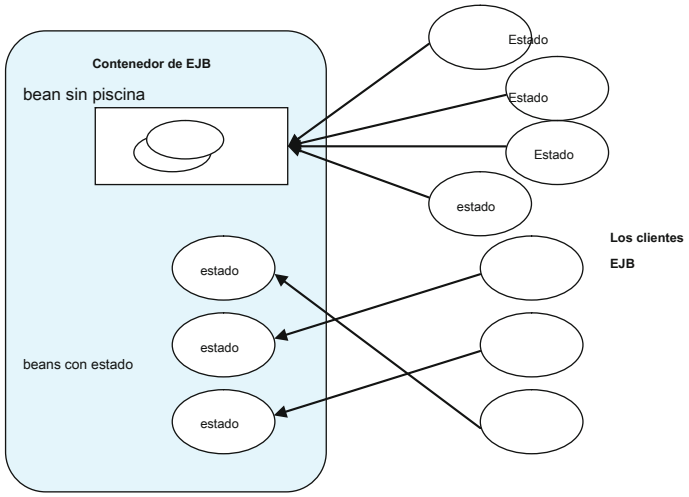


Fig. 4.16 Sin estado frente a los beans de sesión con estado

Por otro lado, se dice que un bean de sesión con estado para ser conversacional con respecto a su proceso de llamada; por lo tanto puede mantener información de estado acerca de una conversación con un cliente. Una vez que un cliente obtiene una referencia a un bean de sesión con estado, todas las llamadas posteriores al frijol utilizando esta referencia están garantizados para ir a la misma instancia del bean. El contenedor crea una nueva, dedicada bean de sesión con estado para cada cliente que crea una instancia del bean. Los clientes pueden almacenar cualquier información de estado al que desean en el grano, y pueden estar seguros de que todavía estará allí la próxima vez que acceda al frijol.

contenedores EJB asumen la responsabilidad de la gestión del ciclo de vida de los beans de sesión con estado. El contenedor escribirá el estado de un frijol en el disco si no se ha utilizado durante un tiempo, y restaurará automáticamente el estado cuando el cliente realiza una llamada posterior sobre el grano. Esto se conoce como pasivación y activación del bean con estado. Los contenedores también pueden ser con fi gurada para destruir un bean de sesión con estado y sus recursos asociados, si un grano no se utiliza durante un período especi fi cado de tiempo.

En muchos aspectos, los granos de mensajes impulsado son manejados por el contenedor EJB de una manera similar a los granos de sesión sin estado. Ellos no retienen fi c de datos de conversación de cliente-específico, y por lo tanto instancias pueden ser asignados para manejar los mensajes enviados desde cualquier cliente. beans controlados por mensajes no reciben solicitudes directamente de los clientes, sin embargo. Más bien son con fi gurada para escuchar a una cola JMS, y cuando los clientes envían mensajes a la cola, que se entregan a una instancia de un bean de mensaje para procesar.

4.5.3 Stateless bean de sesión Ejemplo de programación

Para crear un componente EJB EJB en la versión 3.0, el desarrollador debe proporcionar la clase bean de sesión y una remoto interfaz de negocio. los remoto interfaz contiene el negocio

métodos ofrecidos por el grano. Estos son de aplicación supuesto específico. A continuación se muestra un (reducir) Ejemplo de interfaz remota para un bean de sesión sin estado. Tenga en cuenta que esto es una interfaz estándar de Java que se adorna simplemente con @ Remoto anotación:

```
import javax.ejb.Remote;
@Remote
public interface Broker {
    public int newAccount(String name, String address,
        int credit)
        throws EJBException, SQLException;

    public void buyStock(int accno, int stock_id, int amount)
        throws EJBException, SQLException;

    public void updateAccount(int accno, int credit)
        throws EJBException, SQLException;
}
```

interfaz. Además, utilizando el 58

La clase de fi nición es nuevo estándar de Java, y simplemente se anota con @Apátrida. Los @ Apátrida anotación indica que esta clase es un bean de sesión sin estado, y la interfaz de negocio se utiliza para invocarlo.

```
import javax.ejb.Stateless;
@Stateless
public class BrokerBean implements Broker {
       // methods defined here …(not shown)
}
```

Acceder a un cliente EJB en EJB 3.0 es muy simple de hecho, lo que requiere el uso de la @EJB anotación, a saber:

```
@EJB BrokerBean broker;
Broker.updateAccount ( 99, 10000);
```

un bean controlado por mensajes recibir mensajes de un servidor JMS, el bean implementa el javax.jms.MessageListener clientes EJB pueden ser aplicaciones Java autónomas, servlets, applets, o incluso otros EJB. Los clientes interactúan con el grano del servidor completamente a través de los métodos de fi nido en el grano de remoto interfaz.

La historia de los beans de sesión con estado es bastante similar, utilizando el @ stateful anotación. beans de sesión con estado también deben proporcionar un método de frijol específico de inicialización para establecer el estado del frijol, y un método anotado con @ Retirar, que es llamada por los clientes para indicar que han fi nalizado con esta instancia de frijol, y el contenedor debe eliminarlo después de que el método se completa.

4.5.4 Message-Driven Programación haba Ejemplo

beans controlados por mensajes son bastante simples para desarrollar también. En el caso más común de

@MessageDriven anotación, el desarrollador especí fi ca el nombre [9] del destino desde el que el grano va a consumir mensajes.

```
import javax.jms.MessageListener;
@MessageDriven(mappedName="jms/BrokerQ")
public class BrokerMessageBean implements MessageListener {
        public void onMessage(Message msg) {
                TextMessage stockMessage =
                        (TextMessage)  msg;
                // process message
        }
}
```

4.5.5 Responsabilidades del contenedor EJB

Debe ser bastante obvio a estas alturas que el contenedor EJB es una pieza bastante compleja del software. Es, por lo tanto vale la pena cubriendo exactamente lo que el papel del contenedor está en ejecución de una aplicación EJB. En general, un recipiente proporciona componentes EJB con una serie de servicios. Estos son:

- Se proporciona una gestión del ciclo de vida de frijol y la agrupación de instancia del bean, incluyendo creación, activación, pasivación, y la destrucción de frijol.
- Se intercepta las llamadas del cliente en la interfaz remota de los granos para hacer cumplir la transacción y seguridad (véase más adelante) limitaciones. También proporciona devoluciones de llamada fi cación NotI en el inicio y el final de cada transacción que implica una instancia de bean.
- Se impone un comportamiento bean de sesión, y actúa como un detector para beans controlados por mensajes.

Con el fin de interceptar las llamadas de los clientes, las herramientas asociadas a un contenedor deben generar clases adicionales para un EJB durante el despliegue. Estas herramientas utilizan mecanismo de introspección de Java para generar dinámicamente clases para implementar el remoto las interfaces de cada grano. Estas clases permiten que el recipiente para interceptar todas las llamadas de los clientes en un grano, y hacer cumplir las políticas ed especificidad del descriptor de despliegue del bean.

El contenedor también ofrece una serie de otras características clave de tiempo de ejecución para los EJB. Estos típicamente incluyen:

- threading: EJB no deben crear explícitamente y manipular las hebras Java. Ellos debe confiar en el recipiente para asignar hilos a los granos de activos con el fin de proporcionar un entorno de ejecución concurrente, de alto rendimiento. Esto hace que los EJB más simples para escribir, como el programador de la aplicación no tiene que implementar un esquema de enhebrado para manejar las solicitudes de cliente simultáneas.
- Almacenamiento en caché: El recipiente puede mantener cachés de las instancias de frijol entidad a la que Hombre- siglos. Típicamente el tamaño de las memorias caché puede ser especificados en los descriptores de despliegue.

---

[9] Específicamente, la anotación contiene una mappedName elemento que especi fi ca el nombre JNDI de la cola JMS donde se reciben los mensajes.

- **La agrupación de conexiones: El contenedor puede gestionar un grupo de conexiones de bases de datos** para permitir el acceso ef fi ciente para gestores de recursos externos mediante la reutilización de las conexiones una vez que las transacciones se han completado.

Por último, también hay algunas características clave y muchos detalles de EJB que no han sido tratados aquí. Probablemente el más importante de ellos, ha aludido anteriormente, son:

- **Actas: Una transacción es un grupo de operaciones que deben realizarse como** una unidad, o nada en absoluto. Bases de datos proporciona la gestión de transacciones, pero cuando un nivel medio, como un contenedor EJB hace que las actualizaciones a través de múltiples bases de datos distribuidas, las cosas pueden complicarse. contenedores EJB contienen un administrador de transacciones (basado en el Java Transaction API específico de cationes), que puede usarse para coordinar las transacciones en el nivel de EJB. Sesión y frijoles controlados por mensajes pueden ser anotados con los atributos de transacción y por lo tanto controlan el commit o rollback de las transacciones de bases de datos distribuidos. Esta es una característica muy potente de EJB.

- **Seguridad: Los servicios de seguridad son proporcionados por el contenedor EJB y se pueden utilizar para** autenticar usuarios y autorizar el acceso a funciones de aplicación. En el estilo típico de EJB, la seguridad puede ser especi fi cado mediante anotaciones en la clase EJB definición, o ser implementado mediante programación. Alternativamente, la seguridad EJB puede ser especi fi externamente a la aplicación en un descriptor de despliegue de XML, y esta información es utilizada por el contenedor para anular la seguridad fi ed anotación específica.

60

4.5.6 Algunas reflexiones

En esta sección se ha dado una breve descripción de los JEE y la tecnología EJB. El modelo de componentes EJB es ampliamente utilizado y ha demostrado ser una poderosa manera de construir aplicaciones del lado del servidor. Y a pesar de las interacciones entre las diferentes partes del código son a primera un poco desalentador, con cierto grado de exposición y experiencia con el modelo, se hace relativamente fácil de construir aplicaciones EJB.

Sin embargo, mientras que la construcción código no es difícil, una serie de complejidades permanecen. Estos son:

- **El EJBmodel hace que sea posible combinar los componentes en una aplicación que utiliza** muchos diferentes patrones arquitectónicos. Esto le da al arquitecto una gama de opciones de diseño para una **aplicación. ¿Qué opción es mejor es a menudo objeto de debate, junto con lo que hace mejor significar en una** aplicación determinada? Estos no son siempre sencillas preguntas, y requiere la consideración de los complejos compromisos de diseño.
- **La forma granos interactúan con el contenedor es compleja, y puede tener un signi fi cativo** efecto del rendimiento de una aplicación. En el mismo sentido, todos los contenedores de servidor EJB no son iguales. Selección de productos y el producto específico con fi guración es un aspecto importante del ciclo de vida de desarrollo de aplicaciones.

Para las referencias discuten ambos estos problemas, consulte la sección de lectura aún más al final de este capítulo.

4.6 Resumen

Ha tomado la mejor parte de 20 años para construir, pero ahora los arquitectos de TI tienen un potente conjunto de herramientas de las tecnologías de middleware sincrónicos y asincrónicos básicos para el apalancamiento en el diseño e implementación de sus aplicaciones. Estas tecnologías han evolucionado por dos razones principales:

1. Ayudan a hacer complejo de edificios, distribuidos, aplicaciones concurrentes más simple.
2. Institucionalizar probadas prácticas de diseño, apoyándolos en las tecnologías de middleware off-the-shelf.

Con toda esta tecnología infraestructura disponible, la habilidad del arquitecto radica en cómo seleccionan, mezclar y combinar arquitecturas y tecnologías de una manera que cumpla con los requisitos y limitaciones de su aplicación. Esto requiere no sólo conocimientos avanzados de diseño, sino también un profundo conocimiento de las tecnologías implicadas, la comprensión de lo que se puede llamar de forma fiable en hacer, e igualmente importante, lo que no pueden hacerlo con sensatez. Muchas aplicaciones fallan o se entregan tarde porque perfectamente buena calidad y tecnología de middleware bien construido se utiliza de una manera en la que nunca fue pensado para ser utilizado. Esto no es culpa de la tecnología - es de los diseñadores. Por lo tanto el conocimiento de middleware, y lo más importante experiencia con las tecnologías en las aplicaciones más exigentes,

Para hacer la vida más compleja, es raro que una sola solución de la arquitectura y la tecnología tiene sentido para cualquier aplicación dada. Por ejemplo, la mensajería simple o un diseño basado en componentes EJB podría tener sentido para un problema particular. Y estas alternativas de diseño lógicas suelen tener múltiples opciones de implementación en términos de productos de middleware candidato para la construcción de la solución.

En tales situaciones, el arquitecto tiene que analizar las diversas ventajas y desventajas entre las diferentes soluciones y tecnologías, y elegir una alternativa (o tal vez nombrar a un conjunto de alternativas en competencia) que cumpla con los requisitos de la aplicación. Para ser honesto, estoy siempre un poco sospechoso de arquitectos que, en tales circunstancias, siempre vienen con la misma respuesta arquitectónica y la tecnología (a menos que trabajen para un proveedor de tecnología - en ese caso, es su trabajo).

La causa de esta “Tengo un martillo, todo es un clavo” comportamiento estilo es a menudo un ferviente creencia de que un diseño particular, y más a menudo una tecnología favorecida, pueden solucionar cualquier problema que pueda surgir. Ya que es el final del capítulo, no voy a entrar en mi caja de jabón. Pero voy a decir simplemente que los arquitectos de mente abierta, con experiencia y tecnológicamente agnósticos son más propensos a considerar una gama más amplia de alternativas de diseño. También son propensos a proponer soluciones más adecuadas a las peculiaridades y limitaciones del problema en cuestión, en lugar de promover con entusiasmo una solución particular que demuestra el eterno “bondad” de su pieza favorita de la tecnología sobre sus competidores “mal”.

4.7 Lectura adicional

Hay un enorme volumen de lectura del potencial en la materia cubierta en este capítulo. Las referencias que siguen deben darle un buen punto de partida para ahondar más profundamente.

4.7.1 CORBA

El mejor lugar para empezar para toda la información relacionada con CORBA es el sitio web del Grupo de Gestión de Objetos, a saber:

http://www.omg.org

Navegar de aquí, y te fi nd información sobre todo lo relacionado con CORBA, incluyendo especificaciones, tutoriales y muchos libros. Para recomendaciones específicas, en mi experiencia, cualquier cosa escrita por Doug Schmidt, Steve Vinosky o Michi Henning es siempre informativo y revelador.

Hablando de Michi Henning, otra tecnología muy interesante representada por el enfoque adoptado en internet Comunicaciones del motor (ICE) de ZeroC (http://zeroc.com/). El hielo es de código abierto, y hay una lista de artículos interesantes en:

http://zeroc.com/articles/index.html

Particularmente interesantes son “Un nuevo enfoque orientado a objetos Middleware” (IEEE Internet Computing, enero de 2004) y La subida y la caída de CORBA (ACM Queue, jun 2006)

4.7.2 middleware orientado a mensajes

El mejor lugar para buscar información MOM es, probablemente, la documentación y los documentos técnicos del proveedor del producto. Utilice su motor de búsqueda favorito para buscar información sobre IBM WebSphere MQ, Microsoft Message Queue (MSMQ), Sonic MQ, y muchos más. Si desea tomar conocimiento de su servicio de mensajería Java especí fi cación, se puede descargar de:

http://java.sun.com/products/jms/docs.html

Si usted está interesado en un análisis muy legible y reciente de algunas editoras suscribirse desempeño de la tecnología, incluyendo un JMS, la siguiente es bien vale la pena descargar:

Piyush Maheshwari y Michael Pang, La evaluación comparativa orientado a mensajes Medio-
cerámica: TIB / RV frente SonicMQ, Concurrencia y Computación: La práctica y la experiencia, volumen 17, páginas 1507-1526, 2005

4.7.3 Servidores de Aplicaciones

Una vez más, Internet es probablemente la mejor fuente de información general sobre los servidores de aplicaciones. productos principales incluyen WebLogic (BEA), WebSphere (IBM), servidor de aplicaciones .NET (Microsoft), y para una implementación de código abierto de alta calidad, JBoss. Hay un buen tutorial para JEE v5.0 en:

http://java.sun.com/javaee/5/docs/tutorial/doc/index.html

También hay un montón de buenos conocimientos sobre el diseño de aplicaciones EJB:

F. Marinescu. EJB patrones de diseño: Patrones Avanzados, Procesos y modismos. Wiley, 2002

D. Alur, D. Malks, J. Crupi. Patrones núcleo JEE: las mejores prácticas y estrategias de diseño. Segunda edición, Prentice Hall, 2003

Dos libros excelentes sobre las transacciones en Java, y en general, son los siguientes:

Mark Little, Jon Maron, Greg Pavlik, procesamiento de transacciones Java: Diseño y Implementación, Prentice-Hall, 2004

Philip A. Bernstein, Eric Newcomer, Principios de Procesamiento de Transacciones, Segunda Edición (la serie Morgan Kaufmann en Sistemas de Gestión de Datos), Morgan Kaufman, 2009

Los siguientes se analiza la forma de comparar las características de middleware y el servidor de aplicaciones:

I. Gorton, A. Liu, P. Brebner. La evaluación rigurosa de COTSMiddleware Tecnología. IEEE Computer, vol. 36, no. 3, páginas 50-55, marzo de 2003

Capítulo 5

Arquitecturas y tecnologías orientadas a servicios

Paul Green campo

5.1 Antecedentes

arquitecturas orientadas a servicios y servicios Web son el último paso en el desarrollo de middleware de integración de aplicaciones. Tratan de fi jar los problemas de interoperabilidad del pasado y proporcionar una base para las aplicaciones distribuidas a escala de Internet en el futuro. También intentan, y hasta cierto punto a tener éxito, para marcar el final de las guerras “middleware” con todos los principales proveedores de fi nalmente ponerse de acuerdo sobre un único rico conjunto de estándares de tecnología de integración de aplicaciones y computación distribuida.

middleware de integración de aplicaciones se utiliza para muchos fines de vincular entre sí los componentes locales para crear aplicaciones de escritorio o servidores Web simples para la construcción de las cadenas de suministro globales que abarcan Internet. Las tecnologías tradicionales en este espacio, como los servidores de aplicaciones JEE y mensajería, pueden ser excelentes soluciones para la creación de aplicaciones a partir de componentes o la integración de aplicaciones que se ejecutan dentro de la misma organización. Sin embargo, ellos están muy por debajo de lo que se necesita para vincular los procesos de negocios administrados por organizaciones independientes que están conectados a través de Internet global. servicios web y arquitecturas orientadas a servicios están diseñados para satisfacer esta necesidad solo.

En muchos sentidos, los servicios informáticos y de Web orientadas a servicios no son nada nuevo. Al igual que las tecnologías y arquitecturas de computación distribuida antes, su propósito principal es permitir que las aplicaciones invoquen funcionalidad proporcionada por otras aplicaciones, tal como JEE middleware permite a las aplicaciones de cliente Java que llaman métodos proporcionados por componentes JEE.

La diferencia real aquí es que el enfoque del modelo basado en servicios y sus tecnologías de soporte está en la interoperabilidad y la solución de los problemas prácticos que surgen debido a las diferencias en las plataformas y lenguajes de programación. Aunque es posible diseñar y construir “sistemas orientados a servicios” utilizando cualquiera de cómputo o la integración de middleware distribuido, sólo las tecnologías de servicios Web se reunirá hoy con el requisito fundamental para la interoperabilidad sin fisuras que es una parte tan importante de la visión orientada a servicios.

Este énfasis en la interoperabilidad pragmática es el resultado de la aceptación de la naturaleza diversa de las empresas de hoy en día, y darse cuenta de que esta diversidad no va a disminuir en el futuro. Soporta casi todas las organizaciones hoy en día una mezcla de plataformas,

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3_5, # Springer-Verlag Berlin Heidelberg 2011

lenguajes de programación, y paquetes de software, incluyendo aplicaciones heredadas críticos para el negocio. Cualquier propuesta de middleware de integración que asume la reescritura mayor de aplicaciones o la migración de aplicaciones a nuevas plataformas que ya están trabajando fallará en el primer obstáculo ya que los costes y los riesgos serán demasiado altos.

La realidad es que las aplicaciones empresariales a gran escala cada vez se entrelazan desde las aplicaciones, paquetes y componentes que nunca fueron diseñados para trabajar juntos e incluso pueden funcionar sobre plataformas incompatibles. Esto da lugar a una necesidad crítica de interoperabilidad, que se hace aún más importante que las organizaciones comienzan a construir una nueva generación de aplicaciones integradas de área amplia que incorporan directamente a las funciones organizadas por los socios comerciales y proveedores de servicios especializados.

servicios web y arquitecturas orientadas a servicios son la respuesta de la industria de la computación a esta necesidad de tecnologías de integración interoperables.

5.2 Sistemas orientadas a servicios

El cambio a sistemas orientados al servicio está siendo impulsado por la necesidad de integrar las aplicaciones y los sistemas de negocio que soportan. La mayoría de las tecnologías de integración existentes están cerrados o privada, y sólo admiten la integración de aplicaciones basadas en la misma tecnología, a menos que las organizaciones están dispuestas a asumir el coste de la compra o escritura compleja, código de adaptador de propósito especial. Estas restricciones sólo pueden ser aceptables dentro de una sola organización, aunque, aún así, las posibilidades de cada aplicación y cada sistema informático es compatible son bastante ligero en la realidad.

Ha habido una necesidad de integración de sistemas de negocio desde entonces se han producido sistemas de negocio. Esta integración tradicionalmente se ha manejado a través del intercambio de documentos en papel, tales como cotizaciones, facturas y órdenes. Estos documentos tradicionales todavía se utilizan hoy en día, pero ahora están casi siempre producidos por los sistemas computarizados. La tarea de la integración de estos sistemas de negocio ha cambiado poco y aunque todavía se hace comúnmente mediante el envío de estos documentos en papel por correo o fax, y luego rekeying sus datos una vez que llegan.

El ahorro de costes y deficiencias fi ciencia que vienen de deshacerse de papel e integrar directamente los sistemas de negocio basados en computadoras han sido obvio (y atractivo) desde hace muchos años, pero han demostrado ser difíciles de alcanzar para casi todo el tiempo. EDI (Intercambio Electrónico de Datos [1]) era un importante primer intento de alcanzar estos beneficios potenciales. En muchos sentidos, fue antes de tiempo y así resultó ser demasiado costoso para todos, pero los más grandes organizaciones debido a la naturaleza cerrada y privada de las redes EDI y el alto costo del software EDI propietaria.

La llegada de los servicios de Internet ha cambiado totalmente andWeb esta imagen. Internet ahora potencialmente conecta cada sistema informático en una red global,

1 http://en.wikipedia.org/wiki/Electronic_Data_Interchange

dejar que las empresas envían documentos electrónicamente a sus socios y clientes en todo el mundo, de forma rápida ya bajo costo. Los servicios Web se dirige a la otra parte del problema al proporcionar un conjunto único de normas de integración de aplicaciones que se implementan por todos los principales proveedores y se envían como parte integral de todas las plataformas de servidores. El resultado de esta evolución es que la integración a nivel de negocio pronto puede ser relativamente fácil, barato y común.

Los servicios web son realmente sólo otra tecnología de integración de aplicaciones, conceptualmente poco diferente de CORBA, JEE, DCOM, o cualquiera de sus competidores. Todas estas tecnologías son muy parecidos: las aplicaciones cliente pueden detectar servidores, encontramos a cabo los servicios que están ofreciendo, e invocar la función que brindan. Lo que es diferente acerca de los sistemas orientados al servicio y sus tecnologías de servicios supportingWeb es que estas aplicaciones y servidores ahora se espera para ser visitada por fuera de las organizaciones e individuos a través de Internet pública. El resultado de este cambio de enfoque es un conjunto de normas y principios de la arquitectura que hacen hincapié en la interoperabilidad haciendo las suposiciones acerca de cómo menor número posible de proveedores de servicios y consumidores trabajar internamente y qué detalles de implementación que tienen en común.

Figura 5.1 muestra una aplicación típica de venta a través de Internet. Los clientes ven una única aplicación integrada que les permite hacer pedidos de libros y discos,

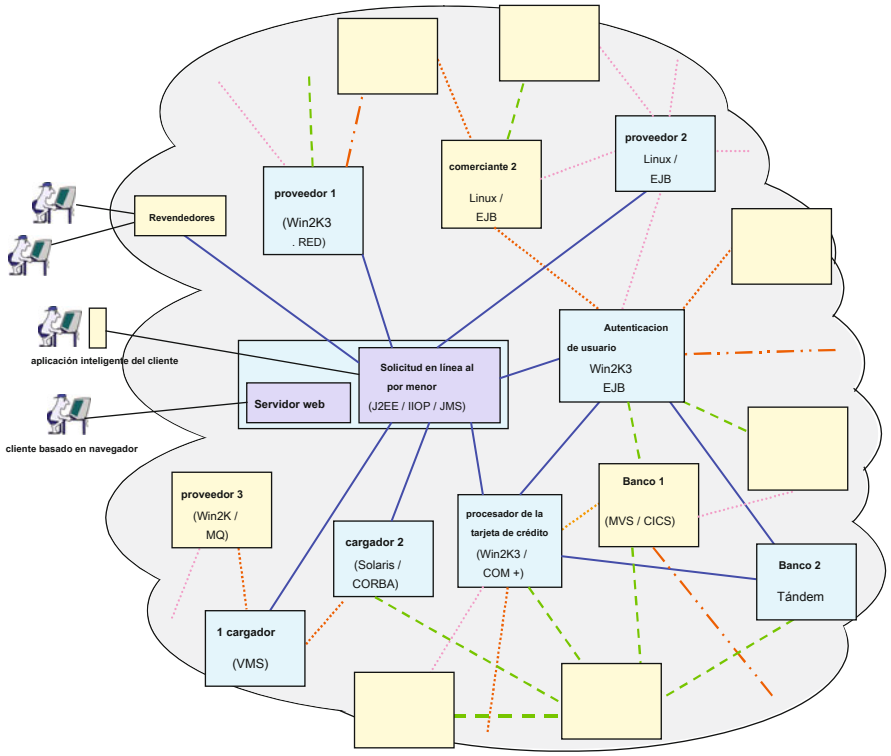


Fig. 5.1 Ejemplo-basado de servicios de aplicaciones al por menor

Realizar pagos. En realidad esta aplicación consiste en sólo un pequeño núcleo de la lógica de negocio proporcionada por el minorista aumentada por los servicios prestados por los socios de negocios, y todo se ejecuta en una mezcla diversa de plataformas y middleware. Los clientes pueden acceder a esta aplicación usando navegadores web o pueden ejecutar aplicaciones de cliente más amigables y más inteligentes que realizan llamadas directamente en los servicios de back-end proporcionados por la aplicación principal del minorista. Estos mismos servicios también pueden ser utilizados para apoyar los servicios llment ful fi orden subcontratada proporcionados a los minoristas especializados, dejando que poseen y operan sus propios ámbitos de la tienda en línea y confiar en el minorista para servicios tales como órdenes de manejo y aceptación de pagos.

Esta aplicación se puede construir utilizando cualquiera de las tecnologías de middleware discutidos en los comprendidos y utilizados con éxito por los desarrolladores remotos. 68
capítulos anteriores. El arquitecto de cualquier sistema de este tipo, sin embargo enfrentar difíciles y complejos problemas que garantizan la interoperabilidad y robustez. Estas son precisamente las áreas abordadas por arquitecturas orientadas a servicios y tecnologías de servicios Web.

Los principios fundamentales arquitecturas orientadas a servicios subyacentes no son nuevos y en gran medida sólo re fl ejar años de experiencia en la construcción de sistemas integrados a gran escala que realmente trabajado y han de pocas abstracciones y supuestos como sea posible con sus clientes. Esto hace que sea más fácil para que puedan ser mantener. Estos principios básicos que subyacen en arquitecturas orientadas a servicios se expresan a menudo como cuatro principios:

- Los límites son explícitos
- Los servicios son autónomos
- esquemas y contratos de Acciones, no implementaciones
- compatibilidad de servicio se basa en la política

servicios y en los estándares de servicios Web de soporte. Buenos servicios tienen interfaces simples y cuota como

Veamos cada uno de estos.

reducción de robustez. La respuesta a este desafío es centrarse en la simplicidad, tanto en la especi fi cación de

5.2.1 Los límites son explícitas

El primero de los principios reconoce que los servicios son aplicaciones independientes, no sólo el código que está obligado en su programa que se puede llamar a casi ningún costo. Acceder a un servicio requiere, al menos, cruzando los límites que separan los procesos, y probablemente atravesar las redes y hacer la límites para ser cruzados también en este caso, con los costos refleja en un aumento del tiempo de desarrollo y la autenticación de usuarios de varios dominios. Cada uno de estos límites (proceso, máquina, fiduciarios) que tiene que ser cruzado reduce el rendimiento, aumenta la complejidad, y aumenta las posibilidades de fracaso. Es importante destacar, que tienen que ser consciente reconocida y tratada en el proceso de diseño.

Los desarrolladores y proveedores de servicios también pueden estar separados geográficamente, por lo que hay

5.2.2 Servicios son autónomos

Los servicios son aplicaciones independientes, autónomas no clases o componentes que están estrechamente ligados a las aplicaciones cliente. Los servicios están destinados a ser desplegados en una red, muy posiblemente el Internet, donde se pueden integrar fácilmente en cualquier aplicación que fi NDS sean de utilidad. Los servicios tienen que saber nada acerca de las aplicaciones de cliente y pueden aceptar solicitudes de servicio entrantes desde cualquier lugar, con tal de que los mensajes de solicitud tienen el formato correcto y cumplir con los requisitos de seguridad específicos ed.

Los servicios se pueden implementar y administrar en su totalidad y los propietarios de estos servicios pueden cambiar sus de fi niciones, implementaciones, o requisitos en cualquier momento. compatibilidad de la versión es un problema de larga data con todos los sistemas y tecnologías distribuidas y se agrava por el carácter abierto de los servicios. ¿Cómo se puede desarrollar un servicio cuando se tiene un gran número (posiblemente desconocida) de clientes que dependen de él?

Por ejemplo, un banco ejecuta un componente de servidor que sólo es llamado por una aplicación interna cajero puede conocer la identidad y ubicación de todos los sistemas cliente, por lo que la actualización del servicio junto con todos sus llamadores es al menos técnicamente factible. Sin embargo, el procesador de tarjetas de crédito que puede aceptar solicitudes de autorización de cualquier comerciante a través de Internet no tiene manera de cualquiera de saber cómo localizar sus clientes (pasados, actuales o potenciales) o conseguir que actualicen sus aplicaciones de llamadas por variadas para que coincida con los nuevos fi niciones servicio de.

Parte de la respuesta a este problema radica en la simplicidad deliberada y extensibilidad del modelo de servicios. Todo lo que los clientes saben de un servicio de mensajes es lo que va a aceptar y volver, y esta es la única dependencia que existe entre un cliente y un servicio. Los propietarios de servicios pueden cambiar la implementación de un servicio a voluntad, con tal de que actualmente los mensajes válidos están siendo aceptadas. También pueden ampliar y evolucionar sus mensajes de solicitud de servicio y de respuesta, al igual que siempre y cuando se mantengan compatibles con versiones anteriores. Nuestro procesador de tarjetas de crédito podría cambiar totalmente cómo se implementa su servicio, tal vez pasar de CICS / COBOL a un C # /. NET, y este cambio no será visible para todos sus interlocutores, siempre y cuando no se realizan cambios incompatibles con el “autorizar mensaje de pago”.

Como los servicios son autónomos, sino que también son responsables de su propia seguridad y tienen que protegerse contra las personas que llaman posiblemente maliciosos. Sistemas desplegadas por completo en un solo sistema o en una red cerrada puede ser capaz de ignorar en gran medida la seguridad o simplemente confiar en rewalls fi o tuberías de red seguras, tales como SSL. Sin embargo, los servicios accesibles a través de Internet abierta tienen que tener una seguridad mucho más en serio.

5.2.3 Compartir esquemas y contratos, que no Implementaciones

Años de experiencia han demostrado que la construcción de sistemas integrados robustos y fiables a gran escala es difícil. Tratando de construir estos sistemas de componentes creados con diferentes modelos de programación y se ejecuta en diferentes plataformas es

mucho más difícil todavía. tecnologías orientadas a servicios abordan este problema apuntando deliberadamente por simplicidad tanto como sea posible. Los servicios no son objetos remotos con herencia, métodos, y el comportamiento en tiempo de ejecución compleja como en CORBA, ni son componentes que soportan eventos, propiedades y llamadas a métodos con estado. Los servicios son sólo las aplicaciones que reciben y envían mensajes. Clientes y servicios comparten nada más que las de fi niciones de estos mensajes y ciertamente no comparten código de método o entornos de tiempo de ejecución complejas.

diferente 70

Todo lo que una aplicación necesita saber acerca de un servicio es su contrato: la estructura (esquema) de los mensajes está dispuesta a aceptar y vuelta, y si tienen que enviarse en un orden particular. Las aplicaciones cliente pueden utilizar este tipo de contrato para construir los mensajes de petición para enviar a un servicio, y los servicios pueden utilizar sus esquemas para validar los mensajes entrantes y asegurarse de que tienen el formato correcto.

5.2.4 Compatibilidad con servicio se basa en la Política

cargadores que ofrecen exactamente los mismos servicios y utilizar los mismos esquemas de mensajes, pero tienen

Los clientes tienen que ser completamente compatible con los servicios que deseen utilizar. Compatibilidad significa no sólo que los clientes están siguiendo los formatos de mensaje fi cados y los patrones de cambio, sino también que se cumplan otros requisitos importantes, tales como si los mensajes deben ser encriptados o la necesidad de realizar un seguimiento para asegurar que ninguno se han perdido en tránsito. En el modelo orientado a servicios, estos requisitos no funcionales son de fi ne mediante políticas, y no sólo por escrito como parte de la documentación de un servicio.

estandaricen y ofrecidos por los proveedores de la competencia. Por ejemplo, nuestro minorista en línea puede utilizar dos

Por ejemplo, nuestro procesador de tarjetas de crédito puede decidir que todos los comerciantes que envíen solicitudes de autorización de pago deberán acreditar su identidad mediante tokens de autenticación basados en X.509. Esta restricción de seguridad se puede representar simplemente como una declaración en la política de seguridad publicada por el servicio de autorización.

Las políticas son colecciones de declaraciones legibles por máquinas que permiten un servicio de fi ne sus requisitos para cosas como la seguridad y la fiabilidad. Estas políticas pueden ser incluidos como parte del contrato de un servicio, lo que le permite especificar por completo el comportamiento y las expectativas de un servicio, o pueden ser mantenidos en las tiendas políticas separadas y fue a buscar dinámicamente en tiempo de ejecución.

para satisfacer las necesidades de un proveedor de servicio en particular. Esto será cada vez más útil como los servicios se

políticas basadas en contratos pueden ser considerados como sólo una parte de la documentación de un servicio, sino que también se pueden utilizar las herramientas de desarrollo para generar automáticamente el código compatible para los clientes y servicios. Por ejemplo, una política de seguridad del lado del servidor se puede utilizar para generar código que se compruebe que las piezas necesarias de un mensaje entrante se cifran y luego descifrar estos datos, presentándola como texto sin formato a la aplicación de servicio. Todo esto se hace sin ningún esfuerzo de codificación del desarrollador.

La separación de las políticas de los contratos también permite a las aplicaciones cliente de forma dinámica se adaptan

requisitos de autenticación. El uso de políticas dinámicas permite a nuestros desarrolladores a escribir una sola aplicación que es compatible con los métodos de autenticación y selecciona dinámicamente cuál usar por ir a buscar la política del servicio de destino antes de construir y enviar cualquier solicitud de entrega.

5.3 Servicios Web

Los servicios web son un conjunto de estándares de tecnología de integración que se han diseñado específicamente para satisfacer las necesidades derivadas de las arquitecturas y sistemas orientados a los servicios. En muchos sentidos, los servicios Web no son realmente muy diferentes de las tecnologías de middleware existentes, pero sí que difieren en su enfoque en la simplicidad y la interoperabilidad. La característica más importante que ofrece servicios web es que todos los principales proveedores de software han puesto de acuerdo para apoyarlos. Interoperabilidad todavía no es, por supuesto, garantiza que sea indoloro, pero al menos los problemas encontrados serán los errores y malas interpretaciones de las normas comunes, no introducen intencionadamente incompatibilidades entre tecnologías patentadas similares pero diferentes.

Todas las tecnologías de integración de aplicaciones, incluyendo servicios Web, en realidad sólo proporcionan cuatro funciones básicas que permiten a los desarrolladores (y programas), haga lo siguiente:

- Encuentra los servicios adecuados (usando UDDI o otro directorio)
- Averiguar acerca de un servicio (utilizando WSDL)
- Pedir un servicio para hacer algo (utilizando SOAP)
- Hacer uso de los servicios como la seguridad (utilizando estándares WS- *)

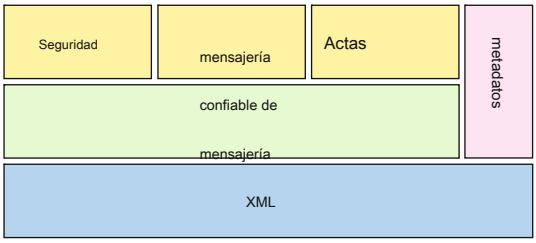
SOAP, WSDL y UDDI fueron los estándares de servicios Web primeras para ser publicados, pero que sólo cumplen con los requisitos más básicos para la integración de aplicaciones. Carecen de apoyo a la seguridad, transacciones, fiabilidad, y muchas otras funciones importantes. Esta brecha está siendo progresivamente llenada por una serie de normas (comúnmente llamado “WS *”) primero esbozó por IBM y Microsoft en un taller del W3C en 2001. La tarea de crear estas normas adicionales y llegar a un acuerdo de toda la industria es un confuso, trabajo en progreso, con especificaciones en diferentes grados de madurez y apoyado por diversos organismos de normalización. Algunas especificaciones se complementan, se superponen y compiten entre sí. En la actualidad hay implementaciones sin embargo listos para la producción disponibles para muchos de ellos. Ver http://www.w3.org/2002/ws/ para algunas ideas sobre estas especificaciones.

Los servicios web son estándares XML. Los servicios se definen usando XML y aplicaciones solicitan servicios mediante el envío de mensajes XML y los estándares de servicios Web hacen un amplio uso de otras normas XML existentes siempre que sea posible. Existen varios estándares de servicios Web y estos se pueden organizar en las categorías que se muestran en la Fig. 5.2 .

Esta serie de normas puede sugerir la complejidad en lugar de la simplicidad deseada, y en muchas aplicaciones, sólo unas pocas normas fundamentales están realmente en uso. También hay cada vez más buena herramienta y una biblioteca / soporte de marco para estas normas, por lo que los desarrolladores sólo tienen que entender las capacidades ofrecidas en lugar de

estándares Web 72

Fig. 5.2 Resumen de los servicios



la sintaxis XML detallada. Para ilustrar esto antes de mostrar las complejidades del XML asociado, a continuación es un simple servicio web de fi nición con la API de Java para servicios web XML (JAX-WS), que forma parte de la plataforma JEE. El uso de anotaciones en la forma utilizada para EJB, la creación de un servicio web es muy simple.

```
brokerservice.endpoint paquete;

javax.jws.WebService importación;

@Servicio web
Broker public class {

        @WebMethod
        Cadena viewStock pública (String nombre) {
                // código omite}}
```

Por lo tanto, los juegos de herramientas como JAX-WS, el desarrollador de servicios no necesita crear o entender los mensajes XML con formato SOAP. El sistema de tiempo de ejecución JAX-WS simplemente convierte las llamadas a la API y las respuestas desde y hacia formatos de mensaje de SOAP subyacentes. Usted será capaz de juzgar por sí mismo en una página o dos, si esto es una buena cosa!

Uno de los principios que subyacen a la simplificación de los servicios Web es que los diversos campos de mensajes fi y atributos utilizados para apoyar las funciones tales como la seguridad y la fiabilidad son totalmente independientes entre sí. Las solicitudes sólo necesitan incluir sólo aquellos pocos campos y atributos necesarios para sus propósitos especí fi cos y pueden ignorar todas las otras normas. Por ejemplo, una solicitud SOAP podría identificar al solicitante de un servicio mediante la inclusión de un nombre de usuario y contraseña en la forma especificada en el WSSecurity UsernameToken per fi l. Este / información relacionada con la contraseña de usuario es el único elemento de cabecera en materia de seguridad incluido en el mensaje. WS-Security es compatible con otras formas de autenticación de usuario, así como de cifrado y firmas digitales, pero como estos no son utilizados por el servicio, que no aparecen en absoluto en la solicitud de mensaje SOAP.

Otro de los objetivos de los estándares de servicios Web es proporcionar un buen soporte para arquitecturas de sistemas que hacen uso de “intermediarios”. En lugar de asumir que los clientes siempre envían peticiones directamente a los proveedores de servicios, el modelo intermediario asume que pasar estos mensajes pueden (transparente) a lo largo de una cadena de otras aplicaciones en su camino hacia su destino fi nal. Estos intermediarios pueden hacer nada con los mensajes que reciben, incluyendo enrutamiento de ellos, el registro, el control de seguridad, o incluso

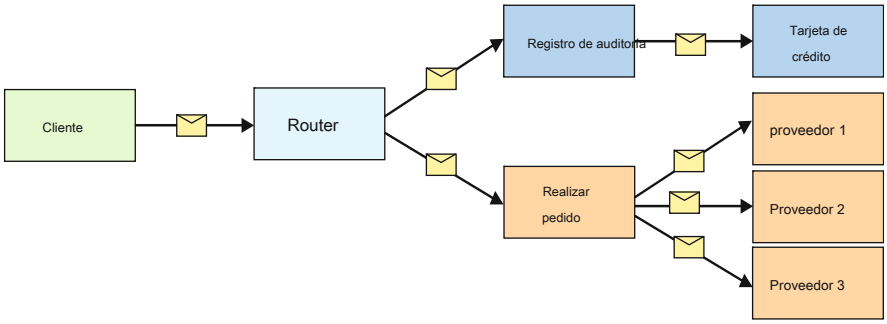


Fig. 5.3 secuencia simple intermediario

adición o sustracción de bits de contenido del mensaje. Este modelo se muestra en la Fig. 5.3 , Donde los intermediarios están proporcionando servicios de enrutamiento y auditoría.

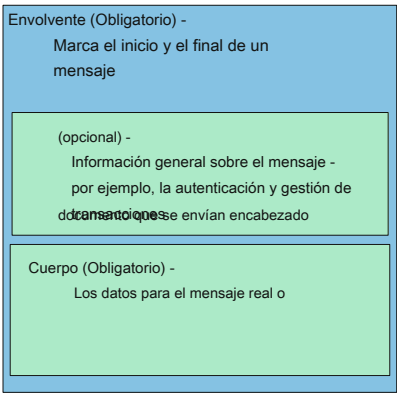
Los servicios Web proporcionan soporte para arquitecturas basadas en intermediarias en un número de maneras. Estos incluyen el etiquetado de elementos de la cabecera con el papel de su destinatario y apoyar el principio “extremo a extremo” para funciones tales como la seguridad, por lo que asegurar que continúan funcionando incluso si los mensajes pasan a través de intermediarios en lugar de viajar directamente desde el cliente al servicio de . Por ejemplo, en la aplicación mostrada en la Fig. 5.3 , El cliente puede utilizar los mecanismos previstos por WS-Security para proteger información confidencial destinado únicamente para la aplicación de tarjeta de crédito, ocultándola de router que el mensaje debe pasar a través de su viaje.

5.4 SOAP y Mensajería

Jabón era el estándar de servicios Web original y sigue siendo el más importante y más ampliamente utilizado. Se especi fi ca un simple pero extensible protocolo de comunicación aplicación toapplication basado en XML, más o menos equivalente a la RPC del DCE o RMI de Java, pero mucho menos complejo y mucho más fácil de implementar como consecuencia de ello. Esta simplicidad proviene de permanecer deliberadamente bien lejos de problemas complejos, tales como la recogida de basura distribuida y los objetos que pasan por referencia. Todo lo que hace el estándar SOAP es de fi ne un protocolo simple pero extensible orientado a mensajes para invocar servicios remotos, utilizando HTTP, SMTP, UDP, u otros protocolos como la capa de transporte y XML para los datos de formato.

mensajes SOAP tienen la estructura simple como se muestra en la Fig. 5.4 . La cabecera tiene información acerca de la carga útil del mensaje, posiblemente incluyendo elementos tales como señales de seguridad y contextos de transacción. El cuerpo tiene el contenido real del mensaje que se pasa entre las aplicaciones. El estándar de SOAP no obliga a lo que puede ir en un encabezado del mensaje, dando SOAP de su extensibilidad como nuevos estándares, tales como WS-Security, puede ser especi fi cado con sólo que define los elementos de la nueva cabecera, y sin requerir cambios en la propia norma SOAP.

Fig. 5.4 estructura de mensaje de SOAP



JABÓN estaba parado originalmente para Simple Object Access Protocol pero es ahora o fi cialmente ya no es un acrónimo, sólo una palabra, y ciertamente nada que ver con el acceso a objetos remotos! clientes SOAP envían mensajes de solicitud de XML para los proveedores de servicios a través de cualquier medio de transporte y se puede obtener mensajes de respuesta XML a cambio. Un mensaje SOAP pidiendo una cita de stock se muestra en la Fig. 5.5 . Esto corresponde a la WSDL definición se muestra en la Fig. 5.6 . La solicitud lleva un nombre de usuario y contraseña con algoritmo hash en la cabecera para permitir que el servicio sabe que está haciendo la solicitud.

Hay una serie de otras normas incluidas en la categoría de mensajería de servicios web, incluyendo WS-Addressing y WS-Eventing. WS-Addressing existe porque los servicios web realmente tienen poco que ver con la Web y no dependen exclusivamente de HTTP como una capa de transporte. Los mensajes SOAP se pueden enviar a través de cualquier protocolo de transporte, incluyendo TCP / IP, UDP, dirección de correo (SMTP) y las colas de mensajes, y WS-Addressing proporciona mecanismos de transporte para direccionar los servicios e identificar mensajes. WS-Eventing proporciona soporte para un modelo de publicación-suscripción por de fi nir el formato de los mensajes de solicitud de suscripción que los clientes envían a los editores. mensajes publicados que cumplen la expresión fi ltrado siempre se envían a las personas que llaman utilizando mensajes SOAP normales.

5.5 UDDI, WSDL y Metadatos

Hay un fuerte tema de metadatos y política que atraviesa los estándares de servicios Web. servicios SOAP se describen normalmente utilizando WSDL (Web Services Description Language) y pueden ser localizados mediante la búsqueda de un UDDI (Universal Description, Discovery and Integration) guía. Los servicios pueden describir sus requisitos para cosas como la seguridad y la fiabilidad mediante declaraciones de política, definida utilizando el marco de WS-Policy, y las normas de política especializados, tales como WS-SecurityPolicy. Estas políticas se pueden unir a aWSDL servicio de definición o se mantiene en tiendas de política separados y recuperados utilizando WS-MetadataExchange.

```
<? Xml version = "1.0" encoding = "UTF-8"?> <Soap: Envelopexmlns: Jabón
=
        "Http://www.w3.org/2003/05/soap-envelope" xmlns: xsi =
"http://www.w3.org/2001/XMLSchema-instance" xmlns: xsd = "http: // www .w3.org / 2001 /
XMLSchema"
xmlns: WSA = "http://schemas.xmlsoap.org/ws/2004/03/addressing" xmlns: wsse =
"http://docs.oasis-open.org/wss/2004/01/oasis-
        200401-WSS-WSSecurity-secext-1.0.xsd "xmlns: wsu ="
http://docs.oasis-open.org/wss/2004/01/oasis
        -  200401-WSS-WSSecurity-utilidad-1.0.xsd "> <soap: Header> <wsa:
Acción>


        http://myCompany.com/getLastTradePrice </ wsa: Acción>
            <Wsa: MessageID> uuid: 4ec3a973-a86d-4fc9-BBC4-ade31d0370dc </ wsa: MessageID>

            <Wsse: Jabón de seguridad: mustUnderstand = "1"
                        <Wsse: UsernameToken>
                                    <Wsse: Nombre de usuario> NNK </ wsse: Nombre de usuario> <wsse: PasswordType
                                    = "http: //docs.oasis- open.org/wss/2004/01/oasis-200401-wss-username

                                                -  token-perfil-1.0 # PasswordDigest ">
                                    weYI3nXd8LjMNVksCKFV8t3rgHh3Rw == </ wsse: Contraseña>

                                    <Wsse: Nonce> WScqanjCEAC4mQoBE07sAQ == </ wsse: Nonce> <wsu: Creado>
              2003-07-16T01: 24: 32Z </ WSU: Creado> </ wsse: UsernameToken> </ wsse: Seguridad> </ jabón: header> <soap:
              Body>



            <M: GetLastTradePrice
            xmlns: m = "http://myCompany.com/stockServices">
                        <Símbolo> DIS </ símbolo> </ m:
      GetLastTradePrice> </ soap: Body>


    </ Soap: Envelope>
```

Fig. 5.5 SOAP de muestra de mensaje

UDDI ha demostrado ser el menos utilizado hasta ahora de los tres estándares originales de servicios Web. En muchos sentidos, ya sea UDDI es el menos interesante o potencialmente más interesante de estas normas, en función de la importancia que le parece ser capaz de descubrir de forma dinámica y enlace a los servicios es a su aplicación. Las organizaciones están desarrollando sistemas de grandes complejos de servicios Web hoy en día sin el uso de directorios UDDI globales, el uso de otros métodos de servicios hallazgo como el contacto personal o listas publicadas de los servicios en los sitios Web. Todo esto podría cambiar en el futuro, sobre todo cuando las asociaciones del sector comienzan a liberar de servicios de fi niciones comunes y necesitan publicar directorios de proveedores de servicios cuali fi.

WSDL se utiliza para describir servicios Web, incluyendo sus interfaces, métodos y parámetros. La descripción WSDL de un servicio llamado StockQuoteService que proporciona una única operación denominada GetLastTradePric se representa en la Fig. 5.31.

```
<? Xml version = "1.0"?>
<Definiciones name = "StockQuote"
        targetNamespace = "http://myCompany.com/stockquote.wsdl" xmlns: TNS =
        "http://myCompany.com/stockquote.wsdl" xmlns: Jabón =”http://schemas.xmlsoap.org/wsdl/soap /”xmlns:
        xsd =" http://www.w3.org/2001/XMLSchema”xmlns = "http://schemas.xmlsoap.org/wsdl/"> <nombre de
        mensaje = "GetLastTradePrice">


                <= Tipo de parte del nombre de "cuerpo" = "xsd: string" /> </ message>

        <Nombre de mensaje = "LastTradePrice">
                <Parte name = tipo "cuerpo" = "xsd: float" /> </ message>


        <Nombre portType = "StockQuotePortType">
                <Nombre de la operación = "GetLastTradePrice">
                        <entrada de mensaje = "TNS: GetLastTradePrice" /> <salida de mensaje =
                "TNS: LastTradePrice" /> </ operación> </ portType>



        <Binding name = "StockQuoteBinding"
                                        type = "tns: StockQuotePortType">
                <Soap: estilo de encuadernación = "documento"

transporte = "http://schemas.xmlsoap.org/soap/http" />
                <Nombre de la operación = "GetLastTradePrice">
                        <Soap: operación soapAction =
                                        "Http://myCompany.com/GetLastTradePrice" />
                        <Input>
                                <Soap: uso del cuerpo = "literal" /> </ input>
                        <salida>

                                <Soap: uso del cuerpo = "literal" /> </ salida> </
                operación> </ binding>



<Service name = "StockQuoteService">
    <Documentation> servicio de la cita </ documentación> <nombre del puerto = "StockQuotePort"

                vinculante = "tns: StockQuoteBinding"> <soap: ubicación de la
            dirección =
                        "Http://myCompany.com/stockServices" />
    </ Puerto> </ service>
</ definiciones>
```

Fig. 5.6 WSDL para el GetLastTradePrice servicio 76

Esta operación tiene un parámetro símbolo de tipo cuerda que los nombres de la población de interés y devuelve una flote que mantiene el precio más recientemente comercializado.

WSDL está bien apoyado por los entornos de desarrollo como Visual Studio, Eclipse, y WebSphere. Estas herramientas pueden generar WSDL automáticamente del método del programa y la interfaz de Definiciones, y se toman en WSDL del servicio Definiciones

y hacer más fácil para los desarrolladores escribir código que llama a estos servicios. Uno de los efectos secundarios adversos de esta herramienta de apoyo es que tiende a animar a los desarrolladores a pensar en los servicios como los métodos remotos, en lugar de mover al modelo basado en mensajes preferible y más rica proporcionada por los servicios Web.

5.6 Seguridad, Operaciones y Fiabilidad

Uno de los problemas que enfrenta la mayoría de los protocolos de middleware es que no funcionan bien en Internet abierta debido a las barreras de conectividad impuestas por rewalls fi. La mayoría de las organizaciones no quiere que extraños tengan acceso a los protocolos y tecnologías que utilizan internamente para la integración de aplicaciones y así bloquean los puertos TCP / IP necesarios en sus rewalls fi perimetrales.

La respuesta común a este problema la tecnología, y la adoptada por los servicios web, ha sido cooptar el protocolo Web, HTTP, como una capa de transporte debido a su capacidad de pasar a través de la mayoría rewalls fi. Este uso de HTTP es conveniente, sino que también crea problemas de seguridad potenciales como HTTP trá fi co ya no es sólo ir a buscar inocua páginas Web. En su lugar, puede estar haciendo llamadas directas sobre las aplicaciones internas.

WS-Security y sus estándares asociados abordan estos problemas proporcionando fuertes mecanismos criptográficos para identificar a los llamantes (autenticación), proteger el contenido frente a intrusos (cifrado), y garantizar la integridad de la información (firmas digitales). Estos estándares están diseñados para ser extensible, dejando que se pueden adaptar fácilmente a las nuevas tecnologías de seguridad y algoritmos, y también la integración de soporte con las tecnologías de seguridad legado.

WS-Security soporta arquitecturas de aplicaciones a base de intermediario al permitir que múltiples elementos de la cabecera de seguridad, cada uno marcado con el papel de su receptor previsto a lo largo de la cadena de procesamiento, y mediante el apoyo de cifrado parcial y firmas parciales. Como una ilustración, en el ejemplo mostrado en la Fig. 5.3 , Los datos de su tarjeta de crédito sensibles pueden ocultarse mediante la encriptación de ellos, dejando el resto del mensaje sin cifrar de forma que pueda ser leído por la aplicación de enrutamiento.

El conjunto final de estándares de servicios web de soporte de transacciones y mensajería confiable. Hay dos tipos de transacciones de servicios Web compatibles con las normas. WS-AtomicTransactions soporta transacciones ACID distribuida convencionales y asume los niveles de confianza y de respuesta rápidos tiempos que hacen de este estándar es adecuada sólo para tareas de integración de aplicaciones internas e inutilizable para fines de integración de aplicaciones a escala de Internet. WS-BusinessActivity es un marco y un conjunto de elementos de protocolo de la coordinación de la terminación de aplicaciones integradas de forma flexible. Proporciona una cierta ayuda para atomicidad invocando compensadores cuando un distribuyeron acabados aplicación FI en fracaso.

El soporte para mensajería confiable en servicios web, simplemente se asegura de que todos los mensajes enviados entre dos aplicaciones realmente lleguen a su destino en el orden en que fueron enviados. WS-Reliable Messaging no garantiza la entrega en el

caso de fallo, a diferencia de la cola middleware de mensajería mediante colas persistentes. Sin embargo, todavía es un estándar útil ya que proporciona como máximo una vez, la entrega de mensajes en orden sobre cualquier capa de transporte, incluso los no fiables tales como UDP o SMTP.

5.7 Servicios Web REST

La “web” en los “servicios Web basados en SOAP” es en realidad un nombre inapropiado como jabón tiene nada que ver con la web, aparte de su uso (opcional) del protocolo Web, HTTP, como una capa de transporte “cortafuegos ambiente fi”. Tal vez como reacción a este mal uso de la palabra “Web” (y falta total de jabón de la adhesión a las filosofías subyacentes a la “Web”), algunos adherentes a la Web-forma-de-hacer-cosas han desarrollado y evangelizada vigorosamente una forma alternativa de hacer los servicios Web: REST (Representational State Transfer).

Los servicios Web RESTful se basan en HTTP como su fi cientemente rica protocolo para satisfacer por completo las necesidades de las aplicaciones de servicios Web. En el modelo de reposo, el HTTP GET, POST, PUT y DELETE verbos se usan para transferir datos (a menudo en forma de documentos XML) entre el cliente y servicios. Estos documentos son “representaciones” de “recursos” que son identificados por los URI (Uniform Web normales ERS recursos identi fi). Este uso de las tecnologías web estándar HTTP y significa que los servicios web RESTful pueden aprovechar la infraestructura Web completo, como el almacenamiento en caché y la indexación.

El siguiente ejemplo muestra como un simple servicio web de la base de datos del cliente se podría implementar usando un enfoque reparador. En este ejemplo, la base de datos del cliente es un “recurso” y los registros de clientes individuales también son “recursos” en su propio derecho. Cada uno de estos recursos tiene un URI único que se puede utilizar como el sujeto de un verbo HTTP.

- el URI http://example.com/customers identi fi ca el recurso de la base de datos de clientes.

  GET solicitudes enviadas a este URI devuelven el conjunto de todos los clientes como un documento XML que contiene solo una lista de URIs que apuntan a los recursos individuales de los clientes.
- El URI para cada cliente en la base de datos se forma añadiendo el custo-

  ID único del mer al cliente establece URI. Por ejemplo, http://example.com/ clientes / 1 identi fi ca el recurso correspondiente al cliente con ID 1.
- Una petición GET enviada a uno de estos URIs únicas del cliente recupera un archivo XML

  documento que contiene una representación del estado actual del cliente correspondiente.
- los recursos de clientes existentes pueden actualizarse por poner un documento XML

  que contiene una representación del nuevo estado deseado de la cliente a la URI cliente apropiado.
- Los clientes nuevos pueden ser añadidos a la base de datos mediante la publicación de documentos XML

  que contiene la representación del nuevo recurso a la URI de colección. Los URIs para los nuevos recursos del cliente se devuelven mediante el encabezado HTTP ubicación en las respuestas del servidor.
- asientos de clientes se pueden eliminar mediante el envío de una petición de borrado al cliente

  URI.

Algunos de los defensores más entusiastas del enfoque REST a los servicios Web se ven como una competencia con las tecnologías basadas en SOAP y sus proveedores. Muchos de los argumentos de los defensores REST provienen de su creencia de que el descanso es “simple” y SOAP y WS- * son “complejas”. Por supuesto, la “simplicidad” y “complejidad” son en relación con los problemas arquitectónicos y técnicos que están tratando de resolver, y el amplio conjunto de servicios proporcionados por el WS * estándares bien puede ser justo lo que necesita para resolver los complejos problemas que se enfrentar en sus aplicaciones empresariales distribuidas. Por el contrario, el enfoque REST para la construcción de servicios Web será adecuado para muchos problemas sencillos, especialmente donde las cuestiones de seguridad robusta, la fiabilidad y la interoperabilidad no son importantes. Si estos temas “complejos” son importantes en la arquitectura de integración de aplicaciones, a continuación, SOAP y WS- * bien puede ser una respuesta mejor, ofreciendo soluciones basadas en estándares e interoperables a estos requisitos no funcionales inherentemente complejos. La elección es suya como SOAP y REST son realmente enfoques complementarios para la implementación de servicios Web, cada uno mejor se adapte a diferentes tipos de aplicaciones distribuidas.

5.8 Conclusión y lectura adicional

Servicios y arquitecturas orientadas a servicios son respuestas pragmáticas a los problemas de complejidad y de interoperabilidad encontradas por los constructores de las generaciones anteriores de las aplicaciones integradas a gran escala. Los servicios web son un conjunto de normas de tecnología de integración que re fl ejan esta necesidad de simplicidad y la interoperabilidad.

Lo que “realmente” la transformación de los servicios Web es que no es (más o menos) sólo un conjunto de normas comunes que todo el mundo utiliza al ofrecer o acceder a los servicios. Estas normas están siendo apoyados por toda la industria de la computación y están disponibles en todas las plataformas de aplicaciones a bajo costo. El carácter generalizado de los servicios Web hace atractivos a utilizar para la integración de aplicaciones, sin duda para aplicaciones a gran escala de plataforma cruzada y, en muchos casos, para las tareas de integración local también.

arquitecturas orientadas a servicios y servicios Web son temas candentes en la industria de TI de hoy en día. Todos los principales fabricantes de software están publicando tutoriales y artículos sobre los servicios y la forma en que son apoyados por sus productos. Hay unos cuantos buenos libros bastantes por ahí y cualquier número de artículos de revistas así. lugares de partida son buenas MSDN de Microsoft, DeveloperWorks de IBM, y los sitios Web de desarrolladores de Sun, en los siguientes lugares:

http://www.msdn.microsoft.com
http://www.ibm.com/developerworks
http://www.developers.sun.com/

También podrá hallar más información sobre los servicios Web y SOA a través de Google que se preocupa de imaginar. O simplemente ir a su propio proveedor de software y ver lo que tienen que decir acerca de la forma en que dan soporte a servicios.

Algunos excelentes libros de texto de servicios Web están alrededor. Los siguientes son tres ejemplos me gustaría recomendar:

Thomas Erl, Patrones de Diseño SOA, Prentice-Hall, 2009 Thomas Erl, los principios de SOA de servicio de diseño, Prentice-Hall, 2007

O. Zimmermann, M. R Tomlinson, S. Peuser, Perspectivas sobre Servicios Web de aplicar el jabón, WSDL y UDDI para Proyectos Mundo Real. Springer-Verlag 2004

G. Alonso, F. Casati, H. Kuno, V. Machiraju, Conceptos de servicios web, arquitecturas y Aplicaciones. Springer-Verlag 2004

S. Chatterjee, J. Webber, El desarrollo de servicios Web de empresa: un arquitecto de Guía. Prentice-Hall, 2004

También hay un montón de material de lectura para mantenerlo ocupado en los servicios web RESTful. Por ejemplo:

Jim Webber, Savas Parastatidis, Ian Robinson, REST en la práctica, O'Reilly Media, 2010

http://java.sun.com/developer/technicalArticles/WebServices/restful/

Leonard Richardson, Sam Ruby, Servicios Web RESTful, O'Reilly Media, 2007

http://www.ibm.com/developerworks/webservices/library/ws-restful/

http://www.xfront.com/REST-Web-Services.html

Por último, el blog de Steve Vinoski siempre es una lectura entretenida y educativa en REST - ver http://steve.vinoski.n

Capítulo 6
Tecnologías avanzadas de middleware

6.1 Introducción

Los tres capítulos anteriores han descrito los bloques de construcción básicos de middleware que se pueden utilizar para implementar arquitecturas de sistemas distribuidos para sistemas empresariales a gran escala. A veces, sin embargo, estos bloques de construcción no son su fi ciente para permitir a los desarrolladores diseñar y construir arquitecturas complejas con facilidad. En tales casos, se necesitan herramientas más avanzadas y diseños, que permiten hacer frente a problemas de arquitectura con tecnologías de middleware más potentes. En este capítulo se describen dos de estos, es decir, intermediarios de mensajes y motores de flujo de trabajo, y analiza las fortalezas y debilidades de estos enfoques.

6.2 intermediarios de mensajes

mensajería básica utilizando MOM y publicación-suscripción oficinas tecnologías suf para muchas aplicaciones. Es una forma simple y efectiva y probada que puede ofrecer altos niveles de rendimiento y fiabilidad.

despliegues de MOM empiezan a ser un poco más complejo, aunque cuando los formatos de mensajes no están totalmente de acuerdo entre las distintas aplicaciones que se comunican utilizando el MOM. Este problema se produce comúnmente en el campo de la integración de la empresa, donde el problema de fondo es la construcción de aplicaciones de negocio de sistemas grandes y complejos comerciales legado que nunca fueron diseñados para trabajar en conjunto y el intercambio de información.

integración de la empresa es todo un campo de estudio en sí mismo (ver lecturas adicionales). Desde la perspectiva de este libro, sin embargo, la integración empresarial ha dado lugar a una clase interesante y ampliamente utilizado de tecnologías middleware, conocidos como intermediarios de mensajes.

Vamos a introducir intermediarios de mensajes a modo de ejemplo motivador. Suponga que una organización tiene cuatro sistemas de negocios diferentes legado que cada contienen información

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3_6, # Springer-Verlag Berlin Heidelberg 2011

acerca de los clientes. [1] Cada uno de estos cuatro tiendas algunos datos comunes acerca de los clientes, así como algunos campos de datos únicos que otros no mantienen. Además, cada una de las aplicaciones tiene un formato diferente para un registro de cliente, y los nombres de campo individuales son diferentes a través de cada uno (por ejemplo, uno utiliza la dirección, otra ubicación, como un nombre de campo de datos de direcciones de clientes). Para actualizar los datos del cliente, un API propietario está disponible para cada sistema heredado.

Si bien esto es conceptualmente muy simple, es un problema que muchas organizaciones tienen. Por lo tanto, vamos a suponer mantener los datos consistentes en cada una de estas cuatro aplicaciones es un problema para nuestra organización hipotética. Por lo tanto, deciden poner en práctica un sitio web que permite a los clientes actualizar sus propios datos en línea. Cuando esto ocurre, los datos introducidos en la página web se pasa a un componente web en el servidor web (por ejemplo, un servlet o una página ASP.NET). La función de este componente es pasar los datos actualizados a cada una de las cuatro aplicaciones heredadas, para que puedan actualizar sus propios datos de los clientes correctamente.

La organización utiliza MOM para la comunicación entre aplicaciones. En consecuencia, el componente Web da formato a un mensaje con los nuevos datos del cliente y utiliza el MOM para enviar el mensaje a cada sistema heredado. [2] El formato del mensaje, con la etiqueta En formato en la Fig. 6.1 , Es un formato acordado que el componente web y todas las aplicaciones heredadas entienden.

cada mensaje. 82

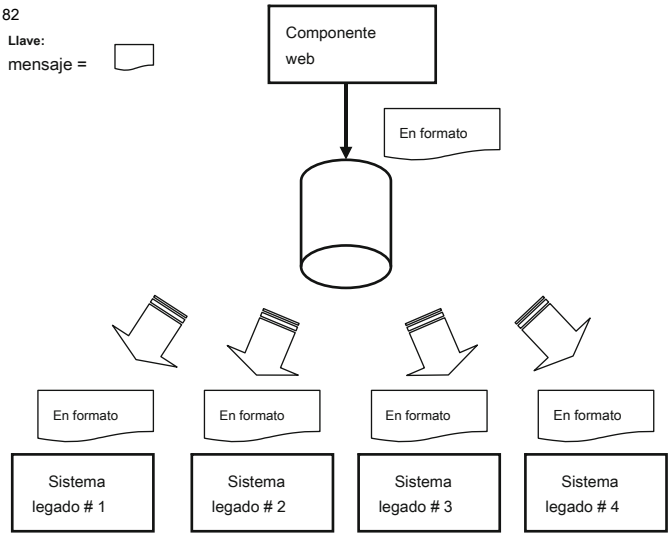


Fig. 6.1 El uso de MOM para comunicar una actualización de los datos del cliente a cuatro sistemas heredados

[1] los fondos de datos duplicados de este tipo son muy comunes en las empresas. Por ejemplo, mi banco se las arregla para enviar mi tarjeta de crédito declaración y de tarjetas de crédito puntos de recompensa a diferentes direcciones.

[2] El MOMmay desplegar una cola diferente para cada uso de la herencia o de una sola cola, e incluyen un “destino” campo en

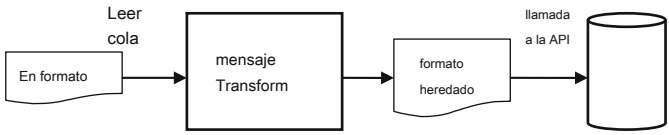


Fig. 6.2 la transformación del mensaje común a un formato heredado-específico

Cada sistema de la herencia tiene un componente de interfaz de colas que pueden leer los mensajes de la cola, y el uso de los datos en el mensaje, crear una llamada a la API de actualización de datos del cliente que es compatible con el sistema anterior. En este ejemplo, el componente de la interfaz sería leer el mensaje de la cola, extraer los fi campos de datos específicos del mensaje que tiene que llamar a la API de su sistema heredado, y emitir finalmente la llamada a la API. Como se muestra en la Fig. 6.2 , El componente de interfaz está básicamente realizar una transformación de la En formato a un formato adecuado para su sistema heredado asociado.

Por lo tanto, para cada aplicación heredada, hay un componente dedicado que ejecuta la lógica para transformar el mensaje entrante en una llamada a la API del sistema heredado el formato correcto. La transformación se lleva a cabo en el código de programa del componente.

Esta solución tiene algunas implicaciones interesantes:

- Si el común En formato cambios de formato del mensaje, entonces el componente web y cada componente del sistema legado que realiza la transformación debe ser modificado ed y probado.
- Si los cambios de la API del sistema legado, a continuación, sólo la transformación de ese sistema debe ser modi fi y probado.
- La modificación de cualquiera de las transformaciones más probable que requiere la coordinación con el equipo de desarrollo que son responsables del mantenimiento del sistema (s) de legado. Estos equipos de desarrollo son los que conocen los detalles íntimos de cómo acceder a la API del sistema legado.

Por lo tanto, hay un estrecho acoplamiento entre todos los componentes de esta arquitectura. Esto es causado por la necesidad de que se pongan de acuerdo sobre el formato de los mensajes que se comunica. Además, en las grandes organizaciones (o incluso con más fuerza, a través de fronteras organizativas), comunicar y coordinar cambios en el formato de mensaje común a través de múltiples equipos de desarrollo de sistema heredado puede ser lento y doloroso. Es el tipo de cosas que le gustaría evitar si es posible.

La solución alternativa obvia es pasar la responsabilidad de la transformación formato de mensaje al componente web. Esto garantizaría que los mensajes se envían a cada componente de la interfaz del sistema heredado en el formato que necesitan para simplemente llamar a la API anterior. La complejidad de transformación está ahora en un solo lugar, el componente web, y el componente de interfaz de sistema legado se vuelve simple. Básicamente se lee un mensaje de la cola y llama a la API asociado el uso de los datos en el mensaje. Los cambios en el En formato mensaje no causan cambios en los componentes de la interfaz de legado, ya que sólo necesita modificar el componente Web y pruebas. Los cambios en cualquier API legado, aunque requieren que el equipo de desarrollo de sistema heredado específico para solicitar un nuevo formato de mensaje del equipo de desarrollo de componentes web.

Esta es una solución mucho mejor, ya que reduce el número de cambios necesarios para los diferentes sistemas de software involucrados (y recuerda, “cambio” significa “prueba”). El mayor inconveniente de esta solución es la complejidad del componente web. La transformación para cada sistema legado está incrustado en su código de programa, por lo que es propenso a modi fi cación ya que está acoplado de manera efectiva a los formatos de mensaje de cada sistema heredado se comunica con.

Aquí es donde intermediarios de mensajes ofrecen una solución alternativa potencialmente atractiva. **Arquitectónicamente, un corredor es un patrón de arquitectura conocida [3] la incorporación de un componente que desacopla** los clientes y servidores por mediación de las comunicaciones entre ellos. Del mismo modo, el middleware intermediario de mensajes aumenta las capacidades de una plataforma de MOM para que la lógica de negocio relacionada con la integración puede ser ejecutado dentro del corredor. En nuestro ejemplo, el uso de un agente que podríamos integrar las reglas de transformación de mensaje para cada sistema heredado dentro del corredor, dando una solución como en la **figura. 6.3 . Una solución de intermediario de mensajes es atractiva porque permite separar por completo el componente web** y los componentes de interfaz de legado. El componente web, simplemente se ensambla y emite un mensaje, y el corredor transforma el mensaje en el formato necesario para cada sistema heredado. A continuación, envía un mensaje de salida a los componentes de la interfaz del sistema heredado en el formato exacto que desean.

Otra atracción es la simplificación de todos los componentes en el sistema, ya que ahora no tienen que preocuparse por la transformación formato de mensaje. La lógica de transformación mensaje se localiza dentro del intermediario de mensajes y se convierte en la responsabilidad del grupo de integración de mantener. En consecuencia, si se necesitan cambios en la red o sistema legado formatos de mensaje, el equipo de desarrollo

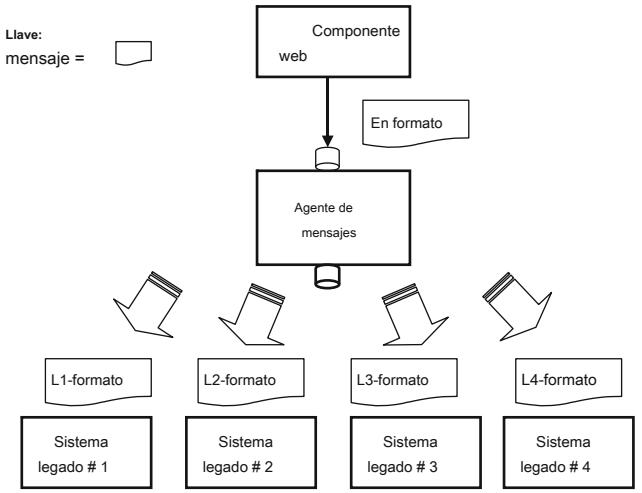


Fig. 6.3 Desacoplamiento de los clientes y servidores con un intermediario de mensajes

[3] **Véase la referencia Buschmann en Lectura adicional, Cap. 1. 84**

responsable solamente necesitan actuar de enlace con el grupo de integración, cuyo trabajo es actualizar correctamente las transformaciones.

No es un trabajo enorme para implementar el patrón intermediario en conjunto con una plataforma estándar de MOM. [4] Tal solución todavía tendría la desventaja de de fi nir la lógica de transformación en el código del programa. Para las transformaciones simples, esto no es un gran problema, pero muchas de estas aplicaciones implican transformaciones complejas con fi formateo de cadenas ddly y concatenaciones, fórmulas para calcular valores compuestos, y así sucesivamente. Nada demasiado difícil de escribir, pero si había una solución mejor que hizo la creación de transformaciones complejas simple, dudo que muchas personas se quejan.

tecnologías de intermediario de mensajes comienzan a sobresalir en esta etapa, ya que proporcionan herramientas especializadas para:

- Gráficamente que describe las transformaciones de mensajes complejas entre formatos de entrada y formatos de salida. Las transformaciones pueden ser simples en términos de movimiento de un valor de campo de entrada fi a un campo de salida fi, o pueden definirse usando lenguajes de script (típicamente producto específico c) que puede realizar varias formato, conversiones de datos, y transforma matemáticos.
- De alto rendimiento motores de transformación mensaje de multiproceso que puede HanDLE múltiples peticiones de transformación simultáneas.
- Al describir y ejecutar flujos de mensaje, en el que un mensaje entrante puede ser encaminado a diferentes transformaciones y salidas en función de los valores en el mensaje entrante.

Un ejemplo de una herramienta de mapeo de mensajes se muestra en la Fig. 6.4 . Esto es de Microsoft Asignador de BizTalk y es típico de la clase de tecnologías de mapeo. En BizTalk,

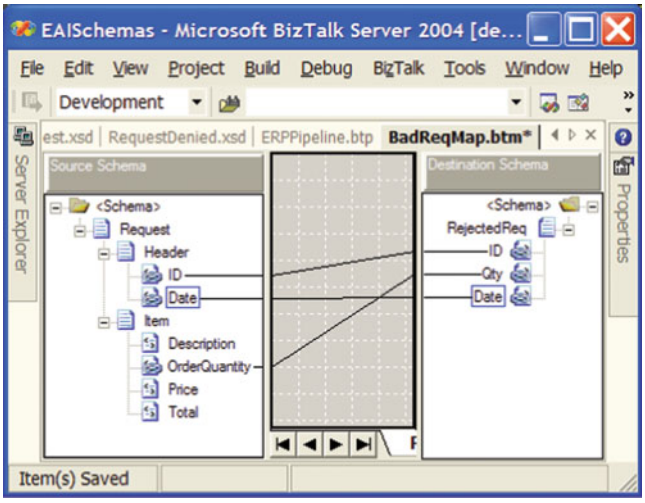


Fig. 6.4 Un ejemplo herramienta de mapeo de intermediario de mensajes

[4] La solución se deja como ejercicio para el lector!

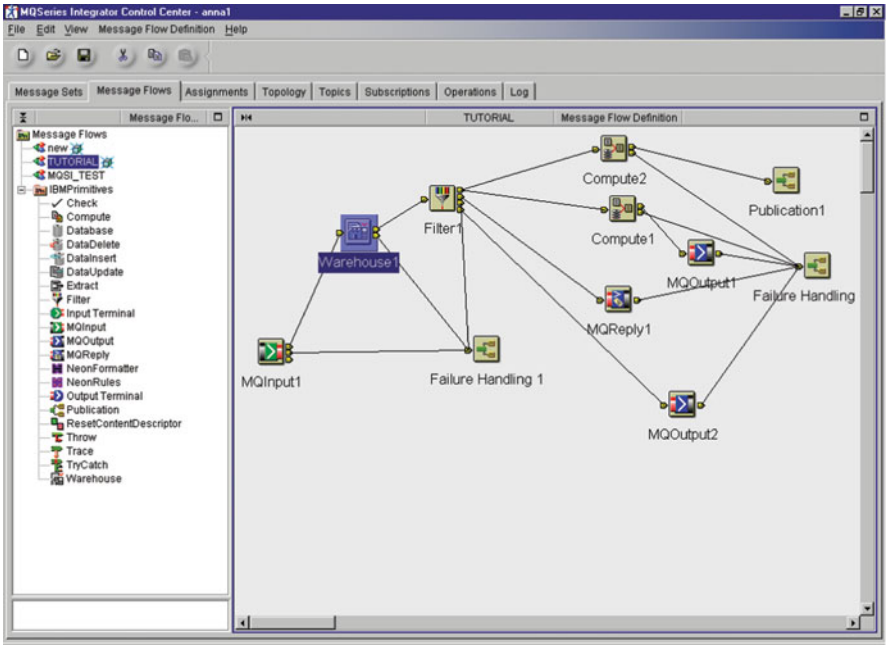


Fig. 6.5 El enrutamiento de mensajes y de procesamiento 86

el asignador puede generar las transformaciones necesarias para mover datos entre dos esquemas XML, con las líneas que representan la asignación entre origen y destino esquemas. Secuencias de comandos (no mostrado en la figura) se pueden asociar con cualquier mapeo para definir asignaciones más complejas.

Un ejemplo de una herramienta de fi nición enrutamiento de mensajes típico se muestra en la Fig. 6.5 . Esta es la tecnología WebSphere de IBM MQSI. Se muestra cómo un mensaje entrante, entregado en una cola, puede ser procesado de acuerdo con algún valor de los datos en el mensaje. En el ejemplo, una Filtrar componente inspecciona los valores de campo mensaje fi entrantes, y en base a condiciones especificado fi, ejecuta uno de los dos cálculos, o envía el mensaje a una de las dos colas de salida. El mensaje de flujo también de fi nes lógica de manejo de excepciones, que se invoca cuando, por ejemplo, se reciben mensajes de formato no válido.

Por lo tanto, intermediarios de mensajes son esencialmente altamente transformación mensaje y motores de enrutamiento especializados. Con sus herramientas de desarrollo personalizados asociados, hacen que sea más fácil de dE transformaciones de los mensajes finas que pueden ser:

- Fácilmente entendido y modificados con el sin cambiar las aplicaciones participantes.
- Gestionado de forma centralizada, lo que permite un equipo responsable de la integración de aplicaciones de coordinar y cambios de prueba.
- Ejecutado por un alto rendimiento, motor de transformación de multiproceso.

Por supuesto, como la lógica de integración se vuelve más y más compleja, utilizando un intermediario de mensajes para implementar esta lógica es esencialmente equivalente a mover la complejidad

desde los puntos finales de integración para el corredor. Es una decisión de diseño arquitectónico, basado en la especi fi cs de una empresa y su entorno técnico y social, si se trata de una buena decisión o no. No hay respuestas simples, recuerda.

Es importante destacar que, intermediarios de mensajes operan en un nivel por mensaje. Reciben un mensaje de entrada, transformarla de acuerdo con las reglas de enrutamiento de mensajes y la lógica, y la salida del mensaje o mensajes que resulta a sus destinos. Corredores funcionan mejor cuando estas transformaciones son de corta duración y ejecutan rápidamente, por ejemplo, unos pocos milisegundos. Esto se debe a que por lo general son optimizados para el rendimiento y por lo tanto tratan de evitar los gastos generales que ralentizar transformaciones. En consecuencia, si un corredor o sus accidentes ordenador central, que se basa en el hecho de que no sólo la transformación puede ser ejecutado de nuevo desde el principio, es decir, no se necesita el estado caro y gestión de transacciones. Tenga en cuenta, sin embargo, que muchos intermediarios de mensajes soportan opcionalmente mensajería transaccional e incluso permiten que el agente para modificar las bases de datos transaccional durante la ejecución de la transformación. Estas operaciones son coordinadas por un gestor de transacciones ACID, tal como el que viene con la tecnología subyacente de MOM.

Para una gran clase de escenarios de integración de aplicaciones, la transformación de alta velocidad es todo lo que se requiere. Sin embargo, muchos de los problemas de integración de negocio requieren la definición de una serie de peticiones fl debido entre diferentes aplicaciones. Cada solicitud puede implicar varias transformaciones de los mensajes, lee y cambios a los sistemas de bases de datos externas, y una lógica compleja para controlar el flujo de mensajes entre aplicaciones y, potencialmente, incluso los seres humanos para la toma de decisiones de fl ine. Para este tipo de problemas, intermediarios de mensajes son insuficientes, y bien, lo has adivinado, se requiere aún más la tecnología. Esto es descrito en la siguiente sección.

Antes de continuar, sin embargo, se debe enfatizar que los intermediarios de mensajes, como todo en la arquitectura y las tecnologías de software, tienen sus desventajas. En primer lugar, muchos de ellos son tecnologías propietarias, y esto conduce a la dependencia de un proveedor. Es el precio que paga por todas esas sofisticadas herramientas de desarrollo y despliegue. En segundo lugar, en las aplicaciones de mensajería de alto volumen, el corredor puede convertirse en un cuello de botella. La mayoría de los productos corredor mensaje de apoyo a la agrupación agente para aumentar el rendimiento, la escalabilidad y fiabilidad, pero esto viene a costa de la complejidad y dólares. Recientemente han surgido corredores de código abierto, con la mula [5] siendo un ejemplo de alta calidad. Estas tecnologías son implementaciones de alta calidad y bien vale la pena considerar en muchos escenarios de integración.

6.3 Procesos de Negocio orquestación

Los procesos de negocio en las empresas modernas pueden ser complejos en términos del número de aplicaciones empresariales que deben acceder y actualizar para completar el servicio de negocio. Como ejemplo, la Fig. 6.6 es una representación simple de un proceso de negocio de pedido de cliente, en el que se produce la siguiente secuencia de eventos.

5 http://www.mulesoft.org/display/COMMUNITY/Home

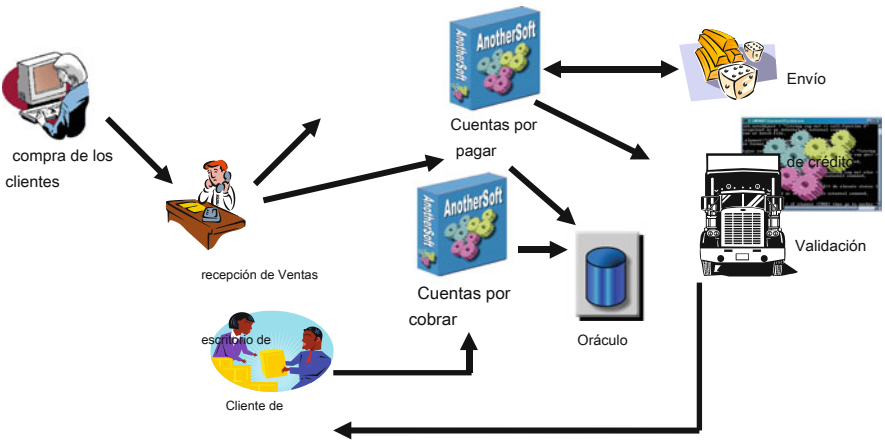


Fig. 6.6 Un proceso típico de negocios 88

Un cliente realiza un pedido a través de un centro de llamadas. Los datos del cliente se almacena en un paquete de gestión de relación con el cliente (por ejemplo, Oracle Siebel). Una vez realizado el pedido, el crédito del cliente es validado mediante un servicio de crédito externo, y la base de datos de las cuentas por pagar se actualiza para registrar la orden y enviar una factura al cliente.

Realizar un pedido provoca un mensaje que se enviará a envío, que actualizar su sistema de inventario y enviar el pedido al cliente. Cuando el cliente recibe la orden, que pagan por los bienes y el pago se registra en el sistema de cuentas recibidas. Todos los datos financieros se extraen periódicamente a partir de los sistemas de contabilidad y se almacenan en un almacén de datos Oracle para informes de gestión y archivo.

La implementación de este tipo de procesos de negocio tiene dos retos principales. En primer lugar, desde el tiempo se hace un pedido a cuando se recibe el pago podría tardar varios días o semanas, o incluso más tiempo si los artículos están fuera de stock. En algún lugar a continuación, el estado actual del proceso de negocio para un fin determinado, lo que representa exactamente en qué etapa se está haciendo, debe ser almacenado, potencialmente durante mucho tiempo. La pérdida de este estado, y por lo tanto el estado del pedido, no es una opción deseable.

En segundo lugar, las excepciones en el proceso de compra puede hacer que el estado de la orden de fallar y deshacer. Por ejemplo, una orden se da por alguna acción del artículo. Vamos a suponer que esta acción no está disponible en el almacén, y cuando se reordena, el proveedor le dice al almacén que la población de edad ya está obsoleta, y que un modelo más nuevo, más caro será reemplazarlo. El cliente es informado de esto, y deciden cancelar el pedido. Cancelación requiere los datos de la orden para ser retirados del almacén, cuentas a pagar, y Siebel Systems. Esto es potencialmente una tarea compleja para realizar de forma fiable y correctamente.

Este estilo de comportamiento de reversión puede ser definido por el diseñador de procesos utilizando una instalación conocida como una transacción de compensación. transacciones de compensación permiten al

diseñador de procesos de forma explícita definen la lógica necesaria para deshacer una transacción fallida que parcialmente completado.

En los procesos de negocio de larga duración, tales como procesamiento de órdenes de venta, transacciones ACID estándar, que se bloquean todos los recursos hasta que se complete la transacción, no son factibles. Esto se debe a que bloquean los datos en los sistemas de negocio para potencialmente minutos, horas o incluso semanas con el fin de lograr el aislamiento de transacción. datos bloqueados no se puede acceder por transacciones simultáneas, y por lo tanto, la contención de bloqueo hará que estos a esperar (o más probablemente fallan a través de tiempo de espera) hasta que los bloqueos se liberan. Tal situación es poco probable que produzca de alto rendimiento y las implementaciones de procesos de negocio escalables para los procesos de negocio de larga duración.

Comportamiento transaccional para procesos de larga duración está, por tanto, por lo general maneja mediante la agrupación de una serie de actividades de proceso en un ámbito de transacción de larga duración. transacciones largo de funcionamiento comprenden múltiples actividades del proceso que no colocan candados en los elementos de datos que se modifican en los diversos sistemas de negocios. Las actualizaciones se realizan y se comprometieron localmente en cada sistema de negocio. Sin embargo, si cualquier actividad en el ámbito de transacción falla, el diseñador debe especificar una función de compensación. El papel del compensador es deshacer los efectos de la transacción que ya se han comprometido. Esencialmente, esto significa deshacer todos los cambios que la transacción había hecho, dejando a los datos en el mismo estado en que estaba antes de la transacción iniciada.

transacciones largo de funcionamiento son notoriamente difíciles de poner en práctica correctamente. Y a veces son algo imposible de aplicar con sensatez - ¿cómo se efectúa la compensación de un proceso de negocio que se ha enviado un correo electrónico confirmando un pedido ha sido enviado o ha enviado una factura? Por lo tanto, las soluciones tecnológicas para compensar las transacciones no erradican cualquiera de estos problemas fundamentales. Sin embargo, sí proporcionan al diseñador una herramienta para hacer la existencia de una transacción de larga duración explícita, y un marco de ejecución que llama automáticamente el compensador cuando se producen fallos. Para muchos problemas, este es su fi ciente para la construcción de una solución viable.

Como la fig. 6.7 ilustra, la orquestación de procesos de negocio (BPO) plataformas están diseñadas para hacer que la aplicación de estas larga ejecución, los procesos de negocio altamente integrados relativamente sencillo. plataformas BPO suelen ser construido como una capa que aprovecha algún tipo de infraestructura de mensajería tales como SOA o un intermediario de mensajes. Aumentan la capa de mensajería con:

- Administración del Estado: el estado de ejecución de un proceso de negocio se almacena persistently en una base de datos. Esto hace que sea resistente a los fallos en el servidor de BPO. Además, una vez que el estado del proceso se almacena en la base de datos, no consume recursos computacionales en el motor de BPO hasta que el trabajo concreto flujo se reanuda la instancia.
- Herramientas de desarrollo: herramientas de definición de procesos visuales se proporcionan para de fi nir Procesos de negocios.
- herramientas de implementación: éstos permiten a los desarrolladores para enlazar fácilmente lógica de procesos de negocio pasos para los sistemas operativos subyacentes utilizando diversos tipos de conectividad, incluyendo las colas de mensajes, protocolos web, SOAP y fi l de los sistemas.

Un ejemplo de la tecnología de BizTalk de Microsoft se muestra en la Fig. 6.8 . Esta muestra el diseño de un simple proceso de negocio para el ejemplo de pedido en la Fig. 6.6 .

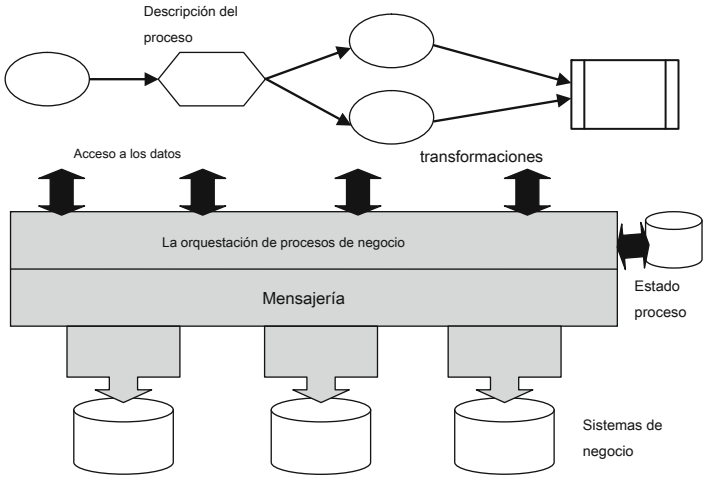


Fig. 6.7 Anatomía de una plataforma de orquestación de procesos de negocio 90

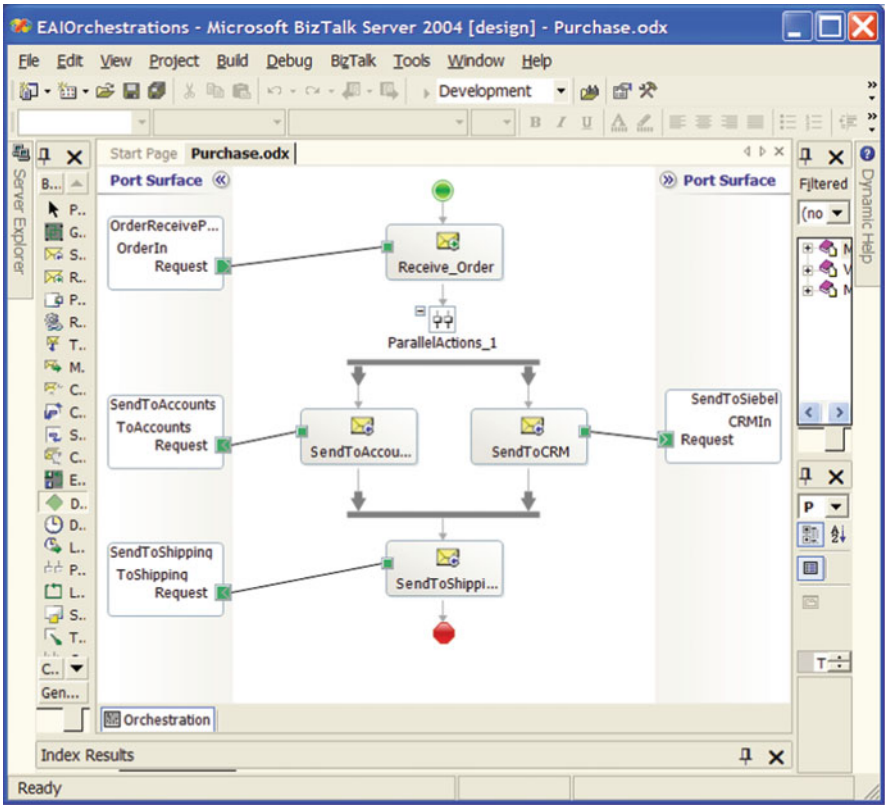


Fig. 6.8 BizTalk procesos de negocio definición

Los mensajes se envían y se reciben por las actividades en el proceso a través de puertos. Puertos, básicamente, se conectan a los sistemas de negocio utilizando un mecanismo fi nida transporte port-de, por ejemplo, HTTP, una cola de mensajes o un archivo. Todos los mensajes gestionados dentro de una orquestación de BizTalk debe ser definido por los esquemas XML. Las actividades pueden ser llevadas a cabo en secuencia o en paralelo como se muestra en el ejemplo.

Los motores de BPO son la adición más reciente a la pila de middleware de TI. La necesidad de su funcionalidad ha sido impulsado por el deseo de automatizar cada vez más procesos de negocio que deben acceder a numerosas aplicaciones de negocios independientes. No cabe duda de que esta tendencia va a continuar como empresas reducir los costes al integrar mejor y coordinar sus aplicaciones internas, y sin problemas de conexión a socios de negocios externos.

6.4 Problemas de integración Arquitectura

La dificultad de integrar aplicaciones heterogéneas en las grandes empresas es grave. Si bien hay muchos temas a tratar en la integración empresarial, en el fondo es un problema arquitectónico respecto a la capacidad modi fi. La historia va así.

Suponga que su empresa tiene cinco diferentes aplicaciones de negocios que necesitan para apoyar la integración de algunos de los nuevos procesos de negocio. Al igual que cualquier arquitecto sensible, decide implementar los procesos de estos negocios uno a la vez (como sabe un enfoque “big bang” está condenado al fracaso!).

El proceso requiere primero uno de los sistemas de negocios para enviar mensajes a cada uno de los otros cuatro, mediante sus interfaces de mensajería publicados. Para ello, el remitente debe crear una carga útil del mensaje en el formato requerido por cada aplicación de negocio. Suponiendo mensajes unidireccionales solamente, esto significa nuestro proceso de negocio primero debe ser capaz de transformar sus datos de origen en cuatro diferentes formatos de mensaje. Por supuesto, si los otros sistemas de negocios deciden cambiar sus formatos, a continuación, estas transformaciones se deben actualizar. Lo que hemos creado con este diseño es un estrecho acoplamiento, es decir, los formatos de mensaje, entre los sistemas de negocio de origen y destino. Este escenario se representa en el lado izquierdo de la figura. 6.9 .

Con el primer trabajo de procesos de negocio, y con muchos usuarios de negocios feliz, usted va a construir incrementalmente el resto. Cuando haya terminado, que encontramos

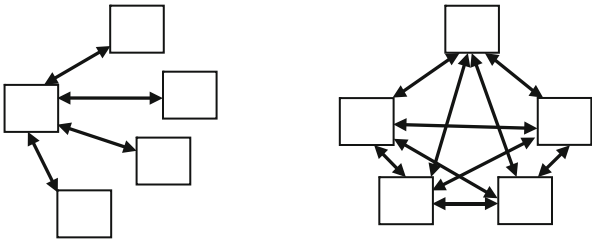


Fig. 6.9 la integración de aplicaciones en una arquitectura de punto a punto

1 de procesos de negocio = 4 interfaces

5 procesos de negocio = 20 interfaces de

que ha creado una arquitectura de esa manera en el lado derecho de la figura. 6.9 . Cada aplicación envía mensajes a cada uno de los otros cuatro, creación de 20 interfaces, o dependencias, que necesitan ser mantenido. Cuando una aplicación de negocios es modi fi cado, es posible que cada uno de los otros tendrán que actualizar sus transformaciones de los mensajes para enviar mensajes en un formato de nueva requerido.

Esto es un ejemplo a pequeña escala de un problema que existe en miles de organizaciones. He visto arquitecturas de software para empresas que tienen 300 interfaces de punto a punto entre 40 o más aplicaciones de negocios independientes. Cambio de interfaz de mensajes de una aplicación se convierte en un ejercicio que da miedo en este tipo de empresas, como tantos otros sistemas dependen de ella. A veces hacer cambios es tan temible, los equipos de desarrollo simplemente no lo hará. Es simplemente demasiado arriesgado.

En el caso general, el número de interfaces entre norte aplicaciones es (norte 2 NORTE).

Así como norte crece, el número de posibles interfaces de crece exponencialmente, por lo que este tipo de arquitecturas de punto a punto no escalable en términos de capacidad modificada.

Ahora es cierto que muy pocas empresas tienen una arquitectura totalmente conectado punto a punto como el que en el lado derecho de la figura. 6.9 . Pero también es cierto que muchas interfaces entre dos aplicaciones son de dos vías, que requiere dos transformaciones. Y la mayoría de las aplicaciones tienen más de una interfaz, por lo que en la realidad el número de interfaces entre dos aplicaciones bien acoplados pueden ser considerablemente mayor que uno.

Otro nombre para una arquitectura de punto a punto es una “arquitectura de espagueti”, es de esperar, por razones obvias. Cuando se utiliza este término, muy pocas personas se refieren a los espaguetis con connotaciones positivas por lo general asociados con la sabrosa comida italiana. De hecho, como la disciplina de la integración de la empresa floreció a finales de 1990, el dogma que emerge es que las arquitecturas de espagueti deben evitarse a toda costa. La solución promovida, por muchas y buenas razones, era utilizar un intermediario de mensajes, como se ha explicado anteriormente en este capítulo.

Vamos a analizar exactamente lo que sucede cuando una arquitectura de espaguetis se transforma mediante un arquitectura de punto a punto con un intermediario de mensajes, como se ilustra en la Fig. 6.10 . La complejidad en los puntos extremos de integración, es decir, las aplicaciones de negocios, se reduce considerablemente, ya que sólo envían mensajes a través de sus formatos nativos al corredor, y éstos se transforman dentro del corredor para el formato de destino deseado. Si necesita cambiar un punto final, a continuación, sólo tiene que modificar las transformaciones de los mensajes dentro del corredor que dependen de ese punto final. No hay otras aplicaciones de negocios saben o cuidado.

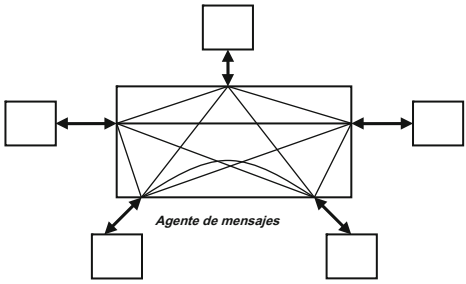


Fig. 6.10 La eliminación de una

A pesar de todas estas ventajas a la introducción de un intermediario de mensajes, la no hay almuerzo gratis [6]
principio, como siempre, se aplica. Las desventajas son:

- La arquitectura de espagueti realmente todavía existe. Ahora es residente dentro del mensaje
  corredor, donde las dependencias complejas entre formatos de mensaje son capturados en definida
  transformaciones Message Broker-de.
- Los corredores son, potencialmente, un cuello de botella, ya que todos los mensajes entre
  aplicaciones deben pasar por el corredor. Los buenos corredores soportan la replicación y despliegues agrupados a escalar su rendimiento. Pero, por supuesto, esto aumenta el despliegue y la complejidad de la gestión, y más que probable que los costos de las licencias asociadas a una solución. vendedores de intermediario de mensajes, tal vez no es sorprendente que rara vez se ve a este último punto como una desventaja.

Así intermediarios de mensajes son muy útiles, pero no una panacea por cualquier medio para arquitecturas de integración. Sin embargo, hay un enfoque de diseño que puede ser utilizado que posee la escalabilidad de una arquitectura de punto a punto con las características modi capacidad fi de solución basada en corredor.

La solución es para definir un modelo de datos de la empresa (también conocido como un modelo de datos canónica) que se convierte en el formato de destino para todas las transformaciones de los mensajes entre aplicaciones. Por ejemplo, un problema común es que todos los sistemas de su empresa tienen diferentes formatos de datos a De la información del cliente definir. Cuando una aplicación se integra con otra, (o un intermediario de mensajes) debe transformar su formato de mensaje al cliente para el formato de mensaje de destino.

Ahora vamos a suponer que definen un formato de mensaje canónica para la información del cliente. Esto puede ser usado como el formato de destino para cualquier aplicación de negocio que necesita para intercambiar datos relacionados con el cliente. El uso de este formato de mensaje canónica, un intercambio de mensajes se ha reducido a los siguientes pasos:

- aplicación de código transforma los datos de los clientes locales en los clientes canónica
  formato de la información.
- Fuente envía mensaje a apuntar con el formato de mensaje canónica como carga útil.
- El objetivo recibe un mensaje y transforma el formato canónico en su propio local,
  la representación de datos de clientes.

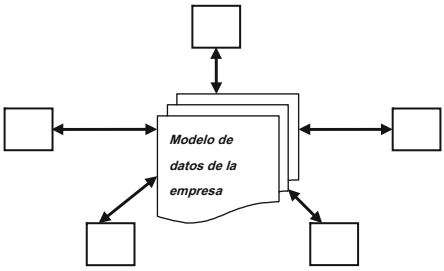
Esto significa que cada punto final (aplicación empresarial) debe saber:

- ¿Cómo transformar todos los mensajes que recibe del formato canónico a su local de
  formato
- ¿Cómo transformar todos los mensajes que envía desde su formato local a la canónica
  formato

Como la fig. 6.11 ilustra, mediante el modelo de datos de la empresa para el intercambio de mensajes, se obtiene lo mejor de ambos mundos. El número de transformaciones se reduce a 2 * N (suponiendo una única interfaz entre cada punto final). Esto nos da mucho mejores características de capacidad fi caciones. También, ya que ahora son considerablemente menores y

[6] http://en.wikipedia.org/wiki/Tanstaa fl

Fig. 6.11 Integración mediante un modelo de datos de la empresa



transformaciones menos complejas para construir, las transformaciones se pueden ejecutar en el extremo apunta a sí mismos. No tenemos ninguna necesidad de una arquitectura centralizada corredor-estilo. Esta escala bien, ya que no es inherentemente sin cuello de botella en el diseño. Y no hay necesidad de hardware adicional para el corredor, y los costes de licencia adicionales para nuestra solución de agente elegido.

Sospecho que algunos de ustedes podrían estar pensando que esto es demasiado bueno para ser verdad. Tal vez hay al menos una opción de bajo costo almuerzo aquí?

Siento decepcionarte, pero hay razones reales por esta arquitectura no es ubicuo en la integración de la empresa. La principal es la enorme dificultad de diseñar, y luego conseguir un acuerdo sobre un modelo de datos de la empresa en una organización grande. En un sitio de campo verde, el modelo de datos de la empresa es algo que puede ser diseñado por adelantado y todos los puntos finales tienen la obligación de adherirse. Pero los sitios de campo verdes son raros, y sistemas de la empresa de mayor organización han crecido de manera orgánica durante muchos años, y rara vez de una manera planificada y coordinada. Es por esto que las soluciones basadas en los agentes tienen éxito. Reconocen la realidad de los sistemas de la empresa y la necesidad de la construcción de muchas transformaciones ad hoc entre los sistemas de una manera fácil de mantener.

Hay otros impedimentos para el establecimiento de formatos de datos canónicos. Si los sistemas se integran con las aplicaciones de un socio de negocios sobre las que no tienes control, entonces lo más probable es imposible establecer un único conjunto acordado de formatos de mensaje. Este problema tiene que ser abordado en una escala mucho más amplia, donde los grupos industriales enteras se reúnen para definir formatos de mensaje común. Un buen ejemplo es RosettaNet [7] que tiene protocolos definida de para la automatización de las cadenas de suministro en la industria de semiconductores. Como estoy seguro de que se pueda imaginar, nada de esto sucede rápidamente. [8]

Para muchas organizaciones, las ventajas de utilizar un modelo de datos empresariales sólo pueden ser explotados de forma incremental. Por ejemplo, una nueva instalación de sistemas de negocio podría presentar oportunidades para iniciar elementos de fi nición de un modelo de datos de la empresa y para construir arquitecturas de punto a punto que se aprovechan de las transformaciones de punto final a formatos canónicos. O su agente podría estar a punto de ser obsoleta y que requieren para actualizar su lógica de transformación? Me gustaría recomendar tomar cualquier oportunidad que tenga.

---

[7] http://www.rosettanet.org

[8] Ver http://www.ebxml.org/ para ejemplos de iniciativas en este ámbito. 94

6.5 ¿Qué es un bus de servicios empresariales

Verá el término “ESB” que se utiliza ampliamente en la literatura arquitectura orientada a servicios. Cuando primero oí esto, se preguntó qué “Extra Bitter especial” tenía que ver con las arquitecturas de integración de software, y cuando descubrí que representaba Enterprise Service Bus, me decepcionó profundamente. De todos modos, aquí está mi interpretación es cierto tanto cínico de donde la sigla ESB vino.

En algún lugar en medio de la última década (~ 2003-2005), SOA se estaba convirtiendo en la “próxima gran cosa” en la integración empresarial. Los proveedores de software necesitan algo nuevo para ayudar a vender su tecnología de integración para soportar una SOA, por lo que uno de ellos (no estoy seguro de quién fue primero) acuñó el término ESB. De repente, todos los proveedores tenía un ESB, que era básicamente sus tecnologías de orquestación broker de mensajes y procesos de negocio rebautizado con, por supuesto, la capacidad de integrar los puntos finales de servicios web. Si se mira debajo de las sábanas de un ESB, que encontramos todos los elementos técnicos y software de integración enfoques descritos en este y en los dos últimos capítulos.

Hay una gran cantidad de de fi niciones que hay para los ESB. Todos más o menos de acuerdo en que un ESB proporciona mecanismos fundamentales para arquitecturas de integración complejos a través de un motor de mensajería orientado a eventos y basada en estándares. Hay cierto debate sobre si un ESB es una tecnología o un patrón de diseño de integración de software, pero algunos debates realmente no vale la pena involucrarse en. Usted puede comprar o descargar productos llamados ESB, y estos suelen ofrecer una infraestructura de middleware basado en mensajería que tiene la capacidad de conectarse a los puntos finales externos del sistema a través de una variedad de protocolos - TCP / IP, SOAP, JMS, FTP, y muchos más. Si lo que ha leído hasta ahora en este libro se ha hundido en algún grado, no creo que realmente necesita saber más.

6.6 Lectura adicional

Hay un enorme volumen de lectura del potencial en la materia cubierta en este capítulo. Las referencias que siguen deben darle un buen punto de partida para ahondar más profundamente.

DS Linthicum. Siguiente Generación de Integración de Aplicaciones: A partir de información simple de Servicios Web. Addison-Wesley, 2003.

David Chappell, Enterprise Service Bus: teoría en la práctica, O'Reilly Media, 2004 Gero M € Sistemas de eventos basados en Uhl, Ludger Fiege, Peter Pietzuch, distribuido, Springer-Verlag 2006.

Los siguientes tres libros tienen una cobertura amplia e informativa de los patrones de diseño para la integración de la empresa y la mensajería.

M. Fowler. Los patrones de arquitectura de aplicación empresarial. Addison-Wesley, 2002.

G. Hohpe, B. Woolf. Los patrones de integración empresarial: diseñar, construir y desplegar soluciones de mensajería. Addison-Wesley, 2003.

C. Bussler, conceptos de integración B2B y Arquitectura, Springer-Verlag 2003.

En términos de tecnologías, aquí hay algunos intermediarios de mensajes de calidad y sistemas de orquestación de procesos de negocio para mirar:

David Dossot, John D'EMIC mula en Acción, Manning Press, 2009. Tijs Rademakers, Jos Dirksen, ESB de código abierto en Acción: Ejemplo aplica-

taciones en mula y ServiceMix, Manning Press, 2008. 96

Capítulo 7

Una Arquitectura de Procesos de Software

Esquema 7.1 Proceso

El papel de un arquitecto es mucho más que simplemente llevar a cabo una actividad de diseño de software. El arquitecto normalmente debe:

- **Trabajar con el equipo de requisitos:** El equipo de requisitos se centrará en la obtención de los requisitos funcionales de los grupos de interés de aplicación. El arquitecto desempeña un papel importante en la recopilación de requisitos mediante la comprensión de las necesidades de los sistemas generales y asegurar que los atributos de calidad apropiados son explícitas y entendido.

- **Trabajar con diferentes grupos de interés de aplicación:** Arquitectos juegan un enlace fundamental papel asegurándose de que todas las necesidades de los interesados de la aplicación se entienden y se incorporan en el diseño. Por ejemplo, además de las necesidades de los usuarios de negocio para una aplicación, los administradores del sistema, será necesario que la aplicación se puede instalar fácilmente, supervisa, gestiona y actualiza.
- **Dirigir el equipo de diseño técnico:** De fi nición de la arquitectura de la aplicación es un diseño actividad. El arquitecto lleva un equipo de diseño, que comprende los diseñadores del sistema (o en grandes proyectos, otros arquitectos) y clientes potenciales técnicos con el fin de producir el diseño de arquitectura.

- **Trabajar con la gestión de proyectos:** El arquitecto trabaja en estrecha colaboración con el proyecto gestión, ayudando con la planificación de proyectos, estimación y asignación de tareas y programación.

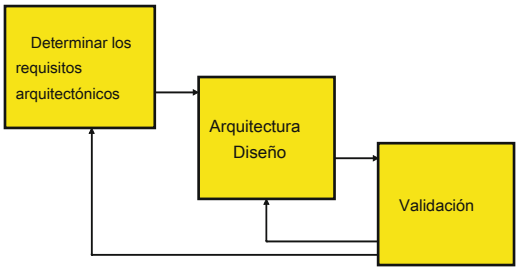
Con el fin de guiar a un arquitecto a través de la definición de la arquitectura de la aplicación, es útil seguir un **proceso de ingeniería de software definida de.** Figura 7.1 **muestra una de tres pasos proceso simple, la arquitectura** iterativo que se puede utilizar para dirigir las actividades durante el diseño. Brevemente, los tres pasos son:

- **De requerimientos de arquitectura de definir:** Esto implica la creación de una declaración o modelo de los requisitos que impulsarán el diseño de la arquitectura.
- **Diseño arquitectónico:** Esto implica la de fi nición de la estructura y responsabilidades de los componentes que formarán la arquitectura.

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3_7, # Springer-Verlag Berlin Heidelberg 2011

arquitectura de tres pasos 98

Fig. 7.1 Un proceso de diseño de la



- Validación: Esto implica “probar” la arquitectura, por lo general caminando a través del diseño, en contra de los requisitos actuales y futuros requisitos conocidos o posibles.

Este proceso arquitectura es inherentemente iterativo. Una vez que se propone un diseño, validación de que puede mostrar que el diseño necesita fi cación modi, o que ciertos requisitos deben ser más definido y entendido. Tanto éstos conducen a mejoras en el diseño, validación posterior, y así sucesivamente, hasta que el equipo de diseño se satisface cuando se cumplan los requisitos.

Es importante tener en cuenta la flexibilidad de este proceso. Arquitectura veces se caracteriza como Big Diseño por adelantado por la comunidad métodos ágiles, pero en realidad no tiene que ser. Si está trabajando en un proyecto utilizando métodos ágiles, es posible que desee tener algunas primeras iteraciones (sprints, o lo que su nomenclatura favorito es) que se centran en el establecimiento de su arquitectura general. El resultado de estas iteraciones será un prototipo de la arquitectura de referencia que encarna y valida las decisiones clave de diseño del sistema. iteraciones posteriores como base y ampliar este prototipo para agregar la funcionalidad emergente. Con la arquitectura en su lugar al principio del proyecto, la refactorización posterior se hace más simple como el núcleo del sistema permanece (la mayoría) estable, que proporciona una base sólida para la aplicación.

El resto de este capítulo se explica cada uno de estos pasos con más detalle.

7.1.1 Determinar los requisitos arquitectónicos

Antes de una solución arquitectónica puede ser diseñado, es necesario tener una idea bastante buena de los requisitos para la arquitectura de la aplicación. requisitos de arquitectura, a veces también llamados arquitectónicamente signi fi cativas requisitos o casos de uso arquitectura, son esencialmente los requisitos de calidad y no funcionales para la aplicación.

7.1.2 Identificación de las necesidades Arquitectura

Como la fig. 7.2 espectáculos, las principales fuentes de requisitos de arquitectura son el documento de requisitos funcionales, y otros documentos que la captura de varios grupos de interés

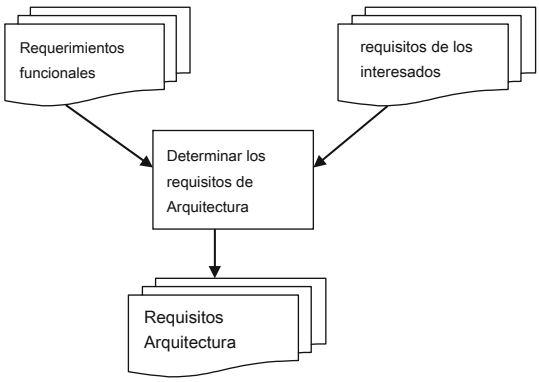


Fig. 7.2 Entradas y salidas para determinar los requerimientos de la arquitectura

necesariamente. El resultado de este paso es un documento que establece los requisitos de arquitectura de la aplicación. Por supuesto, en realidad, gran parte de la información necesita un arquitecto no está documentada. La única manera de obtener la información es hablar con los diferentes grupos de interés. Esto puede ser una tarea lenta y laboriosa, especialmente si el arquitecto no es un experto en el dominio de negocio de la aplicación.

Veamos algunos ejemplos. Una típica arquitectura requisito relativo a la fiabilidad de las comunicaciones es:

> Las comunicaciones entre los componentes deben ser garantizados para tener éxito sin la pérdida de mensajes

Algunos requisitos de arquitectura son realmente limitaciones, por ejemplo:

> El sistema debe utilizar el servidor Web basado en IIS existente y utilizar páginas Active Server para procesar peticiones web

Restricciones imponen restricciones sobre la arquitectura y son (casi siempre) no negociable. Limitan la gama de opciones de diseño de un arquitecto puede hacer. A veces esto hace que la vida de un arquitecto más fácil, ya veces no lo hace.

Mesa 7.1 enumera algunos requisitos ejemplo de arquitectura junto con el atributo de calidad que tratan.

Mesa 7.2 da algunos ejemplos típicos de limitaciones, junto con la fuente de cada restricción.

7.1.3 Requisitos Priorización de Arquitectura

Es una cosa rara cuando todos los requisitos de arquitectura de una aplicación son iguales. A menudo, la lista de requisitos de arquitectura contiene elementos que son de baja prioridad, o “esto sería bueno tener, pero no es necesario” características de tipo. Es por lo tanto importante identificar explícitamente estos, y clasificar los requisitos de arquitectura utilizando prioridades. Inicialmente, por lo general es su fi ciente para asignar a cada requerimiento de una de las tres categorías, a saber:

Tabla 7.1 Algunos atribuyen ejemplo de arquitectura requisitos de

calidad	requisito de la arquitectura
rendimiento de las aplicaciones de rendimiento debe proporcionar tiempos de respuesta sub-cuatro segundos para 90% de peticiones	
Seguridad	Todas las comunicaciones deben ser autenticados y encriptados usando certi fi cados
Administracion de recursos	El componente de servidor deben ejecutar en un extremo inferior de fi ce-servidor basado con 2 GB memoria
usabilidad	El componente de interfaz de usuario debe ejecutar en un navegador de Internet para apoyar a distancia usuarios
Disponibilidad El sistema debe funcionar 24 7 365, con la disponibilidad general de 0.99 Confiabilidad	
	No se permite ninguna pérdida de mensajes, y todos los resultados de entrega de mensajes debe ser conocida con 30 s
escalabilidad	La aplicación debe ser capaz de manejar una carga pico de 500 usuarios concurrentes durante el período de inscripción
Modi fi habilidad La arquitectura debe ser compatible con una migración gradual desde el actual Forth Idioma generación (4GL) versión a una solución de tecnología de sistemas de .NET	

Tabla 7.2 Algunos ejemplos de restricciones restricción

	requisito de la arquitectura
Negocio	La tecnología debe funcionar como un plug-in para MS BizTalk, ya que queremos vender esto a microsoft
Desarrollo El sistema debe estar escrito en Java, por lo que podemos utilizar la Lista personal de desarrollo existentes	
	La primera versión de este producto debe ser entregado dentro de los 6 meses
Negocio	Queremos trabajar estrechamente con y obtener más fondos para el desarrollo de MegaHugeTech Corp, por lo que tenemos que utilizar su tecnología en nuestra aplicación

1. Alto: la aplicación debe ser compatible con este requisito. Estos requisitos coche el diseño de la arquitectura
2. Medio: será necesario este requisito para ser apoyado en algún momento, pero no necesariamente en el / próxima versión primera
3. Bajo: esto es parte de la lista de requisitos de deseos. Soluciones que pueden alojar se desean estos requisitos, pero no son los impulsadores del diseño

Priorización vuelve más complicado en la cara de requisitos contradictorios con fl. Los ejemplos más comunes son:

- Reutilización de los componentes de la solución en función del tiempo de salida al mercado rápida. Fabricación componentes generalizados y reutilizable siempre toma más tiempo y esfuerzo.
- gasto mínimo en comparación con los productos COTS esfuerzo de desarrollo reducida / costo. productos COTS significa que tenga que desarrollar menos código, pero cuestan dinero.

No hay una solución sencilla a estos conflictos. Es parte del trabajo del arquitecto para hablar sobre esto con las partes interesadas, y llegar a posibles escenarios de solución para permitir a los temas que se entienden completamente. Con fl requisitos contradictorios pueden incluso terminar siendo la misma prioridad. Es entonces la responsabilidad de la solución a considerar las compensaciones adecuadas, y para tratar de hallar que la "línea de fi ne" que satisface adecuadamente ambos requisitos sin molestar a nadie ni tener mayor

consecuencias no deseadas en la aplicación. Recuerda que los buenos arquitectos saben cómo decir “no”.

En un proyecto con muchas partes interesadas, por lo general es una buena idea para obtener cada conjunto de partes interesadas a firmar en esta priorización. Esto es especialmente cierto en el rostro de requisitos contradictorios con fl. Una vez que esto está de acuerdo, el diseño de la arquitectura pueda comenzar.

7.2 Arquitectura

Si bien todas las tareas que lleva a cabo un arquitecto son importantes, es la calidad del diseño de la arquitectura que realmente importa. documentos de requerimientos maravillosos y redes atento con las partes interesadas no significan nada si un diseño pobre se produce.

Como era de esperar, el diseño es típicamente el más dif fi cil tarea de un arquitecto emprende. Los buenos arquitectos se basan en varios años de experiencia en ingeniería de software y diseño. No hay sustituto para esta experiencia, por lo que todo este capítulo se puede hacer es tratar de ayudar a los lectores a ganar algunos de los conocimientos necesarios lo más rápido posible.

Como la fig. 7.3 muestra, las entradas a la etapa de diseño son los requisitos de la arquitectura. La etapa de diseño en sí tiene dos pasos, que son de naturaleza iterativa. El primero consiste en elegir una estrategia general para la arquitectura, basada en los patrones de arquitectura probada. El segundo se refiere a la especificación de los componentes individuales que componen la aplicación, mostrando cómo encajan en el marco general y los asignación de responsabilidades. La salida es un conjunto de vistas de arquitectura que capturan el diseño de la arquitectura y diseño de un documento que explica el diseño, las principales razones de algunas de las principales decisiones de diseño, e identi fi ca los riesgos inherentes a la recepción del diseño hacia adelante.

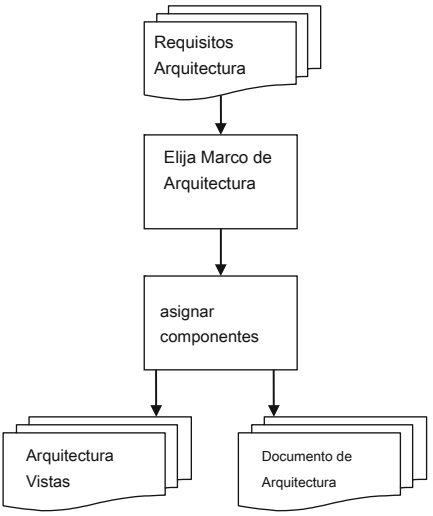


Fig. 7.3 Las entradas y salidas de diseño de la arquitectura

7.2.1 Elección del Marco de Arquitectura

La mayoría de las aplicaciones que he trabajado en los últimos 15 años se basan en torno a un pequeño número de arquitecturas probadas, bien entendidas. Hay una buena razón para esto - que trabajan. Aprovechando las soluciones conocidas minimiza los riesgos de que una aplicación fallará debido a una arquitectura apropiada.

Por lo que el paso inicial de diseño implica la selección de un marco de arquitectura que parece probable que cumpla los requisitos clave. Para aplicaciones pequeñas, un solo patrón de arquitectura como n niveles de cliente-servidor puede bastar. Para aplicaciones más complejas, el diseño va a incorporar uno o más patrones conocidos, con el arquitecto que especifica cómo estos patrones se integran para formar la estructura general.

No hay una fórmula mágica para el diseño de la estructura de arquitectura. Un requisito previo, sin embargo, es entender cómo cada uno de los principales patrones de arquitectura aborda ciertos atributos de calidad. Las siguientes subsecciones brevemente a cubrir algunos de los principales patrones utilizados, y describir la forma en que abordan los requisitos de calidad comunes.

7.2.1.1 N-Tier Cliente Servidor

En la Fig. 7.4 se ilustra la anatomía de este patrón para una aplicación web. Las propiedades principales de este

modelo son:

- Separación de intereses: Presentación, negocio y la lógica de manejo de datos son claramente se repartió en diferentes niveles.
- comunicaciones síncronas: Las comunicaciones entre niveles es sincrónico solicitud-respuesta. Las solicitudes emanan en una sola dirección desde el nivel del cliente, a través de la web y la lógica de negocio a niveles de la capa de gestión de datos. Cada nivel espera una respuesta desde el otro nivel antes de continuar.
- Implementación flexible: No hay restricciones sobre cómo una aplicación multi-nivel es desplegada. Todos los niveles se podrían ejecutar en la misma máquina, o en el otro extremo, cada nivel puede ser desplegado en su propia máquina. En las aplicaciones web, el nivel de cliente es

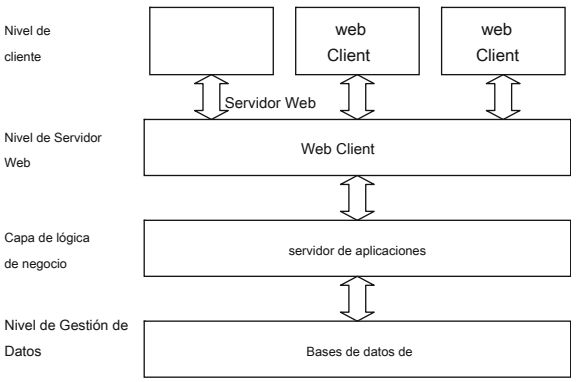


Fig. 7.4 ejemplo cliente-servidor N-tier

Tabla 7.3 atributos de calidad para el atributo patrón de calidad N-Tier Cliente Servidor

	Cuestiones
Disponibilidad	Servidores de cada nivel se pueden replicar, de modo que si uno falla, los demás permanecen disponibles. En general, la aplicación proporcionará una menor calidad de servicio hasta que se restablezca el servidor que ha fallado
el control de fallos	Si un cliente se comunica con un servidor que falla, la mayoría de aplicaciones web y implementan servidores de conmutación por error transparente. Esto significa una petición de cliente es, sin su conocimiento, redirigida a un servidor de réplica vivo que puede satisfacer la petición
Modi fi capacidad	de separación de las preocupaciones mejora la capacidad modi fi, como la presentación, negocios y lógica de gestión de datos están claramente encapsulado. Cada uno puede tener su modi lógica fi ed interna en muchos casos sin cambios de ondulación en otros niveles
Actuación	Esta arquitectura ha demostrado un alto rendimiento. Las cuestiones clave a considerar son la cantidad de hilos concurrentes apoyado en cada servidor, la velocidad de las conexiones entre los niveles y la cantidad de datos que se transfiere. Como siempre ocurre con los sistemas distribuidos, tiene sentido para reducir al mínimo las llamadas necesarias entre los niveles de fi ll ful cada solicitud
escalabilidad	Como los servidores en cada nivel se pueden replicar, y varias instancias de servidor se ejecutan en el iguales o diferentes servidores, la arquitectura escalas fuera y hacia arriba también. En la práctica, el nivel de gestión de datos menudo se convierte en un cuello de botella en la capacidad de un sistema

normalmente un navegador que se ejecuta en el escritorio del usuario, comunicar de forma remota a través de Internet con unos componentes de nivel Web.

Mesa 7.3 muestra cómo los atributos de calidad comunes pueden ser abordados con este patrón.

Precisamente cómo cada atributo de calidad está dirigida depende de la tecnología web y servidor de aplicaciones real que se utiliza para implementar la aplicación. .NET, cada implementación de JEE, y otros servidores de aplicaciones propietaria todos tienen diferente tiempo de diseño y tiempo de ejecución de funciones. Estos deben ser entendidos durante el diseño de la arquitectura de modo que no haya sorpresas desagradables se encuentran mucho más tarde en el proyecto, cuando equis fi son mucho más costosos de realizar.

El patrón de N-Tier cliente-servidor es comúnmente utilizado y el apoyo directo de las tecnologías de servidor de aplicaciones para este patrón hace que sea relativamente fácil de implementar aplicaciones usando el patrón. Es generalmente apropiada cuando una aplicación debe ser compatible con un número potencialmente grande de clientes y solicitudes simultáneas, y cada solicitud toma un intervalo de tiempo relativamente corto (unos pocos milisegundos hasta unos pocos segundos) al proceso.

7.2.1.2 Mensajería

En la Fig. 7.5 se muestran los componentes básicos de la patrón de mensajería. Las propiedades principales de este modelo son:

- comunicaciones asíncronas: Los clientes envían peticiones a la cola, donde el mensaje se almacena hasta que una aplicación elimina. Después de que el cliente haya escrito el mensaje a la cola, que continúa sin esperar a que el mensaje sea eliminado.

Fig. 7.5 Anatomía del patrón de mensajería

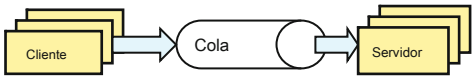


Tabla 7.4 atributos de calidad para el atributo de calidad de patrón de

mensajería	Cuestiones
Disponibilidad	colas físico con el mismo nombre lógico se pueden replicar a través de diferentes instancias del servidor de mensajería. Cuando uno falla, los clientes pueden enviar mensajes a las colas de réplicas
el control de fallos	Si un cliente se comunica con una cola que falla, se puede encontrar una cola de réplica y publicar el mensaje que hay
Modi fi capacidad	La mensajería es inherentemente imprecisa, y esto promueve alta capacidad de modi fi como clientes y servidores no están vinculados directamente a través de una interfaz. Los cambios en el formato de los mensajes enviados por los clientes pueden causar cambios en las implementaciones del servidor. Auto-descripción, formatos de mensaje detectables pueden ayudar a reducir esta dependencia de los formatos de los mensajes
Actuación	tecnología de cola de mensajes puede entregar miles de mensajes por segundo. mensajes no confiable es más rápido que fiable, con la diferencia de rendimiento depende de la calidad de la tecnología de mensajería utilizado
escalabilidad	Las colas se pueden alojar en los puntos finales de comunicación, o se replicarán a través de grupos de servidores de mensajería alojados en una o varias máquinas de servidor. Esto hace que la mensajería una solución altamente escalable

- Con fi gurable QoS: La cola puede ser con fi gurada para alta velocidad, no confiable o más lento, la entrega fiable. operaciones de la cola se pueden coordinar con las transacciones de bases de datos.
- Bajo acoplamiento: No hay una unión directa entre clientes y servidores. los cliente es ajeno a qué servidor recibe el mensaje. El servidor es ajeno en cuanto a qué cliente el mensaje vino.

Mesa 7.4 muestra cómo los atributos de calidad comunes son abordados por mensajería. Una vez más, tener en cuenta, el apoyo exacto para estos atributos de calidad es la mensajería depende del producto.

La mensajería es especialmente apropiado cuando el cliente no necesita una respuesta inmediata directamente después de enviar una solicitud. Por ejemplo, un cliente puede dar formato a un correo electrónico, y lo coloca en una cola en un mensaje para su procesamiento. El servidor en algún momento en el futuro eliminar el mensaje y enviar el correo electrónico mediante un servidor de correo. El cliente realmente no necesita saber cuando el servidor procesa el mensaje.

Aplicaciones que pueden dividir el procesamiento de una solicitud en un número de pasos discretos, conectados por colas, son una extensión básica del patrón de mensajería simple. Esto es idéntico al patrón de "Pipe y Filter" (ver Buschmann).

Mensajería también proporciona una solución flexible para aplicaciones en las que la conectividad a una aplicación de servidor es transitoria, ya sea debido a la red o falta de fiabilidad servidor. En tales casos, los mensajes se mantienen en la cola hasta que el servidor se conecta y elimina mensajes. Por último, como el Cap. 4 explica, mensajería se puede utilizar para implementar síncrono de solicitud-respuesta utilizando un par de colas de petición-respuesta.

7.2.1.3 publicación-suscripción

Los elementos esenciales del patrón de publicación-suscripción se representan en la Fig. 7.6 . Las propiedades principales de este modelo son:

- Muchos-a-muchos de los mensajes: Publicado mensajes se envían a todos los abonados que están registrados en el tema. Muchos editores pueden publicar sobre el mismo tema, y muchos suscriptores pueden escuchar el mismo tema.
- Con fi gurable QoS: Además de la mensajería no confiable y fiable, el sub acostado mecanismo de comunicación puede ser de punto a punto o de difusión / multidifusión. El primero envía un mensaje distinto para cada suscriptor sobre un tema, éste envía un mensaje que recibe cada suscriptor.
- Bajo acoplamiento: Al igual que con la mensajería, no hay ninguna unión directa entre editoras ERS y suscriptores. Los editores no saben quién recibe su mensaje, y los suscriptores no saben qué editor envía el mensaje.

Mesa 7.5 explica cómo publicación-suscripción soportes atributos de calidad comunes. Arquitecturas basadas en publicación-suscripción son altamente flexible y adecuado para aplicaciones que requieren asíncrona de una sola tomany, muchos-a-uno o mensajería de muchos tomany entre los componentes. Como la mensajería, comunicaciones de dos vías es posible utilizando pares tema de petición-respuesta.

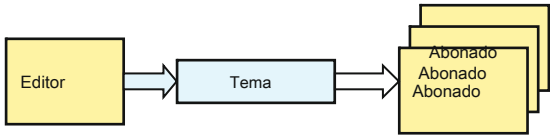


Fig. 7.6 los publicación-suscripción patrón

Tabla 7.5 atributos de calidad para el atributo de calidad de publicación-suscripción patrón

	Cuestiones
Disponibilidad	Temas con el mismo nombre lógico se pueden replicar a través de diferentes servidor instancias gestionan como un clúster. Cuando uno falla, los editores envían mensajes a las colas de réplicas
el control de fallos	Si un editor se está comunicando con un tema alojado en un servidor que falla, puede encontrar un servidor de réplica en vivo y enviar el mensaje de que hay
Modi fi capacidad	Publicación-suscripción está inherentemente imprecisa, y esto promueve alta capacidad de modi fi. Nuevos editores y suscriptores pueden añadirse al sistema sin cambio en la arquitectura o con fi guración. Los cambios en el formato de los mensajes publicados pueden causar cambios en las implementaciones de abonado
Actuación	Publicación-suscripción puede ofrecer miles de mensajes por segundo, con no confiable de mensajería más rápido que fiable. Si un corredor de publicación-suscripción soporta multidifusión / difusión, que entregará varios mensajes en un tiempo más uniforme a cada suscriptor
escalabilidad	Los temas pueden ser replicados a través de grupos de servidores alojados en una sola o varias máquinas de servidor. Los racimos de servidor pueden escalar para proporcionar un rendimiento muy alto volumen de mensajes. Además, las soluciones de multidifusión / difusión escalan mejor que sus contrapartes de punto a punto

7.2.1.4 Broker

Los principales elementos del patrón Broker se muestran en la Fig. 7.7 . Las propiedades de una solución basada en agente son:

- arquitectura hub-and-spoke: El corredor actúa como un centro de mensajería, y los remitentes y los receptores se conectan como radios. Las conexiones con el corredor son a través de los puertos que están asociados con un formato de mensaje especí fi co.
- Lleva a cabo el enrutamiento de mensajes: El corredor incrusta lógica de procesamiento para entregar una mensaje recibido en un puerto de entrada a un puerto de salida. La ruta de entrega puede ser codificado duro o dependen de valores en el mensaje de entrada.
- Lleva a cabo la transformación mensaje: La lógica de intermediario transforma el MeS- fuente Tipo de salvia recibidas en el puerto de entrada para el tipo de mensaje de destino deseado en el puerto de salida.

Mesa 7.6 muestra el apoyo del patrón de atributos de calidad comunes. Los corredores son adecuados para aplicaciones en las que los componentes intercambian mensajes que requieren una amplia transformación entre formatos de origen y de destino. El corredor desacopla el emisor y el receptor, lo que les permite producen o consumen

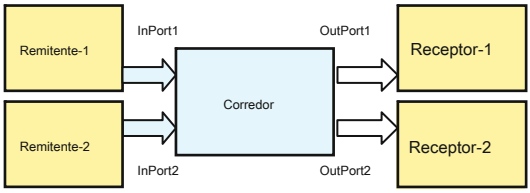


Fig. 7.7 Elementos del patrón broker

Tabla 7.6 atributos de calidad para el atributo de calidad de patrón de

agente	Cuestiones
Disponibilidad	Para construir arquitecturas de alta disponibilidad, los corredores deben replicarse. Esto es normalmente apoyado mediante mecanismos similares a la mensajería y la publicación-suscripción agrupación de servidores
Como el control de fallos corredores han escrito los puertos de entrada, validan y descartar cualquier mensaje	que se envían en un formato incorrecto. Con corredores replicados, los remitentes pueden conmutar por error a un agente activo en caso de una de las réplicas fallar.
Modi fi capacidad	Brokers separan la lógica de transformación y el enrutamiento de mensajes de la remitentes y receptores. Esto mejora la capacidad modi fi, como cambios en la transformación y la lógica de enrutamiento se pueden hacer sin afectar a los remitentes o receptores
Actuación	Los corredores pueden potencialmente convertirse en un cuello de botella, especialmente si tienen que dar servicio grandes volúmenes de mensajes y ejecutar lógica de transformación compleja. Su rendimiento es generalmente más bajos que la simple mensajería con entrega confiable
escalabilidad	La agrupación de los casos de los agentes hace que sea posible la construcción de sistemas escalan a manejar cargas altas de solicitud

su formato de mensaje nativo, y centraliza la definición de la lógica de transformación en el agente para facilitar la comprensión y la modi fi cación.

Coordinador 7.2.1.5 Proceso

El patrón Coordinador de proceso se ilustra en la Fig. 7.8 . Los elementos esenciales de este patrón son:

- Proceso de encapsulación: El coordinador proceso encapsula la secuencia de pasos necesarios para cumplir fi ll el proceso de negocio. La secuencia puede ser arbitrariamente compleja. El coordinador es un único punto de definición para el proceso de negocio, por lo que es fácil de entender y modificar. Se recibe una solicitud de inicio del proceso, llama a los servidores en el orden definido por el proceso, y emite los resultados.
- Bajo acoplamiento: Los componentes de servidor no son conscientes de su papel en la general de procesos de negocio, y del orden de los pasos en el proceso. Los servidores simplemente de definir un conjunto de servicios que se pueden llevar a cabo, y el coordinador de llamadas si es necesario como parte del proceso de negocio.
- Comunicaciones flexibles: Las comunicaciones entre el coordinador y los servidores puede ser síncrono o asíncrono. Para las comunicaciones síncronas, el coordinador de espera hasta que el servidor responde. Para las comunicaciones asíncronas, el coordinador ofrece una devolución de llamada o respuesta cola / tema y espera hasta que el servidor responde utilizando el mecanismo de fi nido.

El patrón coordinador de procesos se utiliza comúnmente para implementar procesos de negocio complejos que deben emitir peticiones a varios componentes de servidor diferentes. Al encapsular la lógica de proceso en un solo lugar, es más fácil de cambiar, gestionar y supervisar el rendimiento del proceso. tecnologías de intermediario de mensajes y Orchestrator de procesos de negocio están diseñados específicamente para apoyar este patrón, la primera de las solicitudes de breve duración, este último para los procesos que pueden tardar varios minutos, horas o días en completarse. En aplicaciones menos complejas, el patrón también es relativamente fácil de implementar sin el apoyo de la tecnología sofisticada, aunque el control de fallos es un asunto que requiere atención cuidadosa.

Mesa 7.7 muestra cómo este patrón se ocupa de los requisitos de calidad.

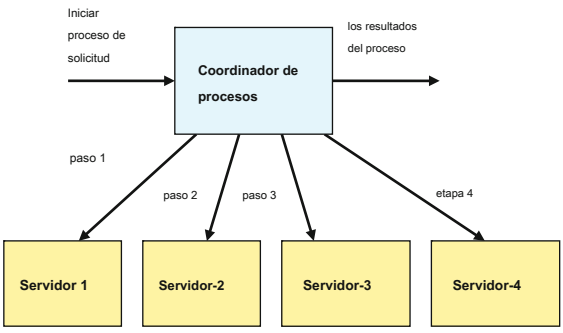


Fig. 7.8 Los componentes del patrón coordinador de procesos

Tabla 7.7 atributos de calidad para el atributo patrón de calidad de coordinador de procesos

	Cuestiones
Disponibilidad	El coordinador es un único punto de fallo. Por lo tanto, necesita ser replicado en crear una solución de alta disponibilidad
el control de fallos	el control de fallos es complejo, ya que puede ocurrir en cualquier etapa en el negocio coordinación proceso. El fallo de un paso posterior del proceso puede requerir pasos anteriores para ser deshecho usando transacciones de compensación. Gestión de Fallos necesita un diseño cuidadoso para asegurar que los datos mantenidos por los servidores se mantiene intacta
Modi fi capacidad	Proceso capacidad modificado se ha mejorado debido a que el proceso de definición de fi encapsulado en el proceso coordinador. Los servidores pueden cambiar su aplicación sin afectar el coordinador u otros servidores, siempre que su servicio externo definición no cambia
Actuación	Para lograr un alto rendimiento, el coordinador debe ser capaz de manejar múltiples solicitudes simultáneas y gestionar el estado de cada medida que progresan a través del proceso. Además, el rendimiento de cualquier proceso estará limitada por la etapa más lenta, a saber, el servidor más lento en el proceso de
escalabilidad	El coordinador puede ser replicado a escala de la aplicación tanto hacia arriba y hacia fuera

7.2.2 Asignar Componentes

Una vez se ha seleccionado un marco global arquitectura, basada en uno o más patrones de arquitectura, la siguiente tarea es para definir los principales componentes que formarán el diseño. El marco de fi ne los patrones generales de comunicación para los componentes. Esto debe ser aumentada por la siguiente:

- La identificación de los componentes principales de la aplicación, y la forma en que se conectan a la marco de referencia.
- La identificación de la interfaz o servicios que soporta cada componente.
- La identificación de las responsabilidades del componente, que indica lo que se puede confiar sobre que hacer cuando se recibe una solicitud.
- La identificación de las dependencias entre los componentes.
- La identificación de las particiones en la arquitectura que son candidatos para su distribución por servidores en una red.

Los componentes de la arquitectura son los principales abstracciones que existirán en la aplicación. Por lo tanto, es probable que no es de extrañar que el diseño de componentes tiene mucho en común con las técnicas de diseño orientado a objetos de uso generalizado. De hecho, los diagramas de clases y paquetes a menudo se utilizan para representar componentes de una arquitectura.

Algunas pautas para el diseño de componentes son:

- Minimizar las dependencias entre los componentes. Luchar por una débilmente acoplado solución en la que cambia a un componente no ondulación a través de la arquitectura, la propagación a través de muchos componentes. Recuerde, cada vez que cambie algo, tiene que volver a probar.

- componentes de diseño que encapsulan un conjunto altamente “cohesivo” de responsabilidades. La cohesión es una medida de lo bien que las partes de un componente fi cio juntos. Componentes altamente cohesivos tienden a tener un pequeño conjunto de responsabilidades bien de fi nidas

que implementan una función lógica individual. Por ejemplo, una EnrollmentReports componente encapsula todas las funciones necesarias para producir informes sobre la matrícula de un estudiante en los cursos. Si los cambios que informe son necesarios formato o tipo, entonces es probable que los cambios serán realizados en este componente. Por lo tanto, una fuerte cohesión limita muchos tipos de cambios a un solo componente, lo que minimiza los esfuerzos de mantenimiento y pruebas.

- Aislar las dependencias de middleware y las tecnologías de infraestructura COTS.

  El menor número de componentes que dependen de especi fi co y middleware COTS llamadas componentes de la API, más fácil es cambiar o actualizar el software intermedio o de otros servicios de infraestructura. Por supuesto, esto requiere más esfuerzo para construir, e introduce una penalización en el rendimiento.

- Utilice la descomposición de estructurar jerárquicamente componentes. El nivel más externa componente define el interfaz a disposición del público para el componente compuesto. Internamente, las llamadas a esta interfaz se delegan en los componentes fi nidas a nivel local, de cuyas interfaces no son visibles desde el exterior.
- Reducir al mínimo las llamadas entre los componentes, ya que pueden resultar costoso si el compo- nentes se distribuyen. Trate de secuencias de agregados de llamadas entre componentes en una sola llamada que puede realizar el procesamiento necesario en una sola solicitud. Esto crea métodos de granos gruesos o servicios en las interfaces que hacen más trabajo por encargo.

Vamos a explorar un estudio de caso simple para ilustrar algunas de estas cuestiones. Figura 7.9 es un ejemplo de una vista estructural de una aplicación de procesamiento de pedidos, definida usando

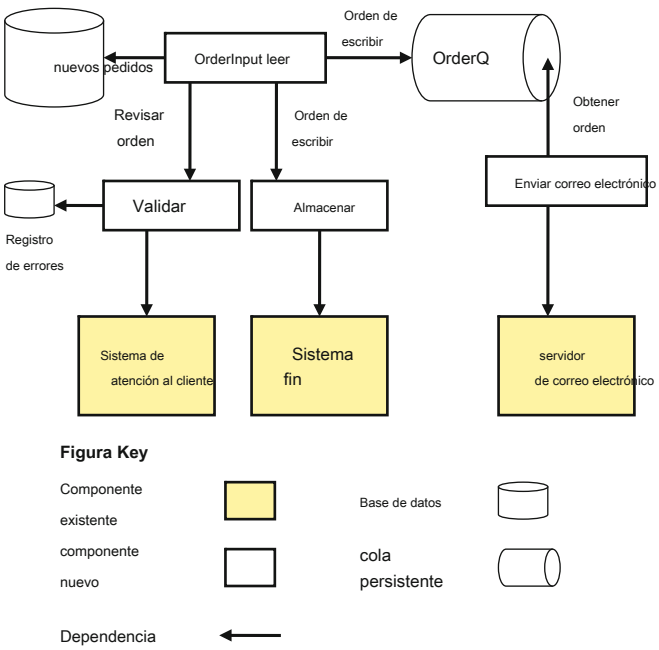


Fig. 7.9 Orden de procesamiento de ejemplo arquitectura

un simple notación informal. Los nuevos pedidos se reciben (de donde es irrelevante) y se cargan en una base de datos. Cada orden debe ser validado en contra de un sistema de datos de clientes existentes para comprobar la información de los clientes y de que existen opciones de pago válidos. Una vez validados, los datos del pedido simplemente se almacena en la base de datos de procesamiento de pedidos, y un correo electrónico se genera al cliente para informarles de que su pedido está siendo procesado.

El marco general de la arquitectura se basa en mensajes sencillos. Los detalles de la orden del cliente se leen de la base de datos, validados, y si es válido, se almacenan en el mensaje de solicitud de pedido y se escriben en una cola. La información sobre cada orden válida se elimina de la cola, con formato de un correo electrónico y se envía al cliente mediante el servidor de correo. Por lo tanto, el uso de una cola de mensajes esta arquitectura desacopla el procesamiento de pedidos del formato de correo electrónico y la entrega.

Cuatro componentes se introducen para resolver este problema. Estos se describen a continuación, junto con sus responsabilidades:

en la construcción de un prototipo que crea un simple 110

- **OrderInput: Esta es responsable de acceder a la nueva base de datos pedidos, encapsulada**

  lating la lógica de procesamiento de pedidos, y la escritura en la cola.
- **Validar: Esto encapsula la responsabilidad de interactuar con el cliente**

  sistema para llevar a cabo la validación, y la escritura a los registros de error si un pedido no es válido.
- **Almacenar: Esto tiene la responsabilidad de interactuar con el sistema para almacenar el**

  datos de los pedidos.
- **Enviar correo electrónico: Esto elimina un mensaje de la cola, da formato a un mensaje de correo electrónico**

  y lo envía a través de un servidor de correo electrónico. Se encapsula todo el conocimiento del formato de correo electrónico y el servidor de correo electrónico de acceso.

Así, cada componente tiene dependencias claras y un pequeño conjunto de responsabilidades, la creación de una arquitectura de acoplamiento flexible y cohesivo. Volveremos a este ejemplo y más analizamos sus propiedades en la siguiente sección, en la que se discute la validación de un diseño de arquitectura.

primero implica esencialmente la prueba manual de la arquitectura usando escenarios de prueba. El segundo consiste

7.3 Validación

Durante el proceso de la arquitectura, el objetivo de la fase de validación es aumentar la con fi anza del equipo de diseño que la arquitectura es fi cio para el propósito. Validar un diseño de arquitectura plantea algunos retos difíciles. Ya se trate de la arquitectura para una nueva aplicación, o una evolución de un sistema existente, el diseño propuesto es, así, sólo que - un diseño. No se puede ejecutar o probado para ver que ful fi LLS sus requisitos. También es probable que constará de nuevos componentes que han de ser construidas, y la caja de negro componentes off-the-shelf tales como middleware y bibliotecas especializadas y las aplicaciones existentes. Todas estas piezas tienen que integrarse y obligados a trabajar juntos.

Por lo tanto, lo que sensatamente se puede hacer? Hay dos técnicas principales que han resultado útiles. El

arquetipo de la aplicación deseada, de modo que su capacidad para satisfacer los requisitos se puede evaluar con más detalle a través de pruebas de prototipo. El objetivo de ambos es identificar potenciales AWS fl y debilidades en el diseño de modo que puedan ser mejorados antes de que comience la ejecución. Estos enfoques deben ser usados para identificar explícitamente las áreas de riesgo potencial para el seguimiento y la monitorización durante las actividades de construcción posteriores.

7.3.1 Uso de Escenarios

Los escenarios son una técnica desarrollada en el SEI para desentrañar las cuestiones relativas a la arquitectura a través de la evaluación manual y pruebas. Los escenarios están relacionados con las preocupaciones arquitectónicas como atributos de calidad, y su objetivo es poner de relieve las consecuencias de las decisiones arquitectónicas que están encapsulados en el diseño.

El trabajo SEI ATAM se describen escenarios y su generación con gran detalle. En esencia, sin embargo, los escenarios son relativamente simples artefactos. Implican la de fi nición algún tipo de estímulo que tendrá un impacto en la arquitectura. El escenario implica entonces la elaboración de cómo la arquitectura responde a este estímulo. Si la respuesta es deseable, a continuación, un escenario se considera que es fi satisfechos por la arquitectura. Si la respuesta no es deseable, o es difícil de cuantificar, a continuación, un fl aw o por lo menos un área de riesgo en la arquitectura puede haber sido descubierto.

Los escenarios pueden ser concebidas para hacer frente a cualquier requisito de calidad de interés en una aplicación dada. Algunos ejemplos hipotéticos generales se muestran en la Tabla 7.8 . Estos escenarios ponen de relieve las implicaciones de las decisiones de diseño de arquitectura en el contexto del estímulo y los efectos que provoca. Por ejemplo, el escenario de “disponibilidad” muestra que los mensajes se pueden perder si un servidor falla antes de entrega del mensaje. La implicación aquí es que los mensajes no se conservan en el disco, muy probablemente por razones de rendimiento. La pérdida de mensajes en algunos contextos de aplicación puede ser aceptable. Si no es así, esta situación pone de relieve un problema, lo que puede forzar el diseño de adoptar la mensajería persistente para evitar la pérdida de mensajes.

Veamos algunos ejemplos más concretos fi cas para el ejemplo de procesamiento de pedidos introducido en la sección anterior. El diseño en la fig. 7.9 necesita ser validado, y los escenarios de la tabla 7.9 sondear más profundamente en la arquitectura, en busca exponer AWS fl o áreas de riesgo.

Los primeros dos escenarios parecen provocar respuestas positivas en el diseño. los Validar componente limita los cambios necesarios para acomodar una nueva base de datos de clientes, y por lo tanto aísla otros componentes del cambio. Y si el servidor de correo electrónico no esté disponible, la implicación es que los correos electrónicos no son más que retrasarse hasta que el servidor de correo electrónico devoluciones.

El fracaso de la Cliente o Orden aplicaciones es más revelador, sin embargo. Las comunicaciones con estos dos sistemas es sincrónica, por lo que si bien no está disponible, el procesamiento de pedidos debe detener hasta que las aplicaciones se restauran. Esto puede ser menos que deseable.

Tabla 7.8 ejemplos de escenarios de atributos

de calidad de estímulo		Respuesta
Disponibilidad	La conexión de red para los consumidores de mensajes falla	Los mensajes se almacenan en el servidor MOM hasta la conexión se restablece. Mensajes sólo se perderán si falla el servidor antes de la conexión vuelve a subir
Modi fi capacidad	Un nuevo conjunto de análisis de datos componentes deben estar disponibles en la aplicación	La aplicación necesita ser reconstruido con el nuevo bibliotecas, y el todo con fi guración fi les debe actualizarse en cada escritorio para hacer los nuevos componentes visibles en la caja de herramientas de interfaz gráfica de usuario
Seguridad	no se reciben solicitudes en una sesión de usuario durante 10 min	El sistema trata esta sesión como potencialmente insegura e invalida las credenciales de seguridad asociados con la sesión. El usuario debe iniciar sesión de nuevo para conectarse a la aplicación
Modi fi capacidad	El proveedor de la motor de transformación va a la quiebra	Un nuevo motor de transformación debe estar comprar. La capa de servicio abstracto que envuelve el componente motor de transformación debe ser reimplantado para apoyar el nuevo motor. componentes de cliente no se ven afectados, ya que sólo utilizan la capa de servicio abstracto
escalabilidad	La petición de usuario concurrente la carga se duplica durante el período de inscripción 3 semanas	El servidor de aplicaciones se escala a cabo en una de dos clúster máquina para manejar la carga solicitud aumentado

Tabla 7.9 Los escenarios para el atributo de calidad ejemplo el procesamiento de

pedidos	Estímulo	Respuesta
Modi fi La capacidad	Sistema de atención al cliente envasada aplicación se actualiza a una base de datos Oracle	los Validar componente debe ser reescrito para interfaz con el sistema Oracle
Disponibilidad	El servidor de correo electrónico falla	Los mensajes se acumulan en el OrderQ hasta el reinicia el servidor de correo electrónico. Los mensajes son enviados por el Enviar correo electrónico componente para eliminar el retraso. El procesamiento de pedidos no se ve afectada
Confiabilidad	los Cliente o Orden sistemas no están disponibles	Si bien falla, se detiene el procesamiento de pedidos y alertas son enviados a los administradores del sistema para que el problema puede ser fijo

Nota el diseño no discrimina entre las interacciones con las dos aplicaciones. Es bastante obvio sin embargo que la interacción con el Sistema de atención al cliente requiere una respuesta diciendo si los datos de orden es válida. Si no es así, está escrito en un registro de error y el procesamiento de pedidos cesa para ese fin. los Sistema fin aunque simplemente almacena los datos de la orden para el procesamiento posterior. No hay necesidad de que el Almacenar componente que requiere una respuesta inmediata.

Por lo tanto, el escenario fiabilidad ha puesto de relieve un área donde la arquitectura se podría mejorar. Un pedido no puede ser procesada hasta que haya sido validado con éxito, por lo que una respuesta de la Sistema de atención al cliente es necesario. Si no está disponible, el proceso no puede continuar.

Pero el Sistema fin es un asunto diferente. comunicaciones asíncronas es mejor en este caso. Almacenar sólo pudiera escribir en una cola persistente, y el procesamiento de pedidos puede continuar. Otro componente podría introducirse después de leer el orden de la cola y añadir los datos a la Sistema orden. Esta solución es más resistente a un fallo, como el Sistema fin puede no estar disponible, pero el procesamiento de pedidos puede continuar.

7.3.2 Prototipos

Los escenarios son una técnica muy útil para validar una arquitectura propuesta. Sin embargo, algunos escenarios no son tan fáciles de responder basándose únicamente en una descripción del diseño. Consideremos un escenario de rendimiento para el sistema de procesamiento de pedidos:

> El viernes por la tarde, los pedidos deben ser procesados antes del cierre de negocio para asegurar la entrega para el lunes. Cinco mil pedidos llegan a través de diversos canales (Web / Call socios / centro de negocios) cinco minutos antes del cierre de negocio.

La pregunta aquí es, entonces, simplemente, se las 5.000 órdenes serán procesadas en 5 minutos? Esta es una pregunta difícil de responder cuando algunos de los componentes de la solución no existan aún.

La única manera de abordar estas cuestiones con algún grado de confianza es construir un prototipo. Los prototipos son mínimos, restringido o versiones de la aplicación deseada de corte hacia abajo, creado específicamente para probar algo de riesgo alto o aspectos poco conocidos del diseño. Los prototipos se utilizan típicamente para dos propósitos:

1. Prueba de concepto: ¿Puede la arquitectura como diseñó ser construido de una manera que puede satisfacer los requisitos?
2. La prueba de la tecnología: ¿La tecnología (middleware, aplicaciones integradas, bibliotecas, etc.) seleccionados para implementar la aplicación se comporta como se esperaba?

En ambos casos, los prototipos pueden proporcionar evidencia concreta acerca de las preocupaciones que de otro modo difícil, si no imposible de validar en cualquier otra forma.

Para responder a nuestro escenario de rendimiento por encima, ¿qué clase de prototipo podríamos construir? La respuesta general es uno que incorpora todas las operaciones sensibles de rendimiento en el diseño, y que se ejecuta en una plataforma lo más similar posible (idealmente idéntica) a la de la aplicación se implementa en.

Por ejemplo, el arquitecto podría saber que los sistemas de colas y de correo electrónico son fácilmente capaces de soportar 5.000 mensajes en 5 min, ya que estas soluciones se utilizan en otra aplicación similar. por lo tanto, no habría necesidad de construir esto como parte del prototipo. Sin embargo, el rendimiento de las interacciones entre la Cliente y

Orden aplicaciones utilizando sus APIs son un desconocido, y por lo tanto estos dos deben ser probados para ver si pueden procesar 5.000 mensajes en 5 min. La forma más sencilla de hacer esto es:

- Escribir un programa que llama a la prueba Sistema de atención al cliente API de validación 5.000 veces, y el tiempo de duración de esta operación.
- Escribir un programa que llama a la prueba Sistema fin tienda de API 5000 veces, y el tiempo duración de esta operación.

una amplia documentación. 114

Una vez que los prototipos han sido creado y probado, la respuesta de la arquitectura al estímulo en el escenario se puede responder con un alto grado de confianza.

Los prototipos deben utilizarse juiciosamente para ayudar a reducir los riesgos inherentes a un diseño. Ellos son la única manera de que las preocupaciones relacionadas con el rendimiento, escalabilidad, facilidad de integración y requisitos y los equipos de diseño, la recopilación de las aportaciones de las distintas partes interesadas, y la producción de capacidades de los componentes off-the-shelf pueden abordarse con algún grado de certeza.

A pesar de su utilidad, una palabra de precaución en la creación de prototipos es necesario. los esfuerzos de creación de prototipos deben ser cuidadosamente scoped y gestionados. Lo ideal sería que un prototipo debe desarrollarse en uno o dos días, una semana o dos a lo sumo. La mayor prueba de la tecnología y prototipos de prueba de concepto botados documentación. Para grandes proyectos, el proceso puede ser seguido de manera más formal, con la participación de los después de que hayan cumplido su propósito. Son un medio para un fin, así que no deje a adquirir una vida propia y se convierten en un fin en sí mismos.

construir el proyecto. En estos proyectos, el proceso puede ser seguido de manera informal, produciendo un mínimo de

7.4 Resumen y lectura adicional

El diseño de una arquitectura de aplicaciones es una actividad intrínsecamente creativo. Sin embargo, siguiendo un proceso simple que capta explícitamente los requisitos cativos arquitectónicamente signi fi exploits conocidos innovadores). El arquitecto también es probable que sea una parte importante del pequeño equipo de desarrollo que va a patrones de arquitectura y sistemáticamente valida el diseño, algunos de la mística de diseño puede estar expuesto.

El proceso de tres pasos descrito en este capítulo es inherentemente iterativo. El diseño inicial se valida con los requisitos y escenarios, y el resultado de la validación puede causar los requisitos o el diseño para ser revisado. La iteración continúa hasta que todas las partes interesadas están contentos con la directamente con el cliente, o puede ser, de hecho ningún cliente tangible (a menudo en el desarrollo de productos nuevos e arquitectura, que luego se convierte en el modelo a partir del cual comienza el diseño detallado. En proyectos ágiles, iteraciones son cortos, y las implementaciones concretas de la arquitectura resultado de cada iteración.

El proceso es también escalable. Para proyectos pequeños, el arquitecto puede estar trabajando en su mayoría

Por supuesto, existen otros procesos de arquitectura, y probablemente el más ampliamente utilizado es el Rational Uni Proceso fi ed (RUP). Una buena referencia para RUP es:

P. Kruchten. El Racional Unificado de proceso: Una introducción (2ª edición). Addison-Wesley, 2000

La fuente más completa de información sobre métodos y técnicas para la evaluación de la arquitectura es:

P. Clements, R. Kazman, M. Klein. La evaluación de arquitecturas de software: Métodos y Estudios de Casos. Addison-Wesley, 2002

Esto describe el proceso de ATAM, y proporciona excelentes ejemplos que ilustran el enfoque. Su enfoque está evaluando sistemas grandes y complejos, pero muchas de las técnicas son adecuadas para aplicaciones de menor escala.

Capítulo 8

La documentación de una arquitectura de software

8.1 Introducción

documentación de la arquitectura es a menudo un tema espinoso en los proyectos de TI. Es común para que haya poca o ninguna documentación que cubre la arquitectura en muchos proyectos. A veces, si hay alguna, es la fecha fuera de, inapropiada y, básicamente, no es muy útil.

En el otro extremo hay proyectos que tienen masas de información relacionada con la arquitectura capturado en diversos documentos y herramientas de diseño. A veces esto es muy valiosa, pero a veces es fuera de fecha, inadecuado y no es muy útil!

Es evidente, entonces, la experiencia nos dice que la documentación de arquitecturas no es una tarea sencilla. Pero hay muchas buenas razones por las que queremos documentar nuestras arquitecturas, por ejemplo:

- Otros pueden comprender y evaluar el diseño. Esto incluye cualquiera de la aplicabilidad interesados ción, pero con mayor frecuencia que otros miembros del equipo de diseño y desarrollo.
- Podemos entender el diseño cuando volvamos a él después de un período de tiempo.
- Otros en la organización del equipo y desarrollo de proyectos pueden aprender de la arquitectura digiriendo el pensamiento detrás del diseño.
- Podemos hacer el análisis en el diseño, tal vez para evaluar su rendimiento probable, o para generar métricas estándar como acoplamiento y cohesión.

La documentación de arquitecturas es problemático, sin embargo, debido a que:

- No hay un estándar universalmente aceptado documentación de la arquitectura.
- Una arquitectura puede ser complejo, y documentarla en un comprensible manera es mucho tiempo y no trivial.
- Una arquitectura tiene muchos puntos de vista posibles. La documentación de todos los potencialmente útil queridos es largo y costoso.
- Una arquitectura menudo evoluciona a medida que el sistema se desarrolló de forma incremental y más ideas sobre el dominio del problema se obtienen. Mantener los documentos arquitectura actual es a menudo pasado por alto una actividad, especialmente con presiones de tiempo y horario en un proyecto.

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3_8, # Springer-Verlag Berlin Heidelberg 2011

Estoy bastante seguro de las herramientas utilizadas para la documentación predominantes son la arquitectura de Microsoft Word, Visio y PowerPoint, junto con sus equivalentes no son de Microsoft. Y la notación de diseño más utilizado es el “bloque y flecha” informal diagramas, al igual que hemos utilizado en este libro hasta ahora, de hecho. Estos dos hechos son un poco de una acusación sobre el estado de las prácticas de documentación de arquitectura en la actualidad. Debemos ser capaces de hacerlo mejor.

En este capítulo se examinan algunos de los puntos de vista de arquitectura más útiles para documentar, y muestra como la última encarnación de la Unificado de Lenguaje de Modelado, UML v2.0, puede ayudar con la generación de estos puntos de vista. El uso de estas técnicas y herramientas de apoyo, no es excesivamente culto fi cultad o costoso para generar documentación útil y valiosa.

8.2 ¿Qué pasa al documento

Probablemente el elemento más importante de la ecuación “qué documento” es la complejidad de la arquitectura está diseñando. Una aplicación cliente servidor de dos niveles con la lógica de negocio complejas puede ser en realidad bastante simple vista arquitectónico. Se puede requerir no más de un diagrama general “marketeture”, que describe los componentes principales, y tal vez un punto de vista estructural de los componentes principales (tal vez que utiliza una arquitectura modelo-vista-controlador) y una descripción del esquema de base de datos, genera ninguna duda automáticamente por herramientas de bases de datos. Este nivel de documentación es rápida para producir y la rutina de describir.

Otro factor a considerar es la probable duración de la aplicación. ¿El sistema de servir una función de negocio a largo plazo, o está siendo construida para manejar un hecho aislado necesidad de integración, o es sólo un recurso provisional hasta que se instale un paquete ERP completo? Los proyectos con pocas posibilidades de una vida larga, probablemente no necesitan una gran cantidad de documentación. Sin embargo, no deje que esto sea una excusa para cortar juntos un cierto código y tirar buenas prácticas de diseño al viento. A veces, estos sistemas para detener la brecha tienen la costumbre de vivir por mucho más tiempo de lo previsto inicialmente, y alguien (tal vez incluso usted) podría pagar por estos hacks 1 día.

El siguiente factor a considerar es las necesidades de los diferentes actores del proyecto. La documentación de la arquitectura desempeña un papel importante la comunicación entre los distintos miembros del equipo de proyecto, incluyendo arquitectos, diseñadores, desarrolladores, probadores, gestión de proyectos, clientes, organizaciones asociadas, y así sucesivamente. En un pequeño equipo, la comunicación interpersonal con frecuencia es bueno, por lo que la documentación puede ser mínimo, y tal vez incluso mantenido en una o dos pizarra utilizando técnicas de desarrollo ágil. En equipos más grandes, y sobre todo cuando los grupos no son de ubicación conjunta en el mismo o fi cinas o edificio, la documentación de la arquitectura se vuelve de vital importancia para la descripción de los elementos de diseño tales como:

- interfaces de componentes
- restricciones subsistemas
- Los escenarios de prueba

- las decisiones de compra de componentes de terceros
- la estructura del equipo y del horario dependencias
- servicios externos a ser ofrecidos por la aplicación

Por lo tanto, no hay una respuesta simple aquí. Documentación necesita tiempo para desarrollarse, y cuesta dinero. Por lo tanto es importante pensar cuidadosamente acerca de qué documentación va a ser más útil dentro del contexto del proyecto, y producir y mantener estos documentos de referencia como clave para el proyecto.

8.3 UML 2.0

También está la cuestión de cómo documentar una arquitectura. Hasta ahora, en este libro hemos utilizado diagramas de cajas y flechas simples, con una clave diagrama apropiado para dar un significado claro a la notación utilizada. Esto se ha hecho deliberadamente, como en mi experiencia, anotaciones esquemáticas informales son el vehículo más común utilizado para documentar las arquitecturas de aplicaciones de TI.

Por supuesto, hay muchas maneras de describir los diversos puntos de vista de arquitectura que pueden ser útiles en un proyecto. Afortunadamente para todos nosotros, hay un excelente libro que describe muchos de estos de Paul Clements et al. (Ver lecturas adicionales), por lo que no se hará ningún intento para replicar eso. Pero ha habido un desarrollo significativo, ya que el libro fue publicado, y eso es la aparición de la Uni fi ed Modeling Language (UML) 2.0.

A pesar de sus puntos fuertes y débiles en gran medida debatidos, el UML se ha convertido en el lenguaje de descripción de software predominante utilizado en toda la gama de dominios de desarrollo de software. Tiene amplia y ahora soporte de herramientas de calidad y bajo costo, y por lo tanto es fácilmente accesible y utilizable para arquitectos de software, diseñadores, desarrolladores, estudiantes - todos en realidad.

UML 2.0 es una actualización importante del lenguaje de modelado. Se añade varias características nuevas y, signi fi cativamente, se formaliza muchos aspectos de la lengua. Esta formalización ayuda de dos maneras. Para los diseñadores, que elimina la ambigüedad de los modelos, lo que ayuda a aumentar la comprensibilidad. En segundo lugar, es compatible con el objetivo del desarrollo basado en modelos, en los que se utilizan modelos UML para la generación de código. También hay un gran debate sobre la utilidad de desarrollo dirigido por modelos, y este tema es especí fi camente cubierto en un capítulo posterior, por lo que no vamos a ahondar en él ahora.

Los UML 2.0 notaciones de modelado cubren tanto los aspectos estructurales y de comportamiento de los sistemas de software. La estructura Diagramas de fi ne la arquitectura estática de un modelo, y camente especificaciones son:

- Los diagramas de clases: Mostrar las clases en el sistema y sus relaciones.
- diagramas de componentes: Describir la relación entre los componentes con bienestar de fi ne las interfaces. Componentes comprenden típicamente múltiples clases.
- Los diagramas de paquetes: Dividir el modelo en grupos de elementos y describir la dependencias entre ellos a un alto nivel.

- Los diagramas de despliegue: Mostrar cómo los componentes de software y otros artefactos como procesos se distribuyen a hardware físico.
- Los diagramas de objetos: Muestran cómo los objetos están relacionados y utilizados en tiempo de ejecución. Estos son a menudo llamados diagramas de instancia.
- diagramas de estructura de material compuesto: Mostrar la estructura interna de las clases o componentes en términos de sus objetos compuestos y sus relaciones.

Por el contrario, los diagramas de comportamiento muestran las interacciones y los cambios de estado que se producen como elementos en el modelo ejecutan:

- diagramas de actividad: Similares a los gráficos de fluencia, y se utiliza para de fi nir la lógica del programa y los procesos de negocio.
- diagramas de comunicación: diagramas de colaboración denominados en UML 1.x, se describir la secuencia de llamadas entre los objetos en tiempo de ejecución.
- Los diagramas de secuencia: A menudo llamados diagramas de natación carriles después de su tiempo- verticales líneas, muestran la secuencia de mensajes intercambiados entre los objetos.
- diagramas de máquina de estados: Describir el funcionamiento interno de un objeto, mostrando sus estados y eventos, y las condiciones que causan las transiciones de estado.
- Descripción interacción diagramas: Estos son similares a los diagramas de actividad, pero puede incluir otros diagramas de interacción UML, así como actividades. Están diseñados para mostrar el control de flujo a través de una serie de escenarios más sencillos.
- Diagramas de tiempo: Estos se combinan esencialmente de secuencias y diagramas de estado a describir diversos estados de un objeto a través del tiempo y los mensajes que alteran el estado del objeto.
- Los diagramas de casos: Estas interacciones entre el sistema de captura y su medio ambiente, incluidos los usuarios y otros sistemas.

interacción con un sistema existente. Sin duda, otras clases serán 120

Es evidente, entonces, UML 2.0 es una gran área técnica en sí mismo, y algunos indicadores de buenas fuentes de información se proporcionan al final de este capítulo. En las siguientes secciones sin embargo, vamos a describir algunos de los modelos más útiles UML 2.0 para la representación de arquitecturas de software.

8.4 Arquitectura Vistas

Volvamos al ejemplo de procesamiento de pedidos introducido en el capítulo anterior. Figura 7.9 muestra una descripción informal de la arquitectura utilizando una caja y flecha notación. En la Fig. 8.1 , Un diagrama de componentes UML se utiliza para representar una vista estructural equivalente de la arquitectura del sistema de procesamiento de pedidos. Nótese, sin embargo, sobre la base de la evaluación en el capítulo anterior, una cola se ha añadido a la comunicación entre el Procesando orden y OrderSystem componentes.

Sólo dos de los componentes de la arquitectura requieren código de nuevo sustancial que se creará. La estructura interna de las más complejas de estos, Procesando orden, se muestra en el diagrama de clases en la figura. 8.2 . Incluye tres clases que encapsulan cada

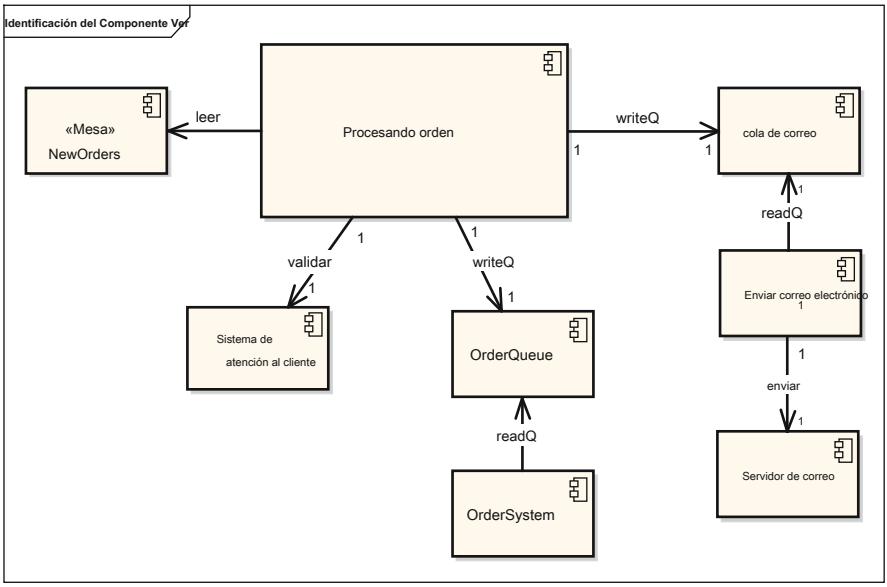


Fig. 8.1 Un diagrama de componentes UML para el ejemplo de procesamiento de pedidos

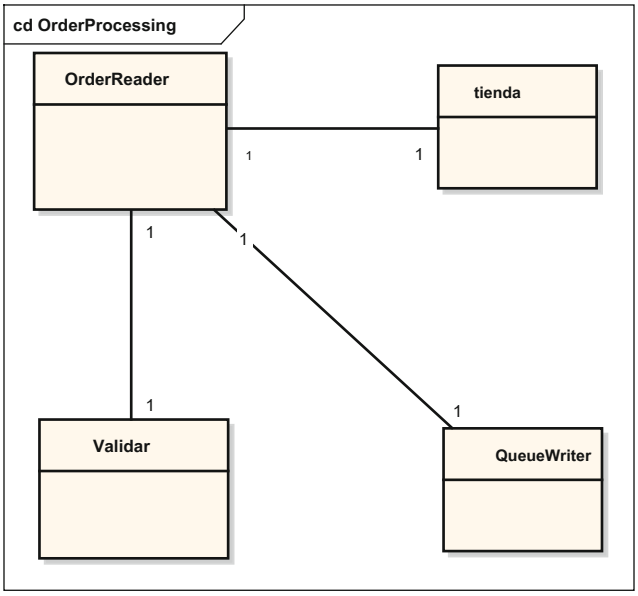


Fig. 8.2 Las clases para el componente de procesamiento de pedidos

introducidos en el diseño ya que se implementa, por ejemplo, uno para representar un nuevo orden, pero estos no se muestran en el diagrama de clases para que no se desordenan con detalles innecesarios. Estos son los detalles de diseño no necesarios en una descripción de la arquitectura.

Con este nivel de descripción, podemos ahora crear un diagrama de secuencia que muestra las principales interacciones entre los elementos arquitectónicos. Esto se muestra en la figura. 8.3 , Que utiliza los estereotipos UML estándar para representar Límite (CustomerSystem, OrderQueue, cola de correo) y Entidad (NewOrder) componentes. Este diagrama de secuencia omite el comportamiento cuando un nuevo orden no es válida, y lo que sucede una vez que los mensajes han sido colocados en el OrderQueue y Cola de correo.

Una vez más, esto mantiene el modelo ordenado. Descripciones de esta funcionalidad adicional o bien podrían ser descritos en posteriores (muy simples) diagramas de secuencia, o simplemente en el texto que acompaña el diagrama de secuencia.

Los diagramas son probablemente la técnica más útil en el UML para modelar el comportamiento de los componentes de una arquitectura. Uno de sus puntos fuertes en realidad se encuentra, irónicamente, en su debilidad inherente en la descripción de un procesamiento complejo y la lógica. Aunque es posible representar bucles y la selección en los diagramas de secuencia, se convierten rápidamente en difícil de entender y difícil de manejar para crear. Esto anima a los diseñadores para mantenerlos relativamente simple, y se centran en la descripción de las principales interacciones entre los elementos signi fi cativos arquitectónicamente en el diseño.

Muy a menudo en este tipo de proyecto de integración de negocio, es posible crear un diagrama de despliegue UML muestra dónde los diversos componentes se ejecutarán.

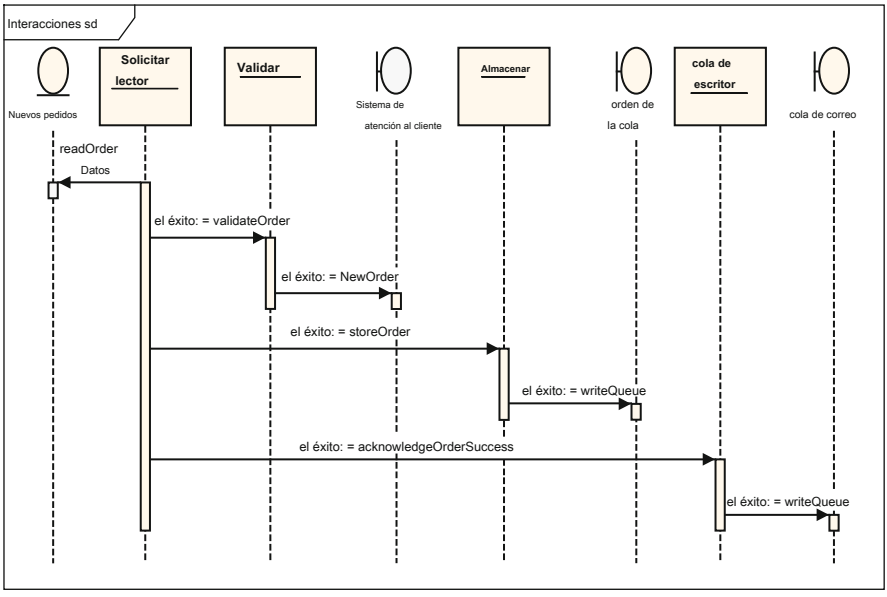


Fig. 8.3 diagrama de secuencia para el sistema de procesamiento de orden 122

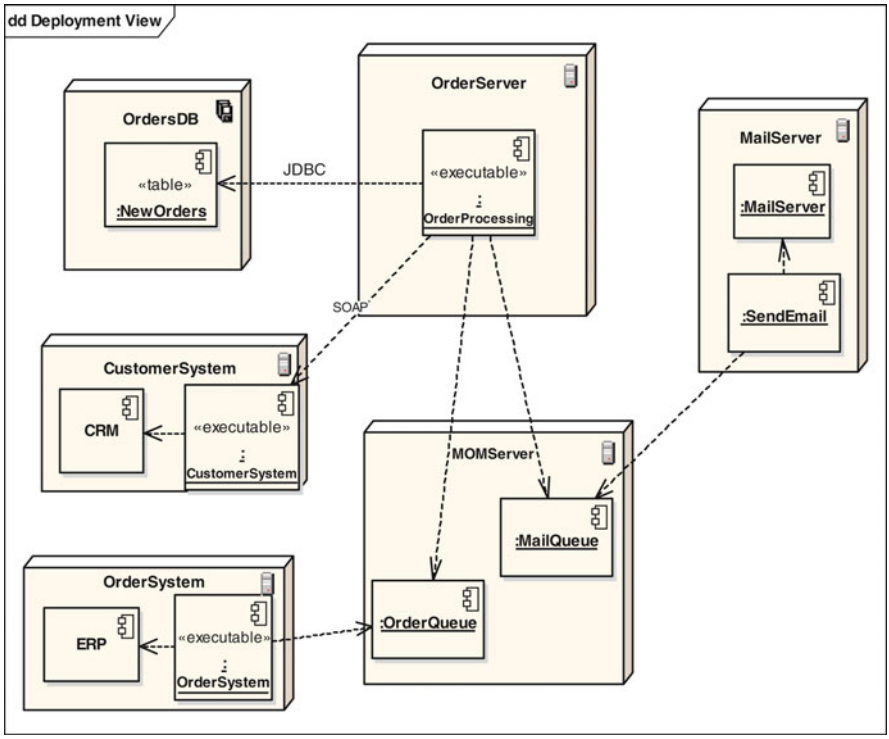


Fig. 8.4 diagrama UML de implementación para el sistema de procesamiento de pedidos

Esto se debe a que muchos de los componentes en el diseño ya existen, y el arquitecto debe mostrar cómo los nuevos componentes interactúan con éstos en el entorno de despliegue. Un ejemplo de un diagrama de despliegue UML para este ejemplo se da en la Fig. 8.4 . Se asigna componentes a los servidores y muestra las dependencias entre los componentes. A menudo es útil para etiquetar las dependencias con un nombre que indica el protocolo que se utiliza para la comunicación entre los componentes. Por ejemplo, el Procesando orden componente ejecutable requiere JDBC [1] para acceder a la Nuevos pedidos mesa en el OrdersDB base de datos.

8,5 Más sobre Los diagramas de componentes

diagramas de componentes son muy útiles para esbozar la estructura de una arquitectura de aplicaciones. En ellos se describen claramente las principales partes del sistema, y pueden mostrar qué tecnologías off-the-shelf serán utilizados, así como los nuevos componentes que

[1] Java Database Connectivity.

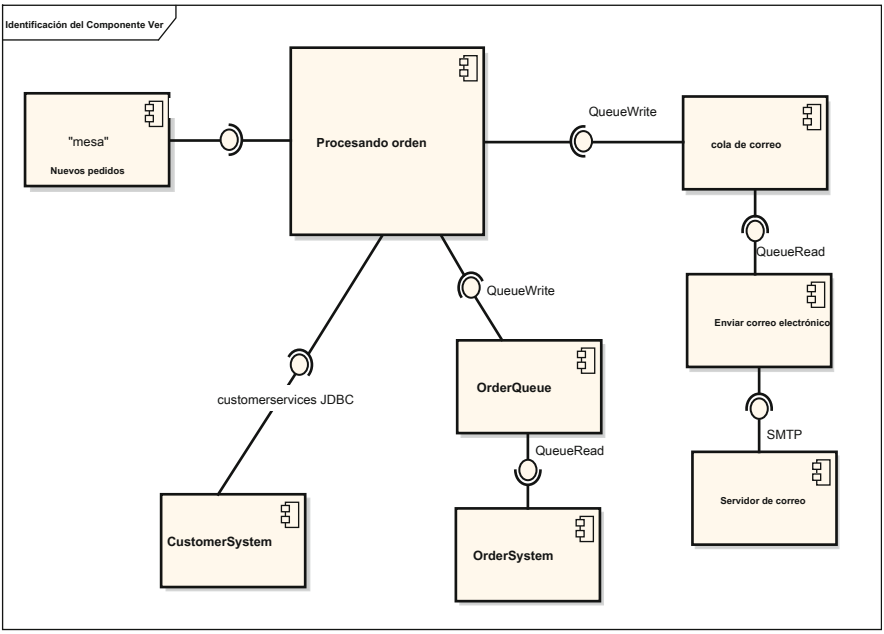


Fig. 8.5 En representación de las interfaces en el ejemplo de procesamiento de pedidos 124

deben construirse. UML 2.0 también ha introducido mejoradas notaciones para la representación de interfaces de componentes. Una interfaz es una colección de métodos que soporta un componente. Además de la UML 1.x notación “piruleta” para la representación de una interfaz con el apoyo de un componente (un “proporcionado” interfaz), la notación “socket” se puede utilizar para especificar que un componente necesita una interfaz particular a ser apoyado por su medio ambiente (un “requerida” interfaz). Estos se ilustran en la Fig. 8.5 . Interfaz de fi nición es particularmente importante en una arquitectura, ya que permite a los equipos independientes de los desarrolladores para diseñar y construir sus componentes de manera aislada, asegurando que se apoyan los contratos definida por sus interfaces.

Mediante la conexión de interfaces proporcionadas y requeridas, los componentes pueden ser “conectados” o “cableados” juntos, como se muestra en la Fig. 8.5 . Las interfaces proporcionadas se nombran, y capturan las dependencias entre los componentes. Los nombres de interfaces deben corresponder a los utilizados por las aplicaciones off-the-shelf en uso, o interfaces de los componentes de la cantera existentes.

UML 2.0 permite refinar la interfaz de fi niciones aún más, y representan la forma en que se apoyan en el contexto de un componente. Esto se hace mediante la asociación de las interfaces con los “puertos”. Puertos de fi ne un opcionalmente llamado único punto, la interacción entre un componente y su entorno exterior. Están representados por pequeños cuadrados en el borde del componente, y tienen uno o más proporciona o requiere interfaces asociadas con ellos.

La arquitectura del sistema de procesamiento de pedidos a través de puertos para el Procesando orden y CustomerSystem componentes se representa en la Fig. 8.6 . Todos los puertos en este diseño

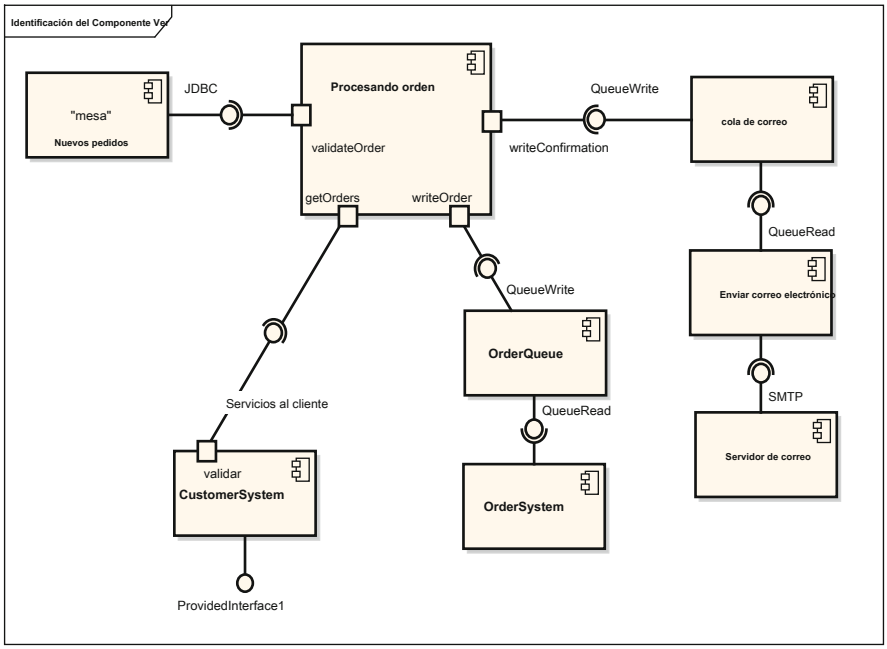


Fig. 8.6 Utilización de los puertos en el ejemplo de procesamiento de pedidos

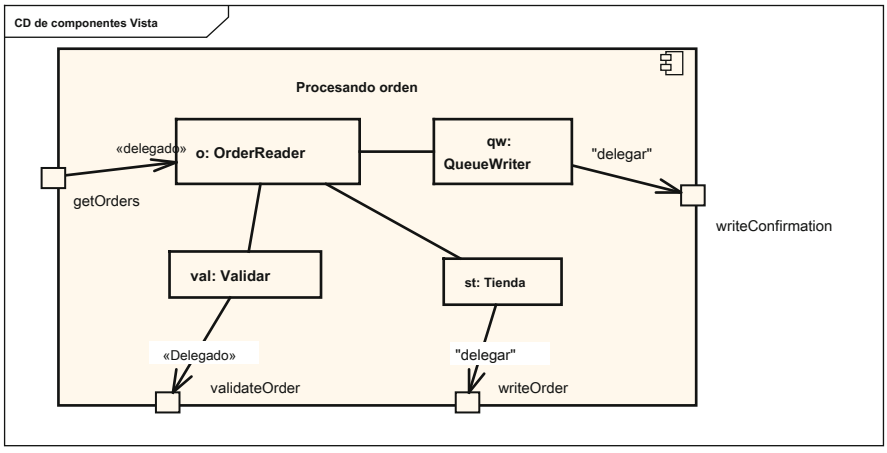


Fig. 8.7 El diseño interno de la Procesando orden componente

son unidireccionales, pero no hay nada que les impida ser bidireccional en términos de apoyo a uno o más proporcione o requiere interfaces. UML 2.0 diagramas compuestos nos permiten mostrar la estructura interna de un elemento de diseño tal como un componente. Como se muestra en la Fig. 8.7 , Podemos describir explícitamente que los objetos comprenden

la implementación del componente, y la forma en que se relacionan entre sí y con los puertos del componente apoya. Los objetos internos están representados por UML 2.0 “partes”. Piezas se definen en UML 2.0 como instancias de tiempo de ejecución de las clases que son propiedad de la clase o componente que contiene. Las piezas son unidas por conectores y describen con fi guraciones de instancias que se crean dentro de una instancia del componente / clase que contiene.

diagramas compuestos son útiles para describir el diseño de componentes complejos o importantes en un diseño. Por ejemplo, una arquitectura en capas podría describir cada capa como un componente que soporta varios puertos / interfaces. Internamente, una descripción de la capa puede contener otros componentes y piezas que muestran cómo se apoya cada puerto. Los componentes también pueden contener otros componentes, por lo arquitecturas jerárquicas pueden ser descrito fácilmente. Veremos algunas de estas técnicas de diseño en el estudio de caso en la siguiente sección.

8.6 Plantilla de Documentación de Arquitectura

Siempre es útil para una organización contar con una plantilla de documento disponible para la captura de la documentación específica fi proyecto. Plantillas reducen el tiempo de puesta en marcha de proyectos, proporcionando estructuras de documentos confeccionados para los miembros del proyecto para usar.

Una vez que el uso de las plantillas se institucionaliza, la familiaridad adquirida con las ayudas de estructura de documento en la e fi ciencia de captura de los detalles del proyecto de diseño.

Arquitectura Documentación de la plantilla Nombre del proyecto: 1 XXX

Contexto del Proyecto 2

Requisitos Arquitectura

2.1 Resumen de los objetivos clave

2.2 Arquitectura de casos de uso

2.3 Requisitos de arquitectura de las partes interesadas

2.4 Limitaciones

2.5 Requisitos no funcionales

2.6 Riesgos

3 Solución

3.1 Patrones arquitectónicos relevantes

3.2 Descripción de la arquitectura

3.3 Vistas estructurales

3.4 Comportamiento Vistas

3.5 Cuestiones de Aplicación 4

Análisis de la arquitectura

análisis 4.1 Escenario

4.2 Riesgos

Fig. 8.8 Arquitectura documentación contorno 126

Las plantillas también ayudan a la formación de nuevo personal, ya que los desarrolladores dicen lo que emite la organización les obliga a considerar y pensar en la producción de su sistema.

Figura 8.8 muestra la estructura encabezamientos para una plantilla de documentación que puede ser utilizado para la captura de un diseño de la arquitectura. Para implementar esta plantilla en una organización, que debe ir acompañada de un texto explicativo y ejemplos de lo que la información que se espera en cada sección. Sin embargo, en vez de hacer eso aquí, esta estructura de plantilla se utilizará para mostrar la solución al problema de estudio de caso ICDE en el siguiente capítulo.

8.7 Sumario y lectura adicional

La generación de la documentación arquitectura es casi siempre una buena idea. El truco es pasar suficiente esfuerzo para producir única documentación que será útil para los diferentes grupos de interés del proyecto. Esto toma un poco de planificación por adelantado y el pensamiento. Una vez que se establece un plan de documentación, los miembros de equipo deben comprometerse a mantener la documentación razonablemente actual, precisa y accesible.

Estoy un poco de un partidario de la utilización de notaciones y herramientas basadas en UML para la producción de documentación de arquitectura. El UML, especialmente con la versión 2.0, hace que sea bastante sencillo para documentar diversos puntos de vista estructurales y de comportamiento de un diseño. Herramientas hacen que la creación del diseño rápido y fácil, y también hacen posible la captura de gran parte de la lógica de diseño, las restricciones de diseño, y otra documentación basada en texto dentro del repositorio de la herramienta. Una vez que está en el repositorio, generación de documentación de diseño se convierte en una simple tarea de seleccionar el elemento de menú correcto y la apertura de un navegador o caminar a la impresora. Dicha producción documentación automática es un truco que está garantizado para impresionar a las partes interesadas no técnicos, e incluso a veces el técnico extraño!

Además, es posible utilizar fl exible UML 2.0 en un proyecto. Se puede utilizar para esbozar una representación abstracta de arquitectura, puramente para fines de comunicación y documentación. También se puede utilizar para modelar estrechamente los componentes y objetos que serán realizadas en la implementación real. Esta “cercanía” puede reducirse aún más en el caso extremo a “exactitud”, en el que se utilizan elementos en el modelo para generar código ejecutable. Si usted está haciendo esto, entonces usted está haciendo el desarrollo dirigido por modelos denominado (MDD).

Hay todo tipo de debates estragos sobre el valor y la importancia de utilizar el UML informalmente en comparación con el uso preciso requerido por el TDM. De vuelta en el Cap. 1, se examinó el papel de una arquitectura de software como una representación abstracta del sistema. La abstracción es una potente ayuda para la comprensión, y si nuestra representación arquitectura es abstracto, entonces se argumenta a favor de un uso más informal del UML en nuestro diseño. Por otra parte, si nuestros modelos UML son una representación precisa de nuestra aplicación, a continuación, que apenas son gran parte de una abstracción. Pero este tipo de modelos detallados hacen posible la generación de código, y salvar la distancia semántica entre modelos y aplicación. Yo personalmente creo que haya un lugar para tanto, sólo se

depende de lo que está la construcción y por qué. Al igual que muchas decisiones de arquitectura, no hay una respuesta correcta o incorrecta, como soluciones deben ser evaluados en el contexto de su problema de definición. Ahora hay respuesta de un consultor clásico.

Para un debate en profundidad sobre la documentación de la arquitectura enfoques, el Vistas & Beyond libro de la SEI es la fuente actual de conocimiento:

P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord,
J. Stafford. Arquitecturas de software de documentación: Vistas y más allá. Addison-Wesley, 2ª edición, 2010

Buenas UML 2.0 libros a su alrededor. El que yo encontramos útil es:

SW Ambler. El Primer Objeto 3 rd Edición: Model Driven desarrollo ágil
con UML 2. Cambridge University Press, 2004

Este libro también proporciona una excelente introducción en los métodos ágiles de desarrollo, y cómo el UML se puede usar de manera ligeros y eficaces.

Hay un estándar IEEE, IEEE 1471-2000, para la documentación de arquitectura que es bien vale la pena leer si usted está buscando en la arquitectura de los estándares de documentación que define para su organización. Esto se puede encontrar en:

http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html

Un área emergente de la investigación es la gestión de la arquitectura del conocimiento, con el objetivo de capturar razón de diseño y “conocimiento tribal” que está inevitablemente asociado con cualquier sistema de software de larga vida. He aquí un excelente libro que le dará punteros a las tecnologías y prácticas emergentes en esta área:

Ali Babar, M .; Dingsøyr, T .; Lago, P .; Vliet, H. van (Eds.), Arquitectura de software
Gestión del Conocimiento: Teoría y Práctica, Springer-Verlag 2008 128

Capítulo 9
Diseño Estudio de caso

9.1 Descripción general

En este capítulo, un diseño para el estudio de caso del ICDE se describe en el Cap. 2 es dada. En primer lugar, se le da un poco más de conocimientos técnicos para el proyecto, por lo que los detalles de diseño son más fáciles de digerir. A continuación se presenta la descripción de diseño, y se estructura mediante la plantilla de documentación de la arquitectura introducido en el capítulo anterior. La única sección que no se incluirá en el documento es el primero, el "Proyecto Contexto", como esto se describe básicamente en el Cap. 2. Así, sin más preámbulos, vamos a bucear en la documentación de diseño.

9.2 Problemas técnicos ICDE

El capítulo dos dio un amplio, descripción de nivel de requisitos de la aplicación ICDE v1.0 y los objetivos para la construcción de la próxima versión. Por supuesto, esta descripción es necesaria para entender los requisitos arquitectónicos, pero en realidad, es sólo el punto de partida para las discusiones técnicas que resultan en un diseño real. Las siguientes secciones describen algunas de las cuestiones técnicas, cuyas soluciones se reflejan en la descripción del diseño resultante adelante en este capítulo.

9.2.1 Datos grandes

La base de datos almacena información sobre ICDE las acciones de cada usuario al utilizar su estación de trabajo y aplicaciones. Esto significa que los eventos tales como las aplicaciones de apertura y cierre, a escribir en los datos, mover el ratón, el acceso a Internet, y así sucesivamente todos los datos sobre las causas que se escriben en la base de datos. Aunque la base de datos se purga periódicamente (por ejemplo, todos los días / semana) para archivar datos antiguos y el tamaño de control, algunas tablas de bases de datos pueden crecer rápidamente a un tamaño de varios millones de filas.

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3_9, # Springer-Verlag Berlin Heidelberg 2011

Esto no es un problema para la base de datos a manejar, pero crea un problema de diseño interesante para el API ICDE. Con la aplicación v1.0 ICDE de dos niveles, la herramienta de análisis de datos puede emitir NaI consultas de base de VE (el clásico Seleccionar de VERYBIGTABLE caso) que se puede volver muy grandes conjuntos de datos. Estos son inevitablemente lento para ejecutar y puede derribar la herramienta de análisis si el conjunto de datos devuelto es muy grande.

Mientras inconveniente, de acuerdo con el "! Le preguntó por él, lo tienes" principio, esto no es un problema servidor, y 130
grave para los usuarios de un sistema de usuario único como en v1.0 ICDE. Que sólo hacen daño a sí mismos, y, presumiblemente, tras derribar la aplicación varias veces, aprenderán mejor.

Sin embargo, con el fin de reducir los costes de despliegue y complejidad de la gestión, la transición del sistema ICDE a ser compartido entre varios usuarios es una opción potencialmente atractiva porque:

- Reduce los costos de licencias de base de datos, ya que sólo se necesita uno por cada despliegue, no por usuario
- Reduce la especificación de la PC que los usuarios necesitan para ejecutar la aplicación ICDE, ya que no es necesario para ejecutar la base de datos, sólo el software de cliente ICDE. Simplemente, esto ahorra dinero para una implementación.
- Reduce los costos de soporte y gestión de bases de datos fi ca simplificada, ya que sólo hay una compartida aplicación de servidor ICDE para gestionar y controlar.

Si la base de datos debe ser compartido por varios usuarios, todavía sería posible utilizar una arquitectura de la aplicación de dos niveles o de tres niveles. La opción de dos niveles es probable que proporcionan un mejor rendimiento para implementaciones pequeñas, y sea más fácil de construir que se necesitan menos componentes (básicamente, ningún nivel medio). La opción de tres niveles es probable que escalar mejor como implementaciones de acercarse a un 100-150 a los usuarios, ya que las conexiones de base de datos pueden agruparse y los recursos de procesamiento adicionales desplegados en el nivel medio.

de resultados y la carga simultánea en el servidor ejercida por otros clientes. Una llamada a la API podría hacer caer el utilizando la API que puede fallar de forma impredecible. Las circunstancias de fracaso dependerán del tamaño del conjunto

En cualquier caso, cuando se utiliza una infraestructura compartida, el comportamiento de cada cliente afecta otros. En este caso, las cuestiones a tener en cuenta son:

- el rendimiento de base de datos
- Para la opción de tres niveles, el uso de recursos en el nivel medio

conjuntos de resultados pueden ser devueltos a partir de una solicitud de API, que significa que es posible crear aplicaciones el uso de memoria en el nivel medio es una cuestión importante a considerar, especialmente como clientes ICDE (tanto los usuarios como herramientas de terceros) podrían solicitar conjuntos de resultados con muchos miles de filas. Mientras el servidor de etapa intermedia podría ser con fi gurada con un gran montón de memoria, si varios clientes solicitan resultado considerable establece al mismo tiempo, esto podría fácilmente consumir todos los recursos de memoria servidores, haciendo que el volteo y bajo rendimiento. En algunos casos, esto hará que las peticiones a fallar debido a la falta de memoria y tiempos de espera, y es probable que hará bajar el servidor en casos extremos.

Para herramientas de terceros escritos a la API de ICDE, esto no es en absoluto deseable. Si potencialmente enormes

hacer que todas las aplicaciones conectadas al servidor a fallar también. Esto no es probable que hacer feliz a los equipos de desarrollo o de apoyo como la arquitectura no estaría proporcionando una plataforma de aplicaciones fiables.

9.2.2 Notificación

Hay dos escenarios cuando se necesita cación evento notificada.

1. Una herramienta de terceros puede querer ser informado cuando el usuario realiza una acción específica, por ejemplo, tiene acceso a un nuevo sitio en Internet.
2. Las herramientas de terceros pueden compartir los datos útiles que almacenan en la base de datos del ICDE con otras herramientas. Por lo tanto necesitan un mecanismo para notificar a los interesados sobre los datos que acaba de escribir en el sistema ICDE.

Ambos de estos casos, pero sobre todo la primera, requieren la noti fi cación del evento para ser despachado rápidamente, básicamente, como se produce el evento. Con una arquitectura de dos niveles, instantánea noti fi cación no es tan natural y fácil de lograr. mecanismos de bases de datos tales como desencadenantes pueden ser utilizados, pero estos tienen desventajas potenciales en términos de escalabilidad, y también fl exibilidad. Un trigger es un bloque de instrucciones que se ejecutan cuando se produce una alteración (INSERT, UPDATE, DELETE) a una tabla en la base de datos. mecanismos de activación tienden a explotar características fi específicas de proveedores de base de datos, que inhibirían la portabilidad.

La flexibilidad es la cuestión clave aquí. El equipo de desarrollo del ICDE no puede saber qué eventos o datos de las herramientas de terceros desean compartir a priori (simplemente, no existen las herramientas aún). En consecuencia, algún mecanismo que permite que los propios desarrolladores para crear y publicitar los tipos de eventos “a la carta” es necesaria. Idealmente, esto debe ser apoyado por la plataforma ICDE sin requerir la intervención de un programador o administrador del ICDE.

9.2.3 Abstracción de datos

La estructura de la base de ICDE ha evolucionado considerablemente desde v1.0 a v2.0. Las razones fueron la incorporación de nuevos elementos de datos, y para optimizar la organización interna por razones de rendimiento. Por lo tanto es importante que la organización interna de la base de datos no está expuesto a los desarrolladores de la API. Si lo fuera, cada vez que cambia el esquema, su código se rompa. Esta sería una situación feliz para, precisamente, a nadie.

9.2.4 Plataforma de Distribución y Problemas

proveedores de herramientas de terceros quieren ser capaces de escribir aplicaciones en plataformas no Windows como Linux. Algunas herramientas que desee ejecutar algunos procesos en el mismo

estación de trabajo como el usuario (en Windows), otros querrán ejecutar sus herramientas de forma remota y comunicarse con el usuario a través de mecanismos ubicuos como el correo electrónico y la mensajería instantánea. Una vez más, la clave aquí es que la solución ICDE debe hacer ambas opciones lo menos doloroso posible.

9.2.5 Problemas de API

La API ICDE permite el acceso mediante programación a la memoria de datos del ICDE. El almacén de datos recoge información detallada, con marca de tiempo sobre las clases de eventos de las acciones del usuario, incluyendo:

- Los eventos de teclado
- los eventos de acceso del navegador de Internet
- De aplicación (por ejemplo, procesador de textos, correo electrónico, navegador) eventos abiertos y cerrados
- eventos de cortar y pegar
- Presentar eventos abiertos y cerrados

De ahí la API debe proporcionar un conjunto de interfaces para la consulta de los datos de eventos almacenados en la base de datos. Por ejemplo, si una herramienta de terceros quiere saber las aplicaciones que un usuario ha abierto ya que el último registrado en (su última sesión ICDE), en pseudocódigo la secuencia de llamada a la API podría ser algo como:

```
Sesión SID = getSessionID (ID de usuario, CURRENT_SESSION); ApplicationData []
aplicaciones = getApplicationEvent (SID,
    APP_OPEN_EVENT, NULL); // = NULL todas las aplicaciones
```

los aplicaciones gama ahora se puede caminar a través y, por ejemplo, las páginas web abiertas por el usuario en su navegador durante la sesión se puede acceder [1] y se analizaron usando más llamadas a la API.

La API ICDE también debe permitir que las aplicaciones almacenan datos en el almacén de datos para compartir con otras herramientas o tal vez el usuario. Una API para este fin, en pseudocódigo, se ve así:

```
ok = escritura (myData, myClassifier, publicar, MyTopic);
```

Esto almacena los datos en una tabla de base de datos previamente designado, junto con un ampli fi cación que se puede utilizar para buscar y recuperar los datos. La API también hace que la información sobre este evento que será publicado en el tema mi tema.

En general, para animar a los desarrolladores de terceras partes, la API ICDE tiene que ser útil en el sentido de poder ofrecer a los desarrolladores con las instalaciones que se necesitan para escribir herramientas. Por lo tanto, debe:

---

[1] El almacén de datos ICDE mantiene copias de todas las páginas web visitadas de modo que incluso cambian dinámicamente páginas web (por

- Ser fácil de aprender y fl exible componer secuencias de consultas de la API para recuperar datos útiles.
- Ser fácil de depurar.
- ubicación apoyo de la transparencia. herramientas de terceros no deben tener que ser escrito a una particular, distribuido con fi guración que se basa en ciertos componentes que son en lugares conocidos, fijos.
- Ser resistente posible a ICDE cambios en la plataforma. Esto significa que las aplicaciones no se rompen cuando se producen cambios en la API ICDE o almacén de datos.

9.2.6 Discusión

Tomados en conjunto, los aspectos mencionados anteriormente tejer una red razonablemente complejo de las necesidades y problemas. Los requisitos fi cación caso caciones apuntan fuertemente a una flexible arquitectura editoras suscribirse a unir herramientas de colaboración. La necesidad de apoyar múltiples plataformas y transparentes puntos distribuidos con fi guraciones a una solución Java con los diversos componentes comunicación a través de protocolos como RMI y JMS. Los datos de gran tamaño y los requisitos de extracción de almacenamiento de datos sugieren que se necesita alguna capa para traducir llamadas a la API en las solicitudes SQL necesarias, y luego gestionar el retorno seguro y fiable de la (potencialmente grande) conjunto de resultados al cliente.

La solución del equipo ICDE seleccionado se basa en una arquitectura de tres niveles 3, junto con una infraestructura de publicación-suscripción para el evento de la noti fi cación. Los detalles de esta solución, junto con detalladas justificaciones siguen en la siguiente sección, que documenta la arquitectura usando la plantilla de Cap. 6.

9.3 Requisitos ICDE Arquitectura

En esta sección se describe el conjunto de requisitos que impulsan el diseño de la arquitectura de la aplicación ICDE.

9.3.1 Resumen de los objetivos clave

El primer objetivo de la arquitectura ICDE v2.0 es proporcionar una infraestructura para apoyar una interfaz de programación de terceras partes herramientas de cliente para acceder al almacén de datos ICDE. Esto debe ofrecer:

- La flexibilidad en términos de necesidades guración plataforma y la aplicación de despliegue / con fi para herramientas de terceros.
- Un marco que permita que las herramientas de “tapón” en el medio ambiente y obtener ICDE información inmediata sobre las actividades del usuario ICDE, y proporcionar información a los analistas y potencialmente otras herramientas en el medio ambiente.
- Proporcionar acceso de lectura / escritura cómoda y sencilla para el almacén de datos ICDE.

El segundo objetivo es evolucionar la arquitectura ICDE para que pueda escalar para soportar despliegues de 100-150 usuarios. Esto debe lograrse de una manera que ofrece un bajo coste por el despliegue estación de trabajo.

El enfoque adoptado debe ser consistente con las necesidades de los interesados, así como las limitaciones y requisitos no funcionales que se detallan en las siguientes secciones.

9.3.2 Arquitectura de casos de uso

Dos casos de uso básicos en relación con el uso de la API se han identi fi cado de las discusiones con un pequeño número de potenciales proveedores de herramientas de terceros. Estos son brevemente describe a continuación:

- **ICDE acceso a datos: Consultas de las herramientas de terceros se centran en las actividades de una** usuario ICDE sola. Una secuencia de consulta comienza por obtener información sobre la asignación de trabajo actual del usuario, que es básicamente el proyecto (es decir, “analizar P fi zer Inc nancials fi”) que están trabajando. Consulta de navegación a continuación, profundiza para recuperar datos detallada sobre la actividad del usuario. Los eventos recuperados se buscan en la secuencia de tiempo que se producen, y la lógica de la aplicación busca los elementos especí fi cos de datos (por ejemplo, títulos de ventanas, los valores de teclado, nombres de documentos, URLs) en los registros recuperados. Estos valores se utilizan para inicializar la actividad, ya sea en la tercera herramienta de análisis de las partes, o crear una salida de información que aparece en la pantalla del usuario.

- **Almacenamiento de datos: Herramientas de terceros tienen que ser capaces de almacenar información en el ICDE** almacén de datos, de modo que puedan compartir datos sobre sus actividades. Un mecanismo de noti fi cación se necesita para disfrutar de herramientas para comunicarse acerca de la disponibilidad de nuevos datos. Los datos de cada herramienta es diversa en estructura y contenido. Por lo tanto, debe contener metadatos asociados detectable si ha de ser útil para otras herramientas en el medio ambiente.

9.3.3 Arquitectura requisitos de los interesados

Los requisitos de las perspectivas de los tres principales actores del proyecto se describen en las siguientes secciones.

9.3.3.1 Los productores herramienta de terceros

- **La facilidad de acceso a datos: El almacén de datos ICDE comprende un moderadamente complejo** componente de software. La base de datos relacional tiene aproximadamente 50 mesas, con 134

algunas interrelaciones complejas. En el entorno v1.0 ICDE, esta complejidad hace que las consultas SQL para recuperar datos no triviales para escribir y probar. Además, como los requisitos funcionales evolucionan con cada nueva versión, cambios en el esquema de base de datos son inevitables, y estos podrían romper las consultas existentes. Por estas razones, un mecanismo para hacer más fácil para herramientas de terceros para recuperar datos útiles que se necesita, así como un enfoque para aislar las herramientas de base de datos cambia. herramientas de terceros no deben tener que entender el esquema de la base y escribir consultas complejas.

- soporte de plataformas heterogéneas: Varias de las herramientas de terceros están desarrollando

  tecnologías en plataformas que no sean Windows. El software v1.0 ICDE está estrechamente unida a Windows. Además, la base de datos relacional utilizado sólo está disponible en la plataforma Windows. Por lo tanto, la versión 2.0 del ICDE debe adoptar estrategias para hacer posible que el software no se ejecuta en Windows para acceder a los datos del ICDE y enchufe en el medio ambiente ICDE.

- Instantáneo evento Noti fi cación: Las herramientas de terceros están desarrollando objetivo

  para proporcionar información oportuna a los analistas (usuarios ICDE) sobre sus actividades. Una consecuencia directa de esto es que estas herramientas necesitan acceder a los eventos registrados por el sistema ICDE a medida que ocurren. Por lo tanto, se necesita algún mecanismo para distribuir los eventos generados por el usuario ICDE ya que son capturados en la Almacén de datos.

9.3.3.2 Los programadores ICDE

Desde el punto de vista del programador API ICDE, la API debe:

- Ser fácil e intuitivo de aprender.
- Ser fácil de comprender y modificar el código que utiliza la API.
- Proporcionar un modelo de programación conveniente, concisa para la implementación común

  casos de uso que atraviesan y acceder a los datos del ICDE.
- Proporcionar una API para escribir herramienta especí fi co de datos y metadatos para los datos del ICDE

  almacenar. Esto permitirá que múltiples herramientas para el intercambio de información a través de la plataforma ICDE.

- Proporcionar la capacidad de atravesar los datos del ICDE en navega- inusual o inesperada

  caminos ción. El equipo de diseño no se puede predecir con exactitud cómo se utilizarán los datos en el almacén de datos, por lo que la API debe ser flexible y no inhibe “creativa” utiliza por los desarrolladores de herramientas.

- Proporcionar “bueno” el rendimiento, idealmente devuelvan conjuntos de resultados en un pequeño (1-5)

  número de segundos en un despliegue de hardware típico. Esto permitirá a los desarrolladores de herramientas para crear productos con tiempos de respuesta predecibles.
- Ser flexible en términos de opciones de implementación y distribución de componentes. Esta voluntad

  hacen que sea rentable para establecer instalaciones ICDE para grupos de trabajo pequeños o grandes departamentos.

- Ser accesible a través de una API de Java.

9.3.3.3 Equipo de Desarrollo del ICDE

Desde la perspectiva del equipo de desarrollo de la ICDE, la arquitectura debe:

- Completamente abstracto de la estructura de base de datos y servidor de aplicación mecanismo, aislante herramientas de terceros a partir de los datos de, y cambios en la estructura de almacenamiento de datos ICDE.
- Soporte facilidad de servidor de modi fi cación con un impacto mínimo en el ICDE existente código de cliente que utiliza la API.
- Apoyar el acceso concurrente a partir de hilos o aplicaciones en ejecución múltiple ICDE en diferentes procesos y / o en diferentes máquinas.
- Ser fácil de documentar y transmitir claramente el uso de los programadores de la API.
- Proporcionar un rendimiento escalable. Como los concurrentes aumenta solicitud de carga en una el despliegue del ICDE, debería ser posible escalar el sistema sin cambios en la implementación de la API. La escalabilidad se logra mediante la adición de nuevos recursos de hardware, ya sea a escala vertical u horizontal del despliegue.

mecanismos que proporcionan fiabilidad se prefieren a los que proporcionan un mejor rendimiento. 136

- signi fi cativamente reducir o idealmente eliminar la capacidad de herramientas de terceros para causar fallas en el servidor, reduciendo en consecuencia el esfuerzo de apoyo. Esto significa que la API debe asegurar que los valores de los parámetros en malas llamadas a la API se encuentran atrapados, y que ninguna llamada a la API puede adquirir todos los recursos (memoria, CPU) del servidor ICDE, bloqueando de este modo a cabo otras herramientas.
- No ser excesivamente caro probar. El equipo de prueba debe ser capaz de crear una conjunto de pruebas integral que puede automatizar las pruebas de la API ICDE.

de fallas y soporte de aplicaciones más fácil y más barato. Donde se deben hacer arquitectónicos compensaciones, los

9.3.4 Limitaciones

- El esquema de base de ICDE v1.0 debe ser utilizado.
- El entorno v2.0 ICDE debe ejecutarse en plataformas Windows.

debido a que pasa valores de entrada malas o el bloqueo de recursos o agotamiento. Esto resultará en menos reportes

9.3.5 Requisitos no funcionales

- Actuación: El entorno ICDE v2.0 debe proporcionar sub cinco segundos los tiempos de respuesta a consultas de la API que recuperan hasta 1000 filas de datos, medido en una plataforma de despliegue de hardware “típico”.
- Confiabilidad: La arquitectura ICDE v2.0 debe ser resistente a los fallos inducidos por herramientas de terceros. Esto significa que el cliente llama a la API no puede hacer que el servidor ICDE a fallar

- **Sencillez: A medida que los requisitos de API en concreto son vagos (porque pocos terceros herramientas de existir), simplicidad en el diseño, basado en un flexible [2] arquitectura de fundación, se ve favorecida por** la complejidad. Esto se debe a que los diseños simples son más baratas de construir, más fiable y más fácil de evolucionar para satisfacer las necesidades concretas que van surgiendo. También asegura que, ya que el equipo de **desarrollo del ICDE es poco probable que posean una previsión perfecta, flexible altamente fl [3] y funcionalidad compleja,** pero tal vez innecesaria no se construye hasta casos de uso concretos justifican el esfuerzo. Una amplia gama de funciones admitidas se produce a costa de la complejidad, y la complejidad inhibe la agilidad diseño y capacidad de evolución.

9.3.6 Riesgos

El principal riesgo asociado con el diseño es el siguiente:

Riesgo	Estrategia de mitigación
requisitos concretos no son fácilmente disponibles, fi ya que sólo unos pocos proveedores de herramientas de terceros se cientemente bien informado sobre ICDE para proporcionar insumos útiles	Mantenga el diseño inicial de la API simple y fácil extensible. Cuando más casos de uso de hormigón son identi fi ed, extender la API cuando sea necesario con características para acomodar nuevos requisitos

Solución 9.4 ICDE

Las siguientes secciones describen el diseño de la arquitectura del ICDE.

9.4.1 Patrones de Arquitectura

Los siguientes patrones de arquitectura se utilizan en el diseño:

- **Tres niveles: herramientas de terceros son los clientes, la comunicación con la API de aplica-** tación en el nivel medio, que consulta el almacén de datos ICDE v2.0.
- **Publicación-suscripción: La capa media contiene una capacidad de publicación-suscripción.**
- **capas: Tanto las capas de cliente y emplear nivel medio interno para estructurar el** diseño.

---

[2] Flexible en términos de fácil evolucionar, ampliar y mejorar, y no incluyendo mecanismos que impiden la fácil adopción de una estrategia de arquitectura diferente.

[3] Flexible en cuanto a la gama de sofisticadas características que se ofrecen en la API para recuperar datos GB.

9.4.2 Descripción de la arquitectura

La Introducción a la arquitectura ICDE v2.0 se representa en la Fig. 9.1 . Los clientes utilizan el ICDE ICDE cliente de API componente para hacer llamadas a la Servicios API ICDE componente. Esta es recibido por un servidor de aplicaciones JEE, y se traduce en llamadas a la API JDBC pide al almacén de datos. La existencia Recopilación de datos cliente en v1.0 ICDE se refactorizado en este diseño para eliminar toda la funcionalidad con dependencias de almacenamiento de datos. Todas las operaciones de acceso de almacenamiento de datos son reubicadas en un conjunto de componentes que ofrecen JEE de datos de los servicios de recogida a los clientes alojados.

Evento noti fi cación se consigue utilizando una infraestructura de publicación-suscripción basado en Java Messaging Service (JMS).

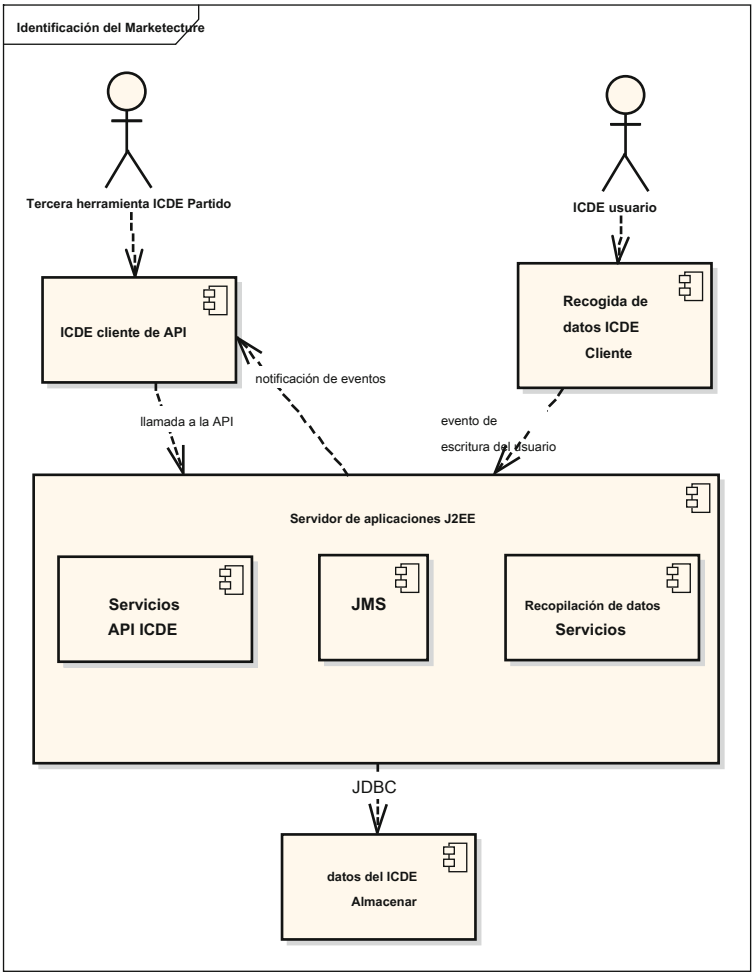


Fig. 9.1 arquitectura API 138 ICDE

El uso de JEE como una infraestructura de aplicaciones, ICDE se puede implementar de manera que un almacén de datos puede soportar:

- Múltiples usuarios que interactúan con los componentes de recogida de datos.
- Múltiples herramientas de terceros que interactúan con los componentes de la API.

9.4.3 Vistas estructurales

Un diagrama de componentes para el diseño API se muestra en la Fig. 9.2 . Esta muestra las interfaces y las dependencias de cada componente, a saber:

- Herramienta de terceros ICDE: Este utiliza el ICDE cliente de API interfaz de componente.

La interfaz API es compatible con los servicios necesarios para la herramienta de terceros para consultar el almacén de datos, escribir nuevos datos en el almacén de datos, y para suscribirse a eventos que son

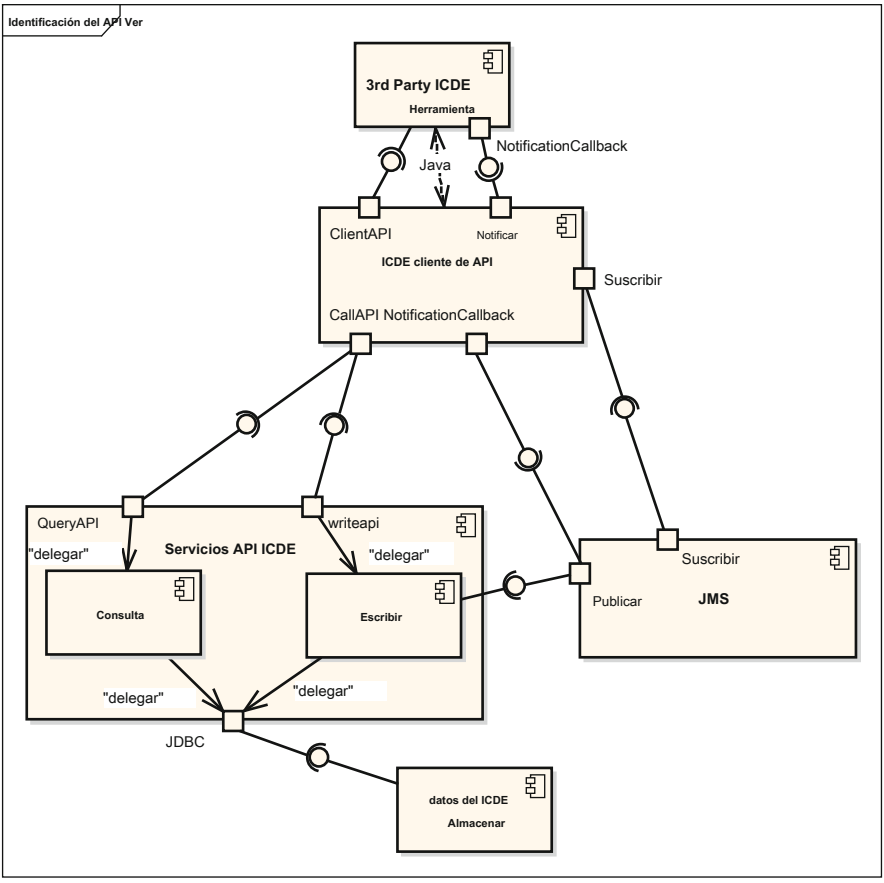


Fig. 9.2 diagrama de componentes para la arquitectura API ICDE

publicado por el JMS. Se debe proporcionar una interfaz de devolución de llamada que el ICDE cliente de API usa para entregar eventos publicados.

- ICDE API de cliente: Esto implementa la parte cliente de la API. Se necesita peticiones de herramientas de terceros, y los traduce a EJB las llamadas a los componentes del servidor API que, o bien leer o escribir datos desde / hasta el almacén de datos. También empaqueta los resultados del EJB y devuelve estos a la herramienta de terceros. Este componente encapsula todo el conocimiento del uso de JEE, el aislamiento de las herramientas de terceros de la complejidad adicional (por ejemplo, localizar, excepciones, grandes conjuntos de datos) de interactuar con un servidor de aplicaciones. Además, cuando una herramienta de terceros solicita una suscripción de eventos, la ICDE cliente de API emite la solicitud de suscripción a las JMS. Por lo tanto, se convierte en el cliente JMS que recibe eventos publicados, y que pasa a éstos en el uso de una devolución de llamada con el apoyo de las herramientas de terceros.

- Servicios API ICDE: El componente de servicios del API comprende sesión sin estado EJB para acceder a la ICDE almacén de datos usando JDBC. los Escribir componente también tiene un valor de parámetro tema de la solicitud del cliente y publica datos sobre el evento sobre el tema asignado un nombre mediante el JMS.
- ICDE almacén de datos: Esta es la base de datos del ICDE v2.0.
- JMS: Este es un servicio de mensajería de Java estándar JEE, y es compatible con una amplia gama de temas utilizados para el evento de la noti fi cación utilizando las interfaces de JMS publicación-suscripción.

Un diagrama de componentes para la funcionalidad de recopilación de datos se representa en la Fig. 9.3 .

Las responsabilidades de los componentes son:

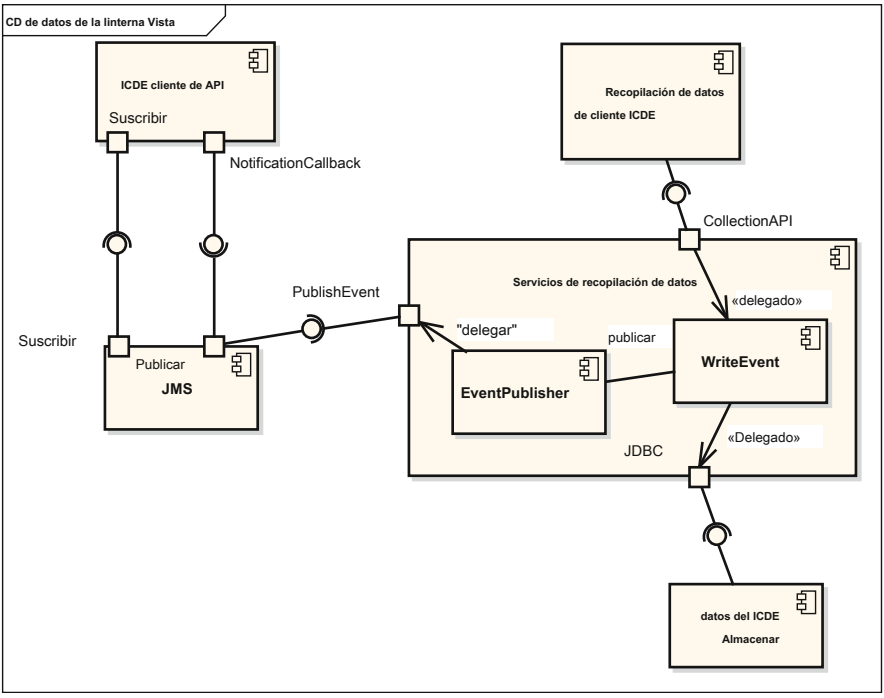


Fig. 9.3 componentes de recogida de datos 140

- **Recogida de datos ICDE cliente:** Esto es parte de la aplicación cliente ICDE ambiente. Que recibe datos de eventos de la aplicación cliente, y llama al método necesario en el CollectionAPI almacenar un evento. Se encapsula todo el conocimiento de la interacción con el servidor de aplicaciones JEE en la aplicación cliente ICDE.

- **Servicios de datos Colección:** Esto comprende los EJB de sesión sin estado que escriben el datos de eventos que se les pasa como parámetros a la ICDE almacén de datos. Algunos tipos de eventos también causan un evento noti fi cación que se pasa a la EventPublisher.
- **EventPublisher:** Esta publica datos de sucesos en las JMS utilizando un conjunto de precon fi gurada temas para los eventos que deben ser publicados (no se publican todos los eventos generados por el usuario, por ejemplo, mover el ratón). Estos eventos se entregan a cualquier ICDE cliente de API componentes que han suscrito el tipo de evento.

Un diagrama de despliegue para la arquitectura ICDE se muestra en la Fig. 9.4 . Muestra cómo los diversos componentes se asignan a los nodos. Sólo un usuario ICDE y una

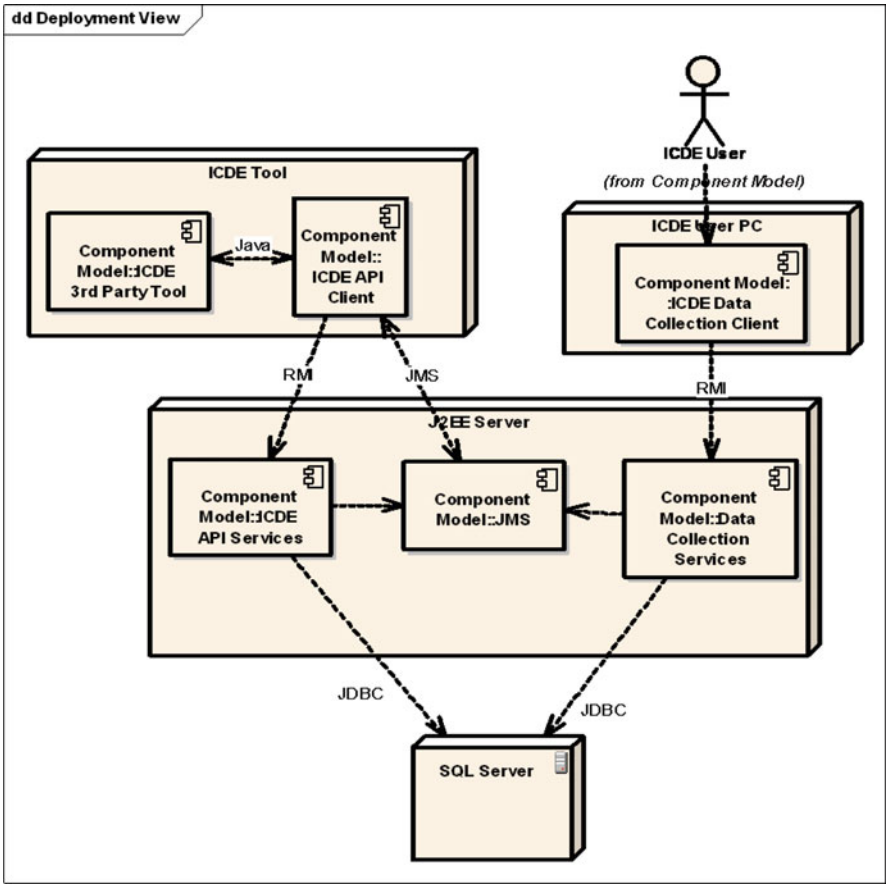


Fig. 9.4 diagrama de despliegue ICDE

herramienta única tercero se muestran, pero el servidor JEE puede soportar múltiples clientes de cualquier tipo. Problemas a la nota son:

- **Aunque se muestran las herramientas de terceros se ejecuta en un nodo diferente a la**

  estación de trabajo de usuario ICDE, esto no es necesariamente el caso. Herramientas o componentes especí fi cos de herramientas, pueden ser desplegados en la estación de trabajo del usuario. Esta es una decisión fi guración tooldependent estafa.

- **Hay uno ICDE cliente de API componente para cada tercer ejemplo herramienta de fiesta.**

  Este componente está construido como un JAR fi l que se incluye en la construcción de herramientas.

9.4.4 Comportamiento Vistas

Un diagrama de secuencia para una llamada de evento API de consulta se muestra en la Fig. 9.5 . La API proporciona una llamada explícita "Inicializar" qué herramientas deben invocar. Esto hace que el ICDE cliente de API establecer referencias a los EJB beans de sesión sin que utilizan el servicio de directorio JEE (JNDI).

Una vez que se inicializa el nivel de API, la herramienta de terceros llama a una de las API de consulta disponibles para recuperar datos de eventos (tal vez una lista de teclas pulsadas durante el uso de la aplicación de procesador de textos en un determinado fi le). Esta petición se pasa a una instancia EJB que implementa la consulta, y se emite la llamada JDBC para obtener los eventos que satisfacen la consulta.

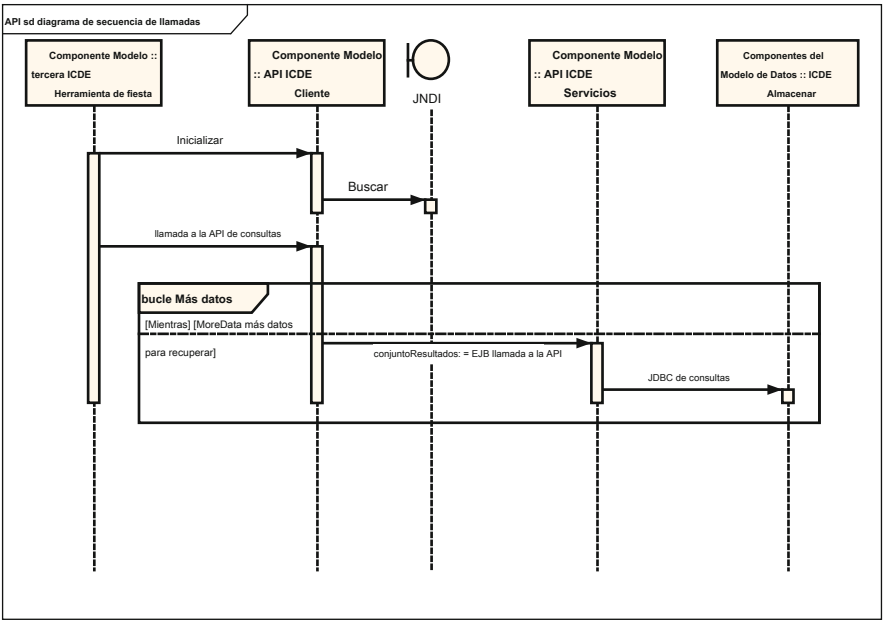


Fig. 9.5 Consulta de la API de llamada diagrama de secuencia 142

Todas las API ICDE que devuelven colecciones de eventos potencialmente pueden recuperar grandes conjuntos de resultados de la base de datos. Esto crea el potencial de agotamiento de los recursos en el servidor JEE, especialmente si múltiples consultas devuelven colecciones de eventos grandes al mismo tiempo.

Para aliviar este rendimiento y la fiabilidad potencial problema, el diseño emplea:

- beans de sesión sin que liberan los recursos utilizados por una consulta al final de cada llamada
- Una variación del iterador página por página [4] para limitar la cantidad de datos cada uno llamar a los recupera bean de sesión

los ICDE cliente de API pasa los valores de los parámetros necesarios para la construcción de la consulta JDBC, junto con una Índice de comienzo y tamaño de página valor. El valor de tamaño de página dice el bean de sesión el número máximo de objetos [5] para volver a partir de una única invocación consulta, y para la llamada consulta inicial, el índice de inicio se establece en NULL.

La llamada JDBC emitido por el bean de sesión explota características de SQL para devolver sólo los primeros tamaño de página filas que satisfacen los criterios de consulta. Por ejemplo, en SQL Server, el operador TOP puede ser utilizado de la siguiente manera:

```
TOP SELECT (PAGESIZE) * DE DONDE KEYBOARDEVENTS (EVENTID> 0 Y EL USUARIO = “Jan” Y
APP_ID = “Firefox”)
```

El conjunto de resultados recuperado por la consulta se devuelve desde el bean de sesión al cliente. Si el conjunto de resultados tiene tamaño de página elementos, el ICDE cliente de API llama al método de consulta EJB nuevo, usando la tecla del último elemento del conjunto de resultados devueltos como el Índice de comienzo parámetro. Esto hace que el bean de sesión para volver a emitir la misma llamada JDBC, excepto con la modi fi cada Índice de comienzo valor utilizado. Esto recupera la siguiente tamaño de página filas (máximo) que responden a la consulta.

los ICDE cliente de API continúa bucle hasta que todas las filas que satisfacen la solicitud se recuperan. A continuación, devuelve la recopilación de eventos agregada a su llamador (la herramienta de terceros). Por lo tanto este esquema oculta la complejidad de recuperar potencialmente grandes conjuntos de resultados desde el programador de la aplicación ICDE.

Un diagrama de secuencia que representa el comportamiento de una escribir llamada API se muestra en la Fig. 9.6 . La llamada a la API de escritura contiene valores de parámetros que permiten la ICDE cliente de API para especificar si un evento debe ser publicado después de una escritura con éxito, y si es así, en qué tema del evento debe ser publicada.

Un diagrama de secuencia para almacenar un evento generado por el usuario ICDE se muestra en la Fig. 9.7 . Un tipo de evento puede requerir múltiples sentencias JDBC INSERT se ejecute para almacenar los datos del evento; por lo tanto los servicios de transacciones de contenedores deben ser utilizados. Después de los datos del evento se almacena con éxito en la base de datos, si es un publicable

---

[4] http://java.sun.com/developer/technicalArticles/JEE/JEEpatterns/

[5] El valor de “Tamaño de página” puede ser ajustado para cada tipo de evento para intentar maximizar el rendimiento del servidor y la red. Un valor típico es de 1.000.

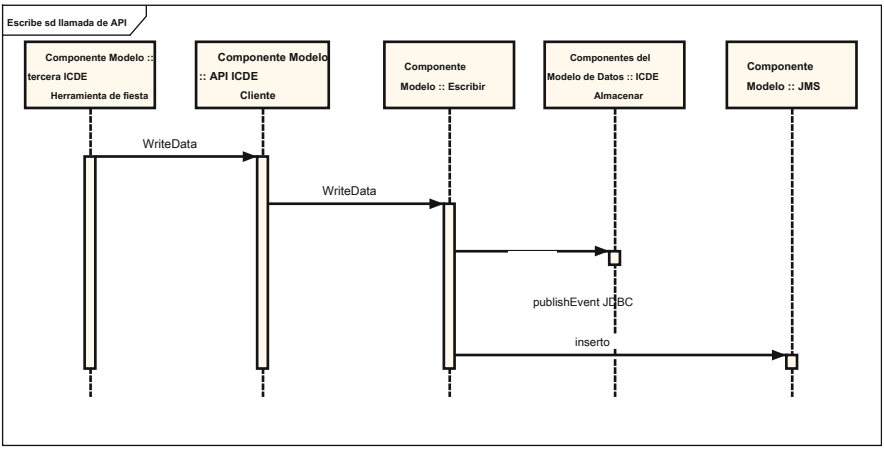
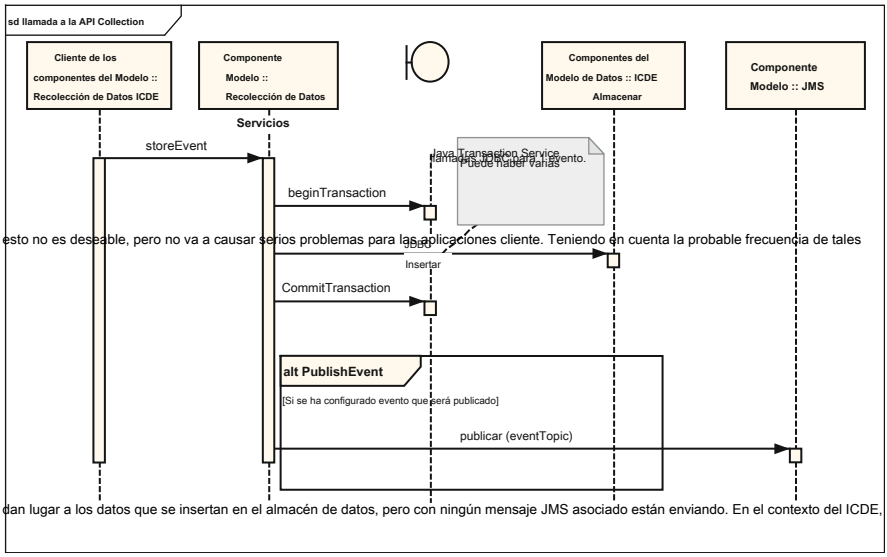


Fig. 9.6 diagrama de secuencia para la API de escritura

fallos que ocurren (es decir, no muy a menudo), esto es una solución de compromiso que es razonable para esta aplicación. 144



esto no es deseable, pero no va a causar serios problemas para las aplicaciones cliente. Teniendo en cuenta la probable frecuencia de tales

dan lugar a los datos que se insertan en el almacén de datos, pero con ningún mensaje JMS asociado están enviando. En el contexto del ICDE,

Fig. 9.7 Diagrama de secuencia para el almacenamiento de los eventos generados por el usuario

tipo de evento, los datos de evento se publica usando el JMS. Los JMS publican operación está fuera del límite de transacción para evitar los gastos generales de una confirmación en dos fases. [6]

[6] Hay una compensación actuación aquí. Como las JMS publican operación está fuera del límite de transacción, no puede haber fracasos que

9.4.5 Cuestiones de implementación

La plataforma Java 2 Enterprise Edition ha sido seleccionado para implementar el sistema ICDE v2.0. Java es una plataforma neutral, satisfaciendo el requisito para la heterogeneidad plataforma. También hay versiones calidad de fuente abierta disponibles para el despliegue de bajo costo, así como las alternativas comerciales de alto rendimiento que pueden ser preferidos por algunos clientes de los sitios más grandes de misión crítica. Además, JEE tiene soporte inherente para sistemas basados en componentes distribuidos, publicación-suscripción caso noti fi cación y acceso a la base de datos.

cuestiones de implementación adicionales a considerar son:

- **threading: los ICDE cliente de API componente debe ser seguro para subprocesos. Esta voluntad**
  permitir a los desarrolladores de herramientas para desovar de forma segura múltiples hebras de la aplicación y emitir llamadas a la API concurrentes.
- **Seguridad: herramientas ICDE se autentican con un nombre de usuario y contraseña. La API SUP-**
  puertos de una iniciar sesión función, que valida la combinación de usuario / contraseña contra las credenciales en el almacén de datos ICDE, y permite el acceso a un conjunto fi cado de datos de usuario ICDE. Este es el mismo mecanismo utilizado en v1.0.
- **EJB: los Servicios de recopilación de datos beans de sesión directos de emisión de llamadas JDBC**
  acceder a la base de datos. Esto es debido a que el JDBC ya las llamadas existe en la v1.0 ICDE de dos niveles, y por lo tanto el uso de estos directamente en el EJB hace que el ejercicio de refactorización menos costoso.

9.5 Análisis de Arquitectura

Las siguientes secciones proporcionan un análisis de la arquitectura ICDE en términos de escenarios y riesgos.

Análisis 9.5.1 Escenario

Los siguientes escenarios se consideran:

- **Modificar la organización del ICDE de datos de la tienda: Los cambios en la organización de base de datos**
  requerirá cambios de código en los componentes de servidor EJB. Los cambios estructurales que no añaden nuevos atributos de datos se encuentran totalmente dentro de estos componentes y no se propagan a la API ICDE. Modi fi caciones que añaden nuevos elementos de datos se requieren cambios en la interfaz de componentes del lado del servidor, y esto será reflejado en el API. versiones interfaz y desaprobación método se puede utilizar para controlar la forma en que estos cambios afectan a la interfaz de componentes de cliente.
- **Mueva la arquitectura ICDE a otro proveedor JEE: Mientras el ICDE**
  aplicación está codificado para los estándares JEE, y no utiliza ninguna extensión de vendedores

clases, experiencia en la industria muestra que las aplicaciones JEE son portátiles de un servidor de aplicaciones a otro con pequeñas cantidades de esfuerzo (por ejemplo, menos de una semana). di fi cultades se encuentran por lo general en las áreas de aplicación de servidor especí fi cos de opciones del descriptor de implementación con fi guración del producto y.

ı **Escalar un despliegue de 150 usuarios: Esto requerirá una cuidadosa planificación de la capacidad** [7]

basado en la especi fi cación de los equipos y las redes disponibles. El nivel de servidor JEE se puede replicar y agrupado con facilidad debido a la utilización de beans de sesión sin estado. Es probable que se necesite una más potente servidor de base de datos para 150 usuarios. También debe ser posible para dividir el almacén de datos ICDE a través de dos bases de datos físicos.

9.5.2 Riesgos

Los siguientes riesgos deberán dirigirse medida que el proyecto avanza ICDE.

Riesgo	Estrategia de mitigación
La planificación de capacidad para una sitio grande será complejo y costoso	Vamos a llevar a cabo las pruebas de rendimiento y carga una vez que el básico entorno de servidor de aplicaciones está en su lugar. Esto proporcionará cifras concretas rendimiento fi que pueden guiar la planificación de capacidad para los sitios ICDE
La API no cumplirá las nuevas necesidades proveedor herramienta de terceros	La API se dará a conocer tan pronto como una versión inicial se haya completado para proveedores de herramientas para ganar experiencia. Esto nos permitirá obtener retroalimentación temprana y adaptar / extendemos el diseño si / cuando sea necesario

9.6 Resumen

ICDE, basado en el número de usuarios simultáneos, las velocidades de red y hardware disponible. 146

En este capítulo se ha descrito y documentado algunas de las decisiones de diseño tomadas en la aplicación ICDE. El objetivo ha sido el de transmitir el pensamiento y análisis que es necesario diseñar una arquitectura de este tipo, y demostrar el nivel de documentación de diseño que debe bastar en muchos proyectos.

Tenga en cuenta que algunos de los detalles más fina del diseño se pasan por alto necesariamente más debido a las limitaciones de espacio de este foro. Pero el ejemplo ICDE es representativo de una aplicación de mediana complejidad, y por lo tanto ofrece un excelente ejemplo de la obra de un arquitecto de software.

---

[7] La planificación de capacidad implica fi gurar cuánto hardware y el software que se necesita para apoyar una instalación especí fi co

Capítulo 10
Middleware Estudio de caso: Medici

Adam Wynne

10.1 Antecedentes Medici

En muchos ámbitos de aplicación en la ciencia y la ingeniería, los datos producidos por los sensores, instrumentos y redes es naturalmente procesados por aplicaciones de software estructurado como una tubería. [1] Tuberías comprenden una secuencia de componentes de software que procesan progresivamente unidades discretas de datos para producir un resultado deseado. Por ejemplo, en un rastreador web que se extrae la semántica del texto en los sitios Web, la primera etapa en la tubería podría ser la eliminación de todas las etiquetas HTML para dejar sólo el texto sin formato del documento. El segundo paso puede analizar sintácticamente el texto prima para descomponerlo en sus partes constituyentes gramaticales, tales como nombres, verbos, y así sucesivamente. Los pasos posteriores pueden buscar nombres de personas o lugares, eventos interesantes o tiempos para que los documentos se pueden secuenciar en una línea de tiempo. El uso de tuberías de Unix, esto podría ser algo como esto: [2]

```
curl47 http://sites.google.com/site/iangortonhome/ |                    \
    ToText | \ Analizar | \ Gente \ \
    -O lugares out.txt
```

Cada uno de estos pasos se puede escribir como un programa especializado que trabaja de manera aislada con otros pasos en la tubería.

En muchas aplicaciones, software simples tuberías lineales son su fi ciente. Sin embargo, las aplicaciones más complejas requieren topologías que contienen horquillas y se une a, la creación de tuberías que comprenden ramas donde la ejecución en paralelo es deseable. También es cada vez más común para las tuberías de proceso muy grandes flujos de datos o archivos de gran volumen que imponen limitaciones de rendimiento de extremo a extremo. Además, los procesos en una tubería pueden tener requisitos especí fi c de ejecución y por lo tanto deben ser distribuidos de servicios a través de una infraestructura de gestión informático heterogéneo y datos.

Desde una perspectiva de la ingeniería de software, estas tuberías más complejos se hacen difícil de implementar. Mientras que las tuberías lineales simples pueden ser construidos usando un mínimo

1 http://en.wikipedia.org/wiki/Pipeline_%28software%29

2 http://en.wikipedia.org/wiki/CURL

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3_10, # Springer-Verlag Berlin Heidelberg 2011

infraestructura, tales como lenguajes de script, topologías complejas y, procesamiento de datos de gran volumen grande requiere abstracciones adecuadas, infraestructuras de gestión de tiempo y herramientas de desarrollo para la construcción de tuberías con las cualidades deseadas de servicio y flexibilidad de evolucionar para manejar los nuevos requisitos.

Lo anterior resume las razones que hemos creado theMeDICi Integración Marco (MIF) que está diseñado para la creación de alto rendimiento, escalable y modi fi tuberías de software capaz. MIF explota una fricción baja, robusta plataforma de middleware de código abierto y se extiende con interfaces de programación de componentes y basadas en los servicios que hacen que la implementación de tuberías complejas sencilla. El tiempo de ejecución MIF maneja automáticamente colas entre elementos de conducción con el fin de manejar la petición ráfagas y automáticamente ejecuta varias instancias de elementos de conducción para aumentar el rendimiento de tuberías. elementos de conducción distribuidos están soportadas mediante una serie de protocolos de comunicaciones gurable con fi, y las interfaces de MIF proporcionan mecanismos fi cientes para mover datos directamente entre dos elementos de conducción distribuidos.

El resto de este capítulo se describen las características del MIF, se muestran ejemplos de las tuberías que hemos construido, y da instrucciones sobre cómo descargar la tecnología.

10.2 Medici Hello World

que se pasan entre cada paso. 148
Comenzaremos con una descripción del ejemplo saludos universales clásico con MIF. los Hola Mundo ejemplo de esta sección demuestra una sencilla tubería de MIF en dos etapas. Para entender este ejemplo, a continuación, cuatro sencillos de fi niciones de los conceptos MIF que se utilizan en el código.

1. **Tubería. aplicación AMIF consiste en una serie de módulos de procesamiento contenida en una canalización de procesamiento. los MifPipeline objeto se utiliza para controlar el estado de la tubería, así como** para agregar y registrar todos los objetos contenidos en la tubería.
2. **código de implementación. Este es el código (por ejemplo, una clase Java o programa C) que** realiza el procesamiento real a un paso dado en la tubería.
3. **Módulo. Amodule envuelve la funcionalidad de algún código de aplicación con una** interfaz que encapsula la aplicación. Un módulo de instancia puede ser incluido como un paso en una tubería MIF.
4. **Punto final. Se conecta un módulo con otros módulos en la tubería mediante el uso de una de** varios protocolos de comunicación estándar, tales como JMS, SOAP, y muchos otros.

En el MIF Hola Mundo ejemplo, cuando se inicia la tubería, se pide al usuario que introduzca un nombre. El nombre se envía a la helloNameModule que llama al helloNameProcessor aplicación para añadir "Hola" al frente del nombre y pasa toda la cadena a la helloHalModule. Esto exige la helloHalProcessor añadir "¿qué estás haciendo" al final de la cadena, que se devuelve al usuario a través de la consola.

En la Fig. 10.1 , Los rectángulos azules son los módulos y los cuadrados negros son puntos finales.
El camino de datos se representa mediante líneas azules de puntos y rayas, que se anota con las cadenas de datos

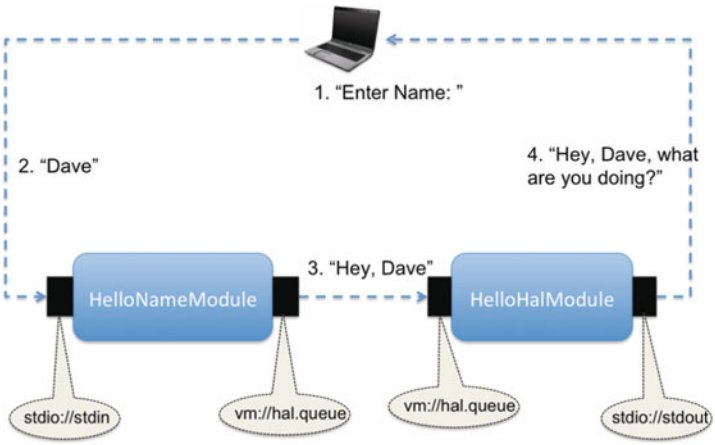


Fig. 10.1 MIF “Hola Mundo” ejemplo

Los siguientes fragmentos de código demuestran las partes pertinentes del Código. El código se presenta de una manera “de arriba abajo”, donde se presenta el código de nivel superior primero y avanzar progresivamente hacia abajo para los detalles de implementación.

En primer lugar, tenemos que crear un objeto MifPipeline que se necesita para iniciar y detener la canalización de procesamiento. Este objeto también se utiliza para crear y registrar todos los objetos que se ejecutan dentro de la tubería.

```
MifPipeline tubería = new MifPipeline ();
```

A continuación, añadimos el HelloNameModule la cual antepone “Hola” al nombre introducido y envía la nueva cadena a la siguiente etapa en la tubería. El primer argumento es una cadena que representa el nombre completo de la clase de la clase de implementación. Los argumentos segundo y tercero son los extremos de entrada y salida que permiten a nuestra
HelloNameModule para recibir y enviar datos.

```
pipeline.addMifModule (HelloNameProcessor.class.getName () "? stdio: // stdin promptMessage =
Introducir nombre:", "vm: //hal.queue");
```

Por último, se añade el HelloHalModule (y sus puntos finales) a la tubería. Esto exige la HelloHalProcessor añadir otro fragmento de frase en el extremo de la cuerda y lo imprime a la consola del usuario.

```
pipeline.addMifModule (HelloHalProcessor.class.getName () "vm: //hal.queue", "stdio: // stdout");
```

Los módulos en el ejemplo anterior se comunican usando criterios de valoración, que se pasan a un módulo como argumentos. Los puntos finales son una abstracción que permiten los protocolos de comunicación entre los módulos para ser fl ed fi exible específico. Esto encapsula la lógica de implementación del módulo de tener que preocuparse de los protocolos de comunicaciones utilizados para el intercambio de mensajes con otra

módulos. Esto significa por ejemplo que un módulo de fi con gurado con un punto final JMS se puede cambiar para enviar datos a través de UDP sin ningún cambio en el código de implementación del módulo.

En este ejemplo, la HelloNameModule ' s punto final de entrada, stdio: // stdin, lee la entrada del usuario desde la consola. Es decir, stdio es un protocolo especial que lee desde la consola y envía los datos de la consola al módulo. El punto final de salida, VM: // hal. cola, es un punto final proporcionado-MIF implementada en la JVM, proporcionando una e fi ciente, en cola mecanismo de comunicación entre los módulos. Tenga en cuenta que los módulos sean capaces de comunicarse, el punto final de salida del remitente debe ser el mismo que el punto final de entrada del receptor.

Después de que los módulos de tuberías se con fi gura en la tubería, se comienza la aplicación llamando al método MifPipeline.start (). Esto inicia el MIF con el gasoducto con fi guración e inicia los módulos para escuchar los datos.

```
pipeline.start ();
```

código de implementación módulo es proporcionado por el diseñador de tuberías para llevar a cabo algún tipo de procesamiento de los datos fl que fluye a través de una tubería. Este código se envuelve por un módulo para formar la unidad de código más pequeño que puede ser colocado en una tubería. El código de aplicación puede ser escrita en Java o cualquier otro idioma. Al utilizar Java, el código se integra directamente en la tubería (se trata como un ejecutable externo para otros idiomas, como se explica más adelante). Por ahora, nos concentraremos en la creación de clases de implementación de Java.

clases de implementación necesitan para implementar el MifProcessor interfaz, que proporciona una escucha método con la siguiente firma:

```
Serializable Público (entrada Serializable);
```

Este método se llama cuando llega un mensaje para el módulo que se envuelve la clase de implementación. El argumento de entrada son los datos recibidos desde el módulo anterior en la tubería, y el valor de retorno es el mensaje que se envía al siguiente módulo de la tubería.

Ahora vamos a echar un vistazo a la implementación de uno de los módulos de este ejemplo. los HelloNameProcesso implementa la funcionalidad para la HelloNameModule. Cuando este módulo recibe datos, su escucha() método se llama, y los datos de entrada se pasa como argumento método. El método procesa los datos de alguna manera y devuelve el resultado que se pasa a la siguiente módulo en la tubería. En este caso, el escucha método simplemente agrega la cadena “Hola” al frente de la cadena recibida y devuelve la nueva cadena.

```
HelloNameProcessor clase pública implementa MifProcessor {
    Serializable Público (nombre Serializable) {
        String cadena = "Hola" + nombre;
        System.out.println ( "HelloNameProcessor:" + str); str regresar; }}
```

10.3 Módulos de Aplicación

Un MIF Módulo representa la unidad básica de trabajo en una tubería MIF. Cada módulo tiene una clase de implementación, conocido como un procesador de MIF. La clase de procesador está especí fi ed como el primer argumento de la addMifModule método de fábrica cuando se crea un módulo:

```
MifModule mi_módulo =
pipeline.addMifModule (MyMifProcessor.class, "vm: //in.endpoint"
,    "Vm: //out.endpoint");
```

Cada clase debe implementar un procesador Procesador interfaz y un asociado escucha método que acepta un objeto que representa la carga útil de datos recibida en el punto final de entrada del módulo. los escucha método también devuelve un objeto que representa la carga útil que se envía a través de punto final de salida del módulo.

```
MyMifProcessor clase pública implementa MifObjectProcessor {
        Objeto Público (entrada Object) {// realizar algún tipo de
        procesamiento en la entrada
            salida de retorno; }}
```

Hay unos pocos tipos diferentes de interfaces de procesador en función de si desea imponer el uso de objetos serializados y si el procesador necesita para manejar de forma explícita propiedades de los mensajes que se envían como un encabezado en todos los mensajes que pasan a través de una tubería MIF. Estas interfaces se encuentran en el gov.pnnl paquete. mif.user y se explica a continuación:

10.3.1 MifProcessor

los MifProcessor interfaz se utiliza para implementar un módulo si desea exigir que los tipos enviados y recibidos por el módulo son Serializable.

```
MifProcessor interfaz pública {
    Serializable Público (entrada Serializable); }
```

10.3.2 MifObjectProcessor

Este es el tipo más general de la interfaz, lo que permite cualquier tipo de objeto a ser recibida por el método de escuchar. A menudo es deseable para comprobar el tipo de objeto recibido (con la en vez de operador) para asegurarse de que coincida con el tipo derivado correcta.

```
MifObjectProcessor interfaz pública {
        Objeto Público (entrada de objetos); }
```

10.3.3 MifMessageProcessor

Este tipo de procesador se utiliza cuando es necesario tener acceso a las propiedades de mensaje asociados con un mensaje dado. Normalmente, un procesador recibe sólo la carga útil del mensaje, pero esta interfaz permite que el módulo para recibir tanto.

```
MifMessageProcessor interfaz pública {
        Objeto Público (entrada de objetos,
                                                MessageProperties messageProperties);
}
```

Propiedades del módulo 10.3.4

Puesto que la clase de procesador para un determinado MifModule se crea una instancia por el contenedor MIF subyacente, no es posible crear de forma manual y con fi gura una clase de procesador antes de añadir en una tubería. Por lo tanto, las propiedades del módulo son proporcionados por la API para que el usuario pueda pasar propiedades de procesador a MIF. MIF entonces poblar las propiedades en el procesador cuando se crea una instancia. Las propiedades se establecen en el procesador de manera similar a las propiedades JavaBean, lo que significa que la clase tiene un constructor sin argumentos y métodos setter y getter estándar.

Por ejemplo, la siguiente es una MifProcessor con los emisores de estilo JavaBean:

```
public class de Apple implementa MifObjectProcessor {

        String color privada; tipo String
        privado;

        Objeto Público (entrada Object) {
              // hacer cosas salida de
           retorno; }

        / * La manzana de color a * / setColor pública (String
        color) {
                this.color = color; }

        / * El tipo / variedad de manzana * / setType pública
        (tipo String) {
                this.type = Tipo; }}
```

Las propiedades de este módulo a continuación, se pueden ajustar con el código siguiente:

```
MifModule appleModule = pipeline.addMifModule (Apple.class "vm: //in.endpoint", "vm: //out.endpoint");
appleModule.setProperty ( "color", "rojo"); appleModule.setProperty ( "tipo", "Honeycrisp");
```

10.4 Criterios de valoración y Transportes

En MIF, la comunicación entre los módulos está habilitado de transportes, que son responsables de la abstracción de la red de comunicaciones y la transmisión de mensajes a través de una tubería. Cada protocolo de comunicación que es soportado por MIF (por ejemplo, JMS o HTTP) se implementa mediante un transporte separado. La mayor parte de la complejidad de un transporte está oculto al usuario por el uso de criterios de valoración con fi gurable, que permiten que un módulo sea ajeno a los protocolos de comunicación se está utilizando. Sin embargo, a veces es necesario o deseable para con fi gura los atributos de un transporte. En tales circunstancias, hay APIs que permiten al programador para crear de forma explícita y con fi gura un conector.

Cada transporte tiene su propio tipo de punto final, que se utiliza para conectar los módulos entre sí de manera que puedan intercambiar mensajes. Cada módulo tiene un conjunto de criterios de valoración entrantes y salientes que se pueden ajustar mediante la API de MIF. Para conectar un módulo a otro, el punto final de salida de un módulo en la tubería debe coincidir con el punto final de entrada de otro.

Las estaciones se con fi gura como cadenas que representan un URI. Es posible establecer las propiedades en un punto final para con un comportamiento especial fi gura o anular las propiedades predeterminadas de un conector. Por ejemplo, es posible con fi gurewhether un punto final es síncrona o asíncrona estableciendo la propiedad “sincrónica”, como explicaremos en breve.

Un punto final URI tiene el formato:

esquema: // host: puerto / ruta / a / servicio propiedad1 = valor1 y valor2 = propiedad2

Donde “régimen” es el tipo particular de transporte utilizado (http, JMS, vm, etc.); “Host: puerto” es un nombre de host y el puerto (que puede o no puede estar presente debido a la naturaleza de un transporte dado), y “camino” es el camino que distingue el punto final de los demás. Propiedades se definen después de un “?” Al final de un camino y se delinean por pares de valores clave.

los puntos finales de cada transporte son ya sea sincrónica o asincrónica por defecto. Este defecto puede cambiarse estableciendo la propiedad “sincrónica” en un punto final. Por ejemplo, extremos HTTP son síncronos por defecto. El siguiente de fi ne un extremo de entrada HTTP que crea un servicio asíncrono, escuchando en la dirección localhost: 9090 / HelloService:

http: // localhost: 9090 / HelloService síncrono = false

10.4.1 Conectores

Conectores se utilizan para con fi gura los atributos de un transporte particular para la tubería o componente de corriente. Por ejemplo, el conector JMS permite al usuario con fi gura la ubicación del servidor JMS. La mayoría de las veces, el usuario no necesita

de forma explícita con fi gurar un transporte ya se creará automáticamente un conector predeterminado para cada tipo de punto final que se produce en la tubería. Sin embargo, es necesario crear y conectores con fi gurar cuando:

1. Es necesario optimizar el rendimiento de un transporte. Esto es común, por ejemplo, al usar los puntos finales TCP.
2. Sin conector predeterminado puede ser creado. Por ejemplo, el uso de JMS requiere un conector explícita, ya que es necesario especificar la ubicación del servidor JMS.

Si sólo hay un conector para un protocolo dado en una tubería MIF, todos los puntos finales asociados con el transporte que se utilizan este conector. Sin embargo, pueden existir múltiples conectores para el mismo transporte en una sola tubería. En este caso, es necesario especificar el nombre del conector como una propiedad en el punto final de manera que MIF sabe qué conector a utilizar para ese parámetro en particular.

en sincrónico ¼ cierto, los mensajes son 154

Por ejemplo, el siguiente fragmento de código muestra un JMS conector definido con el nombre JMS-localhost. A continuación, un módulo es con fi gurado con un punto final de entrada, que especi fi ca que el conector, utilizando el "conector" propiedad de punto final. Esto permite que los puntos finales de una tubería MIF para conectarse a varios servidores JMS:

```
MifConnector conn = pipeline.addMifJmsConnector ( "tcp: // localhost: 61616",
  JmsProvider.ACTIVEMQ); conn.setName ( "JMS-localhost");

MifModule fullNameModule = pipeline.addMifModule (NameArrayProcessor.class,

    "? JMS: // tema: NameTopic conector JMS-= localhost", "stdio: // stdout");
```

Las propiedades se pueden configurar en un conector llamando al setProperty método en un MifConnector, p.ej:

```
MifConnector stdioConn =
    pipeline.addMifStdioConnector ("stdio: // stdout");stdioConn.setProperty (
"messageDelayTime", 1000);
```

que se puede establecer en una VM punto final es si es síncrona o asíncrona. Si esta propiedad se establece

10.4.2 Transportes soportados

La API FOMIN apoya una serie de transportes. A continuación se muestra una descripción de éstos, junto con una descripción de las propiedades útiles que se pueden fijar en los puntos finales y / o conectores de este tipo.

Todos los puntos finales apoyan la conector ¼

ConnectorName propiedad como se ha descrito anteriormente.

10.4.2.1 VM

los VM punto final se utiliza para la comunicación entre componentes dentro de la JVM. La propiedad clave

pasado de forma sincrónica de un módulo a otro de manera que un receptor lento podría ralentizar el remitente. Por otro lado, si esta propiedad se establece en false, el emisor envía tan rápido como se pueda, sin esperar a que el receptor para completar cada solicitud. Internamente, el contenedor MIF gestiona una cola de mensajes asociados con el conector.

Por ejemplo, aquí es un ejemplo de una asíncrona VM punto final

"Vm: // myQueue síncrono = false"

10.4.2.2 STDIO

los STDIO el transporte se utiliza para leer estándar en y escribir a la salida estándar.
Es útil para probar y depurar. Un ejemplo de STDIO criterios de valoración es la siguiente:

```
MifModule appleModule = pipeline.addMifModule (
    TextProc.class, "vm: //in.endpoint",
    "vm: //out.endpoint");
```

Servicio de mensajería de Java 10.4.2.3

El transporte JMS conecta los puntos finales del FOMIN para destinos JMS (temas y colas). Es posible utilizar cualquier proveedor de servidor JMS con MIF. Por conveniencia, la instalación MIF incluye ActiveMQ como el proveedor JMS preferido (y el proveedor de MIF se prueba con).

Por defecto, JMS puntos finales especificar las colas. En ActiveMQ, colas se deben crear administrativamente. Por ejemplo, los siguientes URI especí fi ca que un extremo se conecta a la cola llamada “aQueue”

JMS: // aQueue

Para especificar un tema, anteponer el nombre del destino con la cadena “tema:”. Por ejemplo, el siguiente URI especí fi ca un punto final conectado al tema denominado “bTopic”

JMS: // tema: bTopic

Para crear un conector ActiveMQ JMS, es necesario especificar el URI del servidor, así como el proveedor de JMS. Especificación del proveedor de esta manera permite providerspeci propiedades fi cas que se establecen automáticamente en el conector:

```
pipeline.addMifJmsConnector ( "tcp: //
localhost: 61616", JmsProvider.ACTIVEMQ);
```

10.4.2.4 HTTP

El transporte HTTP permite a los puntos finales entrantes para actuar como servidores web y los puntos finales salientes para que actúen como clientes HTTP. extremos HTTP son síncronos por defecto. El siguiente es un punto final HTTP entrante que crea un servicio asíncrono, escuchando en la dirección localhost: 9090 / HelloService.

http: // localhost: 9090 / HelloService síncrono = false

10.4.2.5 HTTPS

El transporte HTTPS permite la creación de servicios seguros a través de HTTP. Para crear un servicio de este tipo, el conector debe ser con fi gurada para apuntar a la almacén de claves donde se encuentra certi fi cado del servicio. Esto requiere que se establezcan las propiedades siguientes:

propiedades del conector	Descripción
keyStore	La ubicación del almacén de claves de Java fi l
keyStorePassword	La contraseña del almacén de claves
keyPassword	Contraseña de la clave privada

A modo de ejemplo, la siguiente tubería MIF crea un conector HTTPS, que es con fi gurado para usar un certificado autofirmado que se pueden crear con el Java herramienta de claves. [3]

MifPipeline tubería = new MifPipeline ();

MifConnector httpsConn = pipeline.addMifConnector (EndpointProtocol.HTTPS);

httpsConn.setProperty ("almacén de claves", "/dev/ssl/keys/pnl.jks"); httpsConn.setProperty ("keyStorePassword", "storepass"); httpsConn.setProperty ("keyPassword", "keypass");

pipeline.addBridgeModule ("https: // hostname: 9090 / secureService", "stdio: // out"); pipeline.start ();

TCP 10.4.2.6

Este transporte permite la creación de servidores de escucha en sockets TCP primas (para los puntos finales entrantes) y clientes que envíen a las tomas de salida (puntos finales). A modo de ejemplo, el siguiente punto final URI creará un servidor escucha en el host y el zócalo determinado si se utiliza como criterio de valoración de entrada.

tcp: // localhost: 7676

3 http://download.oracle.com/javase/1.4.2/docs/tooldocs/windows/keytool.html

El siguiente es un ejemplo de un conector TCP definición. Dado que el protocolo TCP no tiene el concepto de un mensaje, es necesario para definir un algoritmo de “protocolo” y establecer esto como una propiedad en el conector. Por lo tanto, siempre se debe crear explícitamente un conector TCP y elegir un protocolo TCP procesamiento apropiado.

```
MifConnector conn = pipeline.addMifConnector (EndpointProtocol.TCP);

conn.setProperty ( "tcpProtocol", nuevo EOFProtocol ());
```

los EOFProtocol clase instanciado en este ejemplo es una clase proporcionado-mula que define el límite mensaje a ser cuando se recibe EOF (final de la fi le) en el zócalo. Es decir, el mensaje termina cuando el cliente se desconecta del servidor mediante el envío de un carácter EOF. Todos los protocolos disponibles se pueden encontrar en la documentación de transporte TCP de mula. [4] Por ejemplo, se proporciona un protocolo que asume cada mensaje está precedido por el número de bytes que se enviará, de manera que un mensaje completo se puede construir. También es posible especificar una costumbre, clase de protocolo fi co-aplicación específica.

Otras propiedades pueden establecerse en los conectores TCP para optimizar el rendimiento. Estos incluyen el tamaño del envío y recepción tampones, y para los puntos finales de salida, ya sea una toma de corriente debe permanecer abierta después de cada envío con el fin de mejorar el rendimiento.

Desde MIF es una especialización y simplificación de Mule, los transportes utilizados en MIF son un subconjunto de aquellos en Mule. Por lo tanto, la documentación aquí es una forma muy condensada de la documentación de transporte mula. Puede ser útil hacer referencia a la documentación completa de la mula para aprender el conjunto completo de funciones que ofrece la mula. [5]

La documentación se centra en las propiedades requeridas por las aplicaciones de MIF.

10,5 Medici Ejemplo

La aplicación de ejemplo que describimos aquí analiza los mensajes de chat de Internet para extraer diferentes tipos de contenido de los mensajes. La estructura global de la aplicación se muestra en la Fig. 10.2 .

Básicamente, cuando se inicia la aplicación, el programa principal inicializa una tubería MIF y luego simula un proceso externo que está tirando mensajes de chat fuera de una red y de insertarlos en la tubería a través de JMS. A partir de ahí, el Ingerir módulo toma una línea de datos de chat y lo analiza en un objeto (MapWrapper) que se utiliza en todo el resto de la tubería. Desde el Ingerir módulo, copias separadas de los datos (en forma de una MapWrapper) se encaminan a tres módulos de procesamiento concurrentes (la lógica real de la transformación se delega en el

[4] http://www.mulesoft.org/documentation/display/MULE2USER/TCP+Transport

[5] http://www.mulesoft.org/documentation/login.action?os_destination ¼% 2Fdisplay% 2FMULE2USER% 2FTCP% 2BTransport

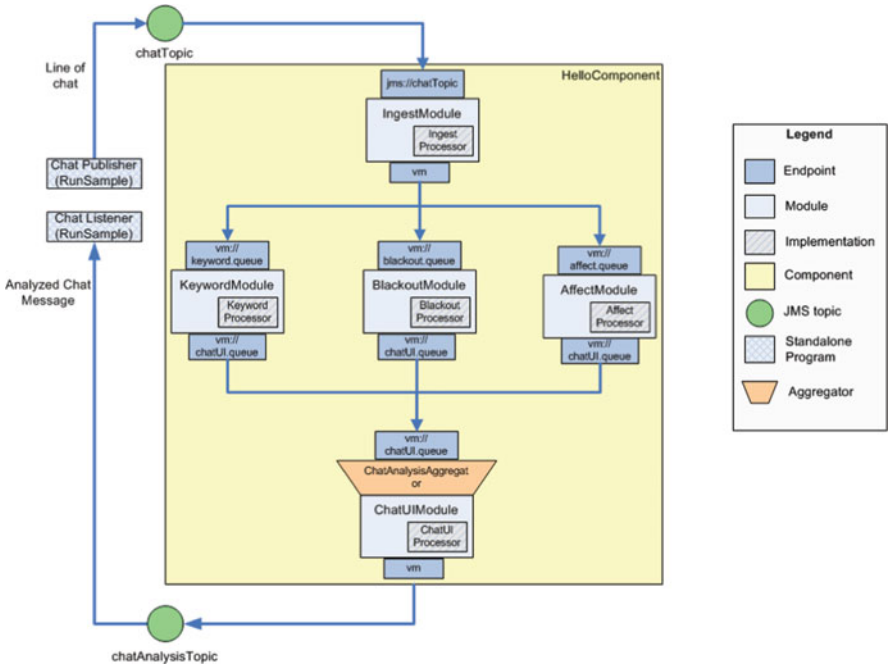


Fig. 10.2 MIF análisis de tuberías de chat 158

chat-especí código fi c llamado por el módulo de MIF y está más allá del alcance de esta descripción). A continuación, un agregador combina los tres resultantes objetos de datos en un solo mensaje que se envía fuera de la tubería para la exhibición, en el caso de este ejemplo, a la consola.

Vamos a examinar esta tubería con más detalle.

10.5.1 Inicializar Pipeline

En primer lugar, tenemos que crear una MifPipeline objeto que se necesita para iniciar y detener la canalización de procesamiento. UN MifPipeline También se utiliza para crear y registrar todos los objetos que se ejecutan dentro de la tubería.

```
MifPipeline tubería = new MifPipeline ();
```

A continuación, crear y añadir un conector JMS, dándole la dirección del servidor y el nombre del servidor **que se va a utilizar (en este caso utilizamos una ActiveMQ proveedor JMS).**

```
pipeline.addMifJmsConnector
        ( "Tcp: // localhost: 61616", JmsProvider.ACTIVEMQ);
```

A continuación, vamos a crear el ChatComponent oponerse, asignar los puntos finales, y añadirlo a la tubería. En esta etapa, podemos empezar la tubería de manera que esté listo para empezar a recibir mensajes. Como veremos más adelante, el trabajo pesado de la tubería con fi guración se encapsula en el ChatComponent componente.

```
ChatComponent chatear = new ChatComponent (); chat.setInEndpoint ( "JMS: //
tema: ChatDataTopic"); chat.setOutEndpoint

        ( "Stdio: RESULTADO // stdio outputMessage = CHAT:?");
pipeline.addMifComponent (chatear); pipeline.start ();
```

Por último, tenemos que invocar un método de utilidad que simula una corriente de mensajes de chat fl debido a la tubería a través de JMS mediante la lectura de un expediente de mensajes de chat y enviarlos a la tubería.

```
simulateChatStream ();
```

10.5.2 Componente de Chat

los ChatComponent encapsula el con fi guración de los módulos de aplicación en una tubería interna. En primer lugar, establecemos los criterios de valoración de los componentes que se transmiten desde el código de llamada (ChatComponentDriver.java en este caso). Nótese cómo el componente es ajeno al transporte que se asocia con los criterios de valoración, un tema JMS y stdout en este caso. Estos detalles se abstraen completamente en el código de componente, y por lo tanto el componente se puede utilizar para comunicarse a través de cualquier medio de transporte que está asociado con sus puntos finales.

```
setInEndpoint pública vacío (String inEndpoint) {this.inEndpoint = inEndpoint; }

setOutEndpoint pública vacío (String outEndpoint) {this.outEndpoint = outEndpoint;
 }
```

ChatComponent tiene un número de módulos internos. Primero, Módulo ingest- se encarga de tomar un mensaje de chat y analizarlo en una estructura de datos (MapWrapper) para ser procesado por todos los módulos de procesamiento aguas abajo. Este módulo tiene tres puntos extremos salientes desde el mensaje de salida se dirigirá a tres módulos de procesamiento aguas abajo (Afectar, apagón, y Palabra clave).

```
MifModule ingestModule = pipeline.addMifModule
            (Ingest.class.getName (), inEndpoint,

            "Vm: //ingest.keyword.queue"); // y añadir puntos
finales salientes adicionales
ingestModule.addOutboundEndpoint ( "vm: //ingest.affect.queue"); ingestModule.addOutboundEndpoint (
"vm: //ingest.blackout.queue");
```

A continuación, los módulos de procesamiento de aguas abajo están conectados mediante la creación de puntos finales entrantes que corresponden a la ingestModule puntos finales salientes (ingerir. keyword.queue En el ejemplo a continuación para la Palabra clave módulo (los otros funcionan de manera similar por lo que dejará a los de esta descripción)).

```
// Añadir PALABRA CLAVE Módulo
pipeline.addMifModule (Keyword.class.getName (),
 s 160          "Vm: ingest.keyword.queue", "vm:
                //keyword.queue");
```

Finalmente, el último paso del componente con fi guración es para agregar los resultados de los módulos de procesamiento en un solo mensaje y enviar el resultado fuera del componente utilizando el punto final de salida. Para lograr esto, se crea la chatAggregateModule y conectarse a él los tres puntos finales salientes de los tres módulos de aguas arriba.

```
MifModule chatAggregateModule =
 ejemplo, el valor de retorno podría representar la feria más bajo regresado después de un tiempo de espera de 30
                pipeline.addMifModule (ChatAggregate.class.getName () "vm: //keyword.queue",
                outEndpoint);

chatAggregateModule.addInboundEndpoint ( "vm: //affect.queue");
chatAggregateModule.addInboundEndpoing ( "vm: //blackout.queue");
```

Los agregadores son módulos especiales MIF que combinan mensajes de múltiples fuentes en un solo mensaje. Pueden ser utilizados para cotejar los resultados de los módulos que trabajan en paralelo (como en nuestro ejemplo de chat aquí) o para reducir un gran volumen de mensajes en un solo objeto. Para cotejar grupos de mensajes, una correlación identi fi er se debe añadir a cada mensaje. Los eventos que deben agregarse tienen grupo de agregados juntos. En una aplicación de la distribución de las solicitudes de las tarifas aéreas, por un valor de correlación idénticos, lo que permite el agregador para combinarlos. Cualquier módulo de MIF se puede asociar con un agregador que nes de fi cómo se combinan múltiples mensajes de entrada.

Para crear un agregador de MIF, necesitamos ampliar el AbstractMif- agregador clase abstracta. Para ello es necesario implementar dos métodos, shouldAggregateEvents y doAggregateEvents. Ambos de estos métodos de tomar una sola MifEventGroup objeto que contiene una lista de objetos que se han recibido en los puntos finales entrantes del devuelve un resultado verdadero para ese grupo. Devuelve un objeto que representa el valor de los objetos de ese agregador.

El primer método, shouldAggregateEvents, se llama cada vez que un nuevo mensaje es recibido por el agregador en ningún punto final. Su valor de retorno es un valor booleano, que indica si o no ese grupo de eventos contiene un conjunto completo que está listo para ser agregado en un solo mensaje. Las acciones típicas realizadas por este método incluyen contando el número de mensajes en el grupo de eventos (para la agregación de los resultados de un cierto número de procesos paralelos) o en busca de la presencia de un mensaje en particular (para digerir los mensajes que llegan durante un cierto período de tiempo).

El segundo método, doAggregateEvents, se pidió a un grupo de mensajes siempre que sea shouldAggregateEvents

se recibe un mensaje. En nuestro ejemplo, el ChatAnalysisAggregator es responsable de la combinación de los mensajes de los tres módulos aguas arriba. Esto se logra mediante el uso de un valor de correlación (es decir, un mensaje único identi fi er para cada mensaje que se asigna en la etapa de la ingesta) y la combinación de estos cuando se han recibido los tres mensajes (una para cada módulo de aguas arriba). En la API de MIF, simplemente creamos el agregador y adjuntarlo a themodule que debe ser associatedwith.

```
MifAggregator chatAnalysisAggregator =
pipeline.addMifAggregator (nuevo ChatAnalysisAggregator ());
chatAggregateModule.setAggregator (chatAnalsysiAggregator);
```

código 10.5.3 Implementación

Para este ejemplo, todos los módulos de procesamiento están escritos en Java. Se envuelven las bibliotecas de procesamiento de texto fi cas que realizan la lógica de la aplicación real. A continuación se muestra un ejemplo de uno de los módulos de procesamiento. los BlackoutProcessor, que implementa diversos algoritmos de protección de identidad, implementa la funcionalidad de la BlackoutModule. Esto representa un ejemplo muy común de la utilización de una clase contenedora para llamar a la lógica de procesamiento "real" (por lo general en una biblioteca o un tarro) sin necesidad de realizar ningún cambio en el código para incorporarlo en una tubería MIF.

En este caso, el código simplemente delegados al método de aplicación especí fi co blackout.processContentAnalysis (mensaje).

```
blackout.processContentAnalysis (mensaje).

BlackoutModule clase pública implementa MifInOutProcessor {
    // un montón de detalles omitidos
    BlackoutId privada apagón estática = null; BlackoutModule pública () {

        initBlackout (); }

    Serializable Público (entrada Serializable) {
        datos MapWrapper = (MapWrapper) de entrada; HashMap
        mensaje = data.getMap (); if (! apagón = null) {

            // lógica de procesamiento de llamada de prueba
            blackout.processContentAnalysis (mensaje); }

        volver nueva MapWrapper (mensaje); }}
```

10,6 generador de componentes

El componente MIF (CB) se basa en Eclipse y puede ser utilizado por los programadores para diseñar tuberías MIF, generar código auxiliar para los objetos en la tubería, implementar el código apagó, y ejecutar la tubería resultante. El CB es capaz

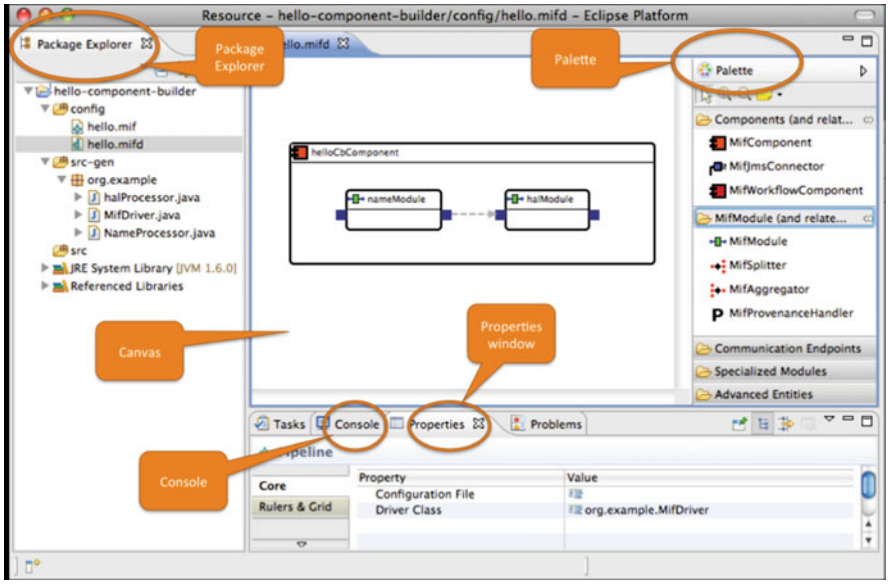


Fig. 10.3 MIF componente constructor 162

del desarrollo de ida y vuelta, que soporta la repetición del proceso de diseño, generar, implementar, ejecutar, hasta que el programador está satisfecho con la tubería. En ese punto, los componentes en la tubería se pueden implementar en una instancia de MIF correr o almacenan en una biblioteca de componentes para su uso posterior. Figura 10.3 muestra un ejemplo del uso del CB y pide a cabo las distintas ventanas que apoyan el desarrollo, a saber:

- Lona. El lienzo representa el diagrama fi l y es donde la tubería está

  con fi gurado. Los objetos se colocan en el lienzo y se conectan para crear una tubería. Cuando los objetos se mueven alrededor de la tela y el diagrama se guarda, los cambios se escriben en una. mifd fi l.

- Paleta. La paleta contiene todos los objetos que componen una tubería MIF

  modelo. Los objetos se seleccionan de entre la paleta y se colocan en el lienzo. La paleta separa los objetos en diferentes categorías para mayor comodidad. los componentes sección contiene componentes MIF y objetos comúnmente usados que pueden aparecer dentro de un componente. los MifModule sección contiene la MifModule

  objeto y otros objetos que se pueden colocar dentro de un módulo. los La comunicación de punto final sección contiene objetos de punto final más las EndpointLink que se muestra en el lienzo como una línea de puntos que conecta un punto final de salida en un módulo al punto final de entrada en la siguiente.

- explorador de paquete. El explorador de paquetes se utiliza para la organización de los archivos fuente,

  con fi guración de archivos, y el FOMIN modelo de archivos.

- La ventana de propiedades. Propiedades de los objetos colocados en el lienzo se editan mediante el

  Eclipse “Propiedades” de la ventana. Selección de una entidad en la tela hace que sea posible ver y editar las propiedades necesarias en la ventana de propiedades. propiedades en

toda la tubería también se puede ajustar haciendo clic en el lienzo Eclipse, y luego mediante la edición de las propiedades que aparecen en la ventana de propiedades. Por ejemplo, tenga en cuenta en la Fig. 10.3 que el Clase controlador propiedad tiene el valor org.example.MifDriver.

Esta propiedad particular significa que este será el nombre de clase de la clase del controlador generado que corre la tubería creada en el generador de componente.

- Consola. La consola es igual que la consola de Java en el que se muestra estándar entrada y salida estándar dentro de la IDE. Además, la consola CB da salida también a estado y los resultados de las acciones de generación de código.

Cada vez que un diseño de MIF se guarda, el modelo subyacente MIF se comprueba la validez. Esto proporciona una característica de comprobación de errores que se implementa mediante la imposición de restricciones en los objetos en el modelo, tales como "una MifModule no debe tener un fi nida unde implementationClass. "Si tal restricción no se satisface, el CB coloca un rojo'icono X'sobre el objeto (s) que viven en el error. Cuando el usuario mueve el ratón sobre uno de estos iconos de error, el CB ofrece una pista sobre cuál es el problema.

10.7 Resumen

Hemos utilizado MIF en varias aplicaciones en los últimos 3 años, en ámbitos tan diversos como la bioinformática, la seguridad cibernética, la modelización del clima, y análisis de red de energía eléctrica. En todos estos proyectos, el FOMIN ha demostrado ser robusto, ligero y altamente flexible.

En la construcción de la tecnología Medici, hemos tenido cuidado de aprovechar las tecnologías existentes siempre que sea posible, y construir el software tan poco como sea posible para apoyar la tubería y abstracciones basadas en componentes que nuestras aplicaciones requieren. Parte del éxito de Medici, por lo tanto, sin duda, radica en las fortalezas de sus fundaciones - mula, ActiveMQ - que proporcionan potencia industrial, plataformas ampliamente desplegados. Vemos esto como un modelo razonable para otros proyectos a seguir, especialmente en la comunidad de investigación científi ca donde los recursos para el desarrollo de tecnologías de clase middleware son escasos.

10.8 Lectura adicional

El proyecto completo Medici es de código abierto y disponible para su descarga desde http://medici.pnl.gov . El sitio contiene una cantidad considerable de documentación y varios ejemplos.

También hemos escrito varios artículos que describen la estructura y las aplicaciones que hemos construido. Algunas de ellas se enumeran a continuación:

I. Gorton, H. Zhenyu, Y. Chen, B. Kalahar, B, S. Jin, D. Chavarria-Miranda, D. Baxter, J. Feo, Un híbrido de alto rendimiento Informática aproximación al análisis de contingencia masiva en la red de alimentación, e-Ciencia, 2009. e-Ciencia '09. Quinta Conferencia Internacional IEEE sobre e-Ciencia, pp. 277-283, 9-11 de diciembre de 2009.

I. Gorton, A. Wynne, J. Almquist, J. Chatterton, La Integración marco Medici
EE.UU., IEEE: 164
trabajo: una plataforma para aplicaciones de alto rendimiento de secuencias de datos, wicsa, pp. 95-104,
Séptima Conferencia de Trabajo IEEE / IFIP en Arquitectura de Software (WICSA 2008), 2008.
Computing Tecnología y Ciencia (CloudCom 2010) 30 de noviembre - 3 de diciembre de la Universidad de Indiana,

I. Gorton, Y. Liu, J. Yin, Explorando Opciones de arquitectura para un federados, Cloud-
basado base de conocimientos biología de sistemas, en la segunda Conferencia Internacional IEEE sobre Cloud

Capítulo 11
Viendo hacia adelante

11.1 Introducción

El mundo de la tecnología de software es un movimiento rápido y el lugar en constante cambio. A medida que nuestros conocimientos de ingeniería de software, métodos y herramientas mejoran, también lo hace nuestra capacidad para abordar y resolver problemas cada vez más complejos. Esto significa que creamos aplicaciones “más grandes y mejores”, sin dejar de subrayar los límites de nuestros conocimientos de ingeniería de software cada vez mejores. No es sorprendente que muchos en la industria sienten que están parados. No parecen ser bene fi ción de las ganancias de productividad y de calidad prometidos de los enfoques de desarrollo mejoradas. Sospecho que está destinado a ser la vida de todos nosotros en la industria del software por lo menos durante el futuro previsible.

11.2 Los retos de la Complejidad

Es la pena detenerse por un momento para considerar lo que podría ser algunos de los principales retos para los fabricantes de sistemas de TI en los próximos años. Es probablemente bastante polémica al afirmar la inevitabilidad de que las aplicaciones de negocio continuarán siendo cada vez más y más compleja. La complejidad es un atributo multidimensional sin embargo. ¿Qué aspectos de la complejidad exactamente tienen más probabilidades de influir en la forma en que diseñamos y construimos la siguiente generación de aplicaciones?

Desde un punto de vista comercial, parece muy probable que el siguiente será controladores para gran parte de lo que hace la profesión de TI en la próxima década:

- **Empresas insistirán su infraestructura de TI apoya cada vez más complejo**
  los procesos de negocio que aumentan su organización e fi ciencia y reducir su costo de los negocios.
- **Para muchas empresas, la tasa de cambio en su entorno empresarial**
  exigir a sus sistemas de TI a ser fácil y rápidamente adaptable. Agilidad en la forma en que una empresa responda a sus necesidades de negocio tendrá un impacto en su línea de fondo.
- **Las empresas siempre quieren una mayor beneficio de ella y para reducir de forma simultánea**
  sus costos de TI. Demasiadas empresas se han visto desperdicio masivo en los sistemas de TI sin éxito. Como consecuencia, ahora necesitan seriamente convincente de la necesidad

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3_11, # Springer-Verlag Berlin Heidelberg 2011

invirtiendo fuertemente en ella, y va a insistir que su departamento de TI continuamente “hacer más con menos”.

Vamos a discutir cada uno de estos y ver qué consecuencias pueden tener, sobre todo desde la perspectiva de un arquitecto de TI.

11.2.1 Procesos de Negocio Complejidad

En las grandes empresas, los procesos de negocio de alto valor abarcan múltiples, inevitablemente, aplicaciones de negocios independientes, todos los que operan en una infraestructura de TI altamente heterogénea. En este tipo de entornos, las herramientas y tecnologías para procesos de negocio de fi nición y la promulgación sido de importancia crítica. En términos prácticos, esto significa que las tecnologías de orquestación de procesos de negocio son propensos a convertirse en productos básicos, componentes de misión crítica en muchas empresas.

necesarios en términos de fiabilidad y seguridad? Tiene que haber algún mecanismo para describir los niveles de
herramientas de orquestación de procesos de negocios de hoy en día están probadas y fabricados, y los maduros son cada vez más capaces de soportar altas cargas de peticiones y a escala. Sin embargo, hay algunos problemas fundamentales que actualmente se encuentran fuera de sus capacidades. Probablemente la necesidad clave es pasar de “estática” de procesos “dinámicas”. Que significa exactamente?

proceso de decidir qué? ¿Cómo sabe qué socio potencial proporcionará el proceso con los niveles de servicio

Un objetivo muy atractivo para los procesos de negocio es la composición dinámica. Por ejemplo, una organización puede tener una acción de compra de procesos de negocio definida por la compra de los proveedores. Inesperadamente, un proveedor va a la quiebra, o en otro aumenta los precios por encima del umbral de la organización quiere pagar. Con las tecnologías actuales, es probable que el proceso de negocio tendrá que
una o más posibilidades para el proceso de negocio para conectarse a. Suponiendo más de uno, ¿como el ser modi fi cado manualmente para comunicarse con un nuevo proveedor. Esto es costoso y lento.

Idealmente, un proceso de negocio sería capaz de “automágicamente” sí figura Recon fi, siguiendo un conjunto de reglas de negocio para conectarse a un nuevo proveedor y restablecer una relación de compra. todo esto iba a
exible fl buscado en base a una serie de propiedades. Eso no es demasiado duro, y una búsqueda podría producir pasar en unos pocos segundos, aliviando la necesidad de participación programador.

Este tipo de evolución dinámica de procesos de negocio no es demasiado difícil, siempre y cuando el entorno es muy limitada. Si hay un fi jo, conjunto conocido de socios potenciales, cada uno con conocida (idealmente las mismas)
tiene que encontrar un nuevo socio adecuado. Esto requiere alguna forma de directorio o registro, que puede
las interfaces, a continuación, los procesos de negocio pueden ser construidos para modificar su comportamiento cuando se producen ciertas condiciones (como una interfaz pareja desaparece). Sin embargo, una vez que estas limitaciones se eliminan, todo el problema se vuelve exponencialmente más difícil.

Para empezar, si los socios comerciales potenciales no son conocidos de antemano, el proceso de negocio

Una vez que un socio de confianza ha sido seleccionado en base a los niveles de servicio que anuncian, es casi necesario fi gura exactamente cómo comunicarse con el socio. No hay garantía de que cada socio sea posible tiene la misma interfaz y acepta y entiende el mismo conjunto de mensajes. Por lo tanto es necesario que el proceso de negocio que solicita para asegurar que envía solicitudes en el formato correcto.

El problema asesino aquí es que aunque una interfaz será normalmente sólo describir el formato de las solicitudes que recibe y envía, y no la semántica de los datos en la solicitud. Esto significa que un mensaje que indica que el precio de un itemmay o no ser en dólares estadounidenses. Si es en euros, y se está esperando dólares de EE.UU., a continuación, en función de los tipos de cambio, es posible que en un choque o una sorpresa agradable.

En sus formas generales, estos problemas de confianza descubrimiento y semántica de datos son más o menos sin resolver. Se están realizando esfuerzos para abordar los problemas de descubrimiento y de confianza con las tecnologías de servicios Web y los problemas semánticos con una colección de tecnologías conocidas como la Web Semántica, que se describen en el capítulo. 12.

11.3 de la agilidad

La agilidad es una medida de la rapidez con que una empresa puede adaptar sus aplicaciones existentes para apoyar las nuevas necesidades de negocio. Si una empresa puede obtener un nuevo servicio en línea de negocios antes que sus competidores, puede empezar a ganar dinero mientras que la competencia se esfuerza para ponerse al día.

Desde un punto de vista arquitectónico, la agilidad está muy estrechamente relacionado con la capacidad modi fi. Si la arquitectura de una empresa es imprecisa y dependencias de las aplicaciones y la tecnología son abstraído detrás de las interfaces sensibles, la implementación de nuevos procesos de negocio podría no ser demasiado oneroso.

Una auténtica barrera a la agilidad es la heterogeneidad. Una arquitectura podría ser muy bien diseñado, pero si, por ejemplo, de repente se hace necesario obtener una nueva aplicación .NET hablar con aplicaciones J2EE usando un JMS existente, entonces la vida puede ser un poco desordenado. En realidad, el gran número de combinaciones incompatibles de tecnología en una empresa por lo general no es algo que es placentero que pensar.

Como se describe en el Cap. 5, los servicios web SOAP y REST son basados en tecnologías útiles que son ampliamente utilizados para vincular entre sí los sistemas heterogéneos. Ellos definen un protocolo estándar y mecanismos para tapar juntas aplicaciones, tanto dentro como fuera de las empresas.

servicios web traiga una mayor agilidad a través de la integración basada en estándares. Pero la integración no es el único impedimento para el aumento de la capacidad de una empresa para modificar y ofrecer nuevas aplicaciones. La mejora de las tecnologías de desarrollo que hacen que cambian menos difícil y costoso también pueden aumentar en gran medida la agilidad de una empresa. Dos enfoques emergentes en este sentido son las tecnologías orientadas a aspectos y Arquitecturas dirigida por modelos (MDA).

tecnologías orientadas a aspectos de la estructura de una aplicación como un conjunto de independientes pero relacionados “aspectos” de una solución, y proporcionar herramientas para fusionar estos aspectos en

construir o en tiempo de ejecución. Como se pueden crear aspectos, comprendido y modi fi cado de forma independiente, que mejoran la agilidad de desarrollo.

MDA, o ningún modelo basado en el desarrollo, ya que está aumentando conocida, promueve el desarrollo de aplicaciones utilizando modelos abstractos basados en UML de una solución. código ejecutable se genera a partir de estos modelos utilizando herramientas de MDA. MDA eleva el nivel de abstracción del proceso de desarrollo, en teoría más fácil hacer cambios a efectuar en los modelos y no en código detallado. MDA herramientas de generación de código también esconden el conocimiento fi co-plataforma específica detallada de la aplicación. Por ejemplo, si la plataforma subyacente (por ejemplo, tecnología MOM) cambia, un generador de código para la nueva plataforma se adquiere simplemente. La aplicación puede entonces ser regenerado de forma automática a partir del modelo a utilizar la nueva plataforma. Ahora hay agilidad para usted! Esa es la teoría, de todos modos.

Aspectos y MDA se describen en Chaps. 13 y 14 respectivamente.

poner a prueba todas estas características ampliamente. 168

11.4 Reducción de costes

Los días embriagadores de finales de 1990 “dot.com” boom y masiva que gasta han quedado atrás, y no hay ninguna señal de su regreso. Ahora las empresas con razón, exigen saber qué negocio beneficio de sus inversiones en TI traerá, y qué retorno oninvestment que pueden esperar. Como arquitecto, la escritura de rentabilidad para las inversiones y adquisiciones es una habilidad que necesita para adquirir, si no lo ha hecho, por supuesto.

En cuanto a la reducción de lo que pasamos, sin dejar de lograr nuestros objetivos de negocio, el lugar para comenzar es comenzar por trabajar más inteligentemente. En su conjunto, la industria de TI ha tenido dificultades para cumplir con las promesas de una mayor e fi ciencia y menores costos de adopción de nuevas tecnologías de desarrollo. tecnologías de objetos y componentes estaban destinados a hacer más fácil para nosotros para diseñar y ofrecer componentes reutilizables que podrían utilizarse en muchas aplicaciones. Construir algo una vez, y lo utilizan

fueron diseñados originalmente para. Usted tiene que añadir más características para atender a un uso más general. Es necesario

para esencia, sin costo, muchas veces más. Eso es un trato nadie puede negarse, y uno que es simple para la gestión de entender.

La verdad es que la industria de TI prácticamente ha dejado de cumplir con la promesa reutilización. reutilización exitosa tiende a tener lugar con gran escala, como los componentes de la infraestructura de middleware y bases de datos. Del mismo modo, Enterprise Resource Planning (ERP) como SAP y sus semejantes han logrado entregar, procesos de negocio personalizables generalizadas a un amplio espectro de organizaciones. Ninguno de ellos ha sido sin sus di fi cultades, por supuesto. Pero pensar en la cantidad de código de otra persona (es decir, la inversión) que está utilizando al implementar una base de datos Oracle o un servidor de aplicaciones JEE. Es significativo hecho.

Pero en una escala más pequeña, componentes reutilizables han tenido menos impacto. La razón de esto es simple y bien explicado por mucha investigación en la comunidad de ingeniería de software. El argumento dice así.

Esencialmente, cuesta dinero para construir componentes de software para que puedan ser utilizados en un contexto que no

Es necesario documentar las características y crear ejemplos de cómo utilizar el componente. Los estudios indican que cuesta entre tres y diez veces más para producir componentes reutilizables de calidad.

Por supuesto, toda esta inversión puede ser útil si los componentes se utilizan una y otra vez. Pero ¿y si no lo son? Bueno, básicamente, que acaba invertido mucho tiempo y esfuerzo en la generalización de un componente para ningún propósito. Eso no es inteligente.

Afortunadamente, algunos arquitectos muy inteligentes pensaban sobre este problema hace unos años. Se dieron cuenta de que la reutilización éxito no acaba de suceder “por arte de magia”, pero podría lograrse si una estrategia de producto se entendía y planeado. Por lo tanto se acuñó el término “arquitectura de línea de productos”. Estos se explican en el capítulo. 15. Ellos representan un conjunto de prácticas probadas que pueden ser adoptadas y adaptadas dentro de una empresa para aprovechar las inversiones en arquitecturas y componentes de software. líneas de productos de software representan el estado del arte en el trabajo inteligente en este momento.

11.5 Lo siguiente

Los siguientes cuatro capítulos de este libro cada cubren un área de práctica o la tecnología que es probable encontrar en una vida de un arquitecto de software. Estos son:

- La Web Semántica
- Programación Orientada a Aspectos
- Arquitecturas basada en modelos (MDA)
- Líneas de Producto Software

Cada uno de los capítulos que siguen se describen los fundamentos de cada enfoque, aborda el estado de la técnica, y especula sobre el futuro potencial y adopción. También describen cómo las técnicas o tecnologías se pueden aplicar al caso de estudio ICDE para proporcionar características mejoradas y funcionalidad.

Esperamos que estos capítulos le brazo con el conocimiento su fi ciente para al menos parecen inteligente e informada cuando un cliente o alguien de su proyecto en el que las capturas por sorpresa y sugiere la adopción de uno de estos enfoques. En tales circunstancias, un poco de conocimiento puede recorrer un largo camino.

capítulo 12

La Web Semántica

Judi McCuaig

12.1 ICDE y la Web Semántica

Intercambiar y compartir datos es un reto fundamental para la integración de aplicaciones. La plataforma ICDE ofrece una instalación de noti fi cación para permitir herramientas de terceros para el intercambio de datos. Supongamos que un usuario ICDE está trabajando con una herramienta de terceros para analizar los registros de transacciones financieras de varias organizaciones. La herramienta genera una lista de palabras clave relacionadas con fi nanciar-para describir el conjunto de registros de transacciones después de un análisis complejo y almacena esta lista en el almacén de datos ICDE. Supongamos, además, que este usuario ICDE ha puesto en marcha otras herramientas de terceros para utilizar sus datos ICDE como entrada a los procesos de herramientas. Una de estas herramientas utiliza la lista de palabras clave almacenada para realizar una búsqueda de información nueva, nunca antes vista en relación con el análisis de las transacciones fi nancieras en curso.

Este escenario sólo es posible cuando las herramientas cooperantes tienen la capacidad de compartir datos. El intercambio debe incluir una comprensión de consenso de la semántica de los elementos de datos que se comparte. Muy a menudo, este consenso se logra mediante la creación de una estructura de datos que se acopla a cada aplicación que utiliza los datos compartidos. La estructura de datos define el formato (por ejemplo, lista, tabla) y la semántica (por ejemplo, nombre del documento, título del documento, ubicación del documento, tema del documento, etc.) de los datos compartidos.

En el marco del ICDE, que compartió comprensión podría ser alcanzado mediante la publicación de una estructura de tabla y que requieren todas las aplicaciones que colaboran para utilizar esa estructura para compartir datos. Sin embargo, el equipo de desarrollo del ICDE nunca podría anticipar las estructuras de datos adecuadas para todas las herramientas de terceros y cada dominio de aplicación en la que operaría ICDE. Nuevas tablas se podrían añadir, por supuesto, pero cada herramienta proveedor de terceros tendrían que negociar con el equipo ICDE para conseguir una adecuada estructura de datos definida, por lo que la integración de herramientas ágiles imposible.

Un enfoque más flexible permitiría herramientas de terceros para publicar datos a través del almacén de datos ICDE utilizando cualquier estructura adecuada. Posteriormente, cualquier otra herramienta autorizada debe ser capaz de descubrir de forma dinámica la estructura de los datos publicados y entender la semántica del contenido. No es necesaria la previa del conocimiento, no modificable de estructuras de datos.

El requisito obvio para una solución flexible tal fl es utilizar estructuras de datos auto-describen para los datos publicados. Extensible Markup Language (XML)

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3_12, # Springer-Verlag Berlin Heidelberg 2011

que bastar, como cualquier programa puede analizar dinámicamente un documento XML y navegar por la estructura de datos. Sin embargo, rawXML no admite semántica descubrimiento hacer comprender ad hoc de los datos problemáticos. Por ejemplo, una herramienta de terceros puede utilizar la etiqueta XML < ubicación> para indicar la ubicación de alguna información, mientras que otro puede usar < URI>, y otro < nombre de ruta>. La semántica de estos nombres de las etiquetas dicen un lector humano que cada etiqueta contiene la misma información, pero no hay manera de hacer que esa conclusión mediante programación utilizando sólo XML. Obligando a todas las herramientas a utilizar el mismo vocabulario estricta etiqueta no es ningún más flexible que obligar a todos a utilizar la misma estructura de datos.

Lo que se necesita es un mecanismo para compartir la semántica del vocabulario elegido, lo que permite el para integrar dinámicamente las aplicaciones de software. 172
descubrimiento programática de términos que describen conceptos similares. El uso de un mecanismo de este tipo, una herramienta puede determinar que < URI> y < ubicación> son en realidad el mismo concepto, aunque la relación no es explícitamente definida en el software o los datos publicados.

La solución a este problema radica en el conjunto de tecnologías asociadas a la Web Semántica. La Web Semántica hace que sea posible describir los datos en formas que hacen su semántica explícita y por lo tanto Semántica. Proporcionan herramientas y enfoques para la gestión de metadatos que son genéricamente útil detectable de forma automática en el software. Una de las innovaciones clave está en el uso de ontologías, que describen los conceptos relevantes en un dominio, y la recogida de las relaciones entre esos conceptos.

Este capítulo presenta las tecnologías básicas de la Web Semántica. A continuación, muestra cómo las ontologías de dominio podrían ser utilizados en la plataforma ICDE para apoyar la facilidad de integración de proveedores de herramientas de componentes. Son precisamente estos problemas que las tecnologías que componen frente a la Web terceros.

12,2 automatizada, Integración distribuida y Colaboración

metadatos apropiados para facilitar la interacción dinámica con las disponibles de información, servicios y Las di fi cultades asociadas con la integración de software han plagado a los ingenieros de software desde los primeros días de la industria de la computación. Los esfuerzos iniciales de integración (ignorando los problemas de hardware y almacenamiento de interoperabilidad) centrados en hacer accesibles los datos a múltiples aplicaciones, normalmente a través de algún tipo de sistema de gestión de base de datos.

Más recientemente se han hecho esfuerzos para crear procesos interoperables utilizando componentes utilizando servicios y componentes autónomos. Los principales desafíos incluyen la creación, gestión y utilización de los tecnologías como CORBA o JEE. Como se ha explicado en los capítulos anteriores, las arquitecturas orientadas a servicios y servicios Web son las últimas tecnologías para dar a los diseñadores de software la oportunidad de crear sistemas de software pegando servicios, posiblemente de una variedad de proveedores, para crear un sistema de software especializado diseñado para que una empresa particular, problema.

Hay di fi cultades asociadas con la localización, integración y mantenimiento de un sistema compuesto de

12.3 La Web Semántica

El propósito de la iniciativa de la Web Semántica es la creación de la máquina información comprensible, donde la semántica son explícitas y utilizable por los algoritmos y programas informáticos. Este objetivo original se ha ampliado para incluir el objetivo de crear servicios o procesos, que son la máquina comprensible y utilizable por otros procesos. Este entendimiento compartido, ya sea de datos o servicios, es posible gracias a una amplia colección de lenguajes de descripción de metadatos y protocolos. En su mayor parte, existe la Web Semántica a causa de estos idiomas.

La interoperabilidad prometida por las tecnologías de la Web Semántica es posible gracias a:

- La formalización de la representación de metadatos
- El desarrollo permanente de la representación del conocimiento
- técnicas de lógica y razonamiento que pueden aprovechar tanto los metadatos y la conocimiento representado

Las capacidades clave que se ofrecen son la representación flexible de metadatos y relaciones, codificado como ontologías. Estos permiten la traducción entre vocabularios de metadatos y el razonamiento acerca de las entidades de metadatos representados.

Figura 12.1 ilustra las relaciones entre algunas de las tecnologías asociadas con la Web Semántica. XML, Unicode, y Uniformes de Recursos los identificadores (URI) forman la columna vertebral y permiten el almacenamiento y recuperación de información. El Marco de Descripción de Recursos (RDF) es la base para describir la estructura de la información dentro de las aplicaciones de la Web Semántica. Ontologías, frecuentemente codifican utilizando el Lenguaje de Ontologías Web (OWL) y taxonomías describen usando la descripción de recursos de estructura de esquema (RDFS), proporcionar la capa en

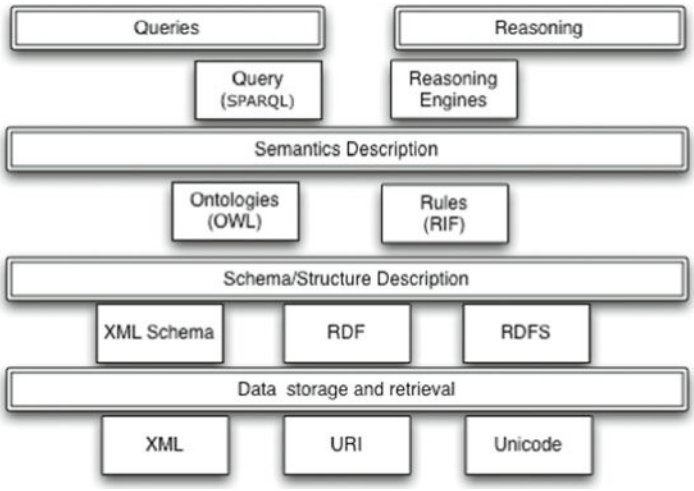


Fig. 12.1 tecnologías de Web Semántica

que la semántica de la información pueden ser descritos y puestos a disposición de las aplicaciones. Una capa adicional de computación proporciona facilidades para consultas y el razonamiento acerca de la información disponible. aplicaciones de Web Semántica son típicamente escritos en la parte superior de esta capa de consulta y el razonamiento.

12.4 Creación y uso de metadatos para la Web Semántica

Las capacidades avanzadas asociadas a la Web Semántica provienen casi en su totalidad en la parte posterior de grandes esfuerzos en la creación y el mantenimiento de los metadatos. La introducción del XML y las tecnologías relacionadas con ella proporcionado un mecanismo estructurado, flexible para describir datos que se entiende fácilmente por las máquinas (y un subconjunto de los seres humanos que les gusta paréntesis angulares). XML proporciona los medios para entidades de etiqueta y sus partes, sino que proporciona capacidades únicas débiles para describir las relaciones entre las dos entidades. Por ejemplo, considere el fragmento de XML en la Fig. 12.2 . En él se describe una Persona en términos de Nombre, Dirección de correo electrónico, y Número de teléfono, y una

Transacción en términos de Tipo, cliente, y Número de cuenta. El ejemplo también muestra el uso de atributos para crear res únicos identificadores (carné de identidad) para cada entidad.

XML embargo, no es adecuada para una fácil identi fi cación de las relaciones entre piezas de información. Por ejemplo, utilizando sólo los metadatos etiqueta XML en la figura, la identi fi cación de la dirección de correo electrónico de la persona que lleva a cabo una transacción específico es algo complejo. Se basa en la capacidad de determinar que la

Cliente ámbito de la transacción representa el nombre de una persona y que si el Cliente

los datos de campo coincide con el Nombre ámbito de una persona una relación puede ser identificados y dirección de correo electrónico de la persona utilizada.

Un ser humano puede hacer rápidamente esa determinación porque un ser humano entiende que las etiquetas Cliente y Nombre significar tanto la información acerca de las personas. Un proceso de software por desgracia no tiene esa capacidad, ya que no tiene ninguna manera de representar esa semántica.

```
<Ejemplo>
        <Persona id = "123">
                <Nombre> J Doe </ nombre>
                <Email_address> DOE @ myplace </ email> <phone_number> 123
            456 7899 </ phone_number> </ Persona>

        <Transacción transID = "567">
                <Downtick> 500 </ downtick> <Client> Josef Doe </ Client>
            <ACCOUNTNUMBER> 333222111 </ ACCOUNTNUMBER> </
            Transacción> </ example>
```

Fig. 12.2 ejemplo XML 174

Para hacer frente a este problema, el RDF fue desarrollado como una representación comprensible máquina de relaciones entre las entidades. Se supone que cada entidad y la relación se pueden identificar fi con un URI. Estos URI se utilizan para formar una declaración RDF de la forma {sujeto, predicado, objeto}, comúnmente llamado un “triple”.

Para continuar con el ejemplo anterior discusión, la adición de una relación de RDF llevada a cabo por (ver ejemplo RDF más abajo) entre la transacción y la persona (utilizando el carné de identidad atributos como el único er fi cación) permite que una máquina para extraer la dirección de correo electrónico del propietario de la transacción, sin que se requiera la replicación de la información. La declaración RDF a continuación indica que la persona que hace referencia ID # 123 lleva a cabo la transacción que hace referencia ID # 567.

<Http://example.net/transaction/id567> <http://example.net/conduc ted_by>
<http://different.example.net/person/id123>

La relación es explícita y fácilmente explotada usando programas de ordenador una vez por humanos identi fi ca y los registros de la existencia de la relación. RDF no resuelve todo el problema sin embargo, porque todavía no existe un mecanismo para identificar automáticamente las relaciones o al detalle de cualquier restricción de los participantes en esas relaciones. Por ejemplo, un ser humano entiende rápidamente que una transacción puede ser realizada por una persona, pero que una persona no puede llevarse a cabo mediante una transacción! El RDF en el ejemplo no tiene tales restricciones, por lo que los algoritmos de procesamiento de la RDF no hay manera de verificar los tipos o atributos esperados de las entidades en las relaciones.

Una solución parcial a la relación de identi fi cación problema se encuentra en los lenguajes de esquema de XML y RDF. Los lenguajes de esquema permiten a priori definición de entidades y relaciones que incluye dominios y rangos para los atributos y entidades. Entidades (o relaciones) que hacen referencia al esquema para su de fi nición a continuación, se puede comprobar su coherencia con el esquema. Los programas se pueden hacer cumplir las restricciones de rango y tipo de datos durante el procesamiento de datos sin intervención humana.

Junto RDF, XML y sus lenguajes de esquema proporcionan un método robusto, que puedan utilizarse para la codificación de metadatos y explotarlo para identificar automáticamente las relaciones entre las entidades. Sin embargo, nuestra kitbag de tecnologías esenciales para la comprensión automática de metadatos también tiene la capacidad de hacer deducciones e inferencias sobre los metadatos.

Consideremos de nuevo el ejemplo de transacción y el cliente. La finalización de una transacción es generalmente el resultado de la colaboración entre varias personas, incluyendo un cliente, consultor fi nanciero, y el secretario por ejemplo. Sería trivial para modificar el ejemplo de metadatos XML dado anteriormente para representar tanto el consultor y empleado como parte de los metadatos de la transacción, por tanto, que representa explícitamente la relación entre la transacción y los individuos que colaboran.

Sin embargo, la colaboración entre cualquier par particular de esas tres entidades (consultor, cliente, ventas) no está representado explícitamente en los metadatos. Un programa que tiene que identificar el cliente y el consultor para una transacción no tiene ningún mecanismo para determinar si los clientes y consultores fi cos se conocen entre sí utilizando nuestro actual conjunto de metadatos. Una forma de solucionar este problema es

sólo tiene que añadir más metadatos e identificar explícitamente la relación cliente-consultor, pero incluso en este pequeño ejemplo, es evidente que los metadatos superaría los datos rápidamente en cantidad. Una solución más general es a de reglas lógicas fi ne que delinean las posibles deducciones con los diferentes tipos de metadatos. Esas reglas lógicas de fi ne la semántica asociada con los metadatos y se describen con frecuencia en relación con la definición de una ontología formal. Las ontologías se explican en la siguiente sección.

Bien de fi nida y ordenó a los metadatos es la columna vertebral de la Web Semántica. Los metadatos se utilizan para ensamblar dinámicamente datos de una variedad de fuentes, para tomar decisiones informadas, y para proporcionar datos para la planificación de las cosas tales como, por ejemplo, las vacaciones y el envío de mercancías. Mientras que las tecnologías de metadatos se utilizan con más frecuencia con la información basada en la Web en el momento, que se pueden utilizar con el mismo poder para identificar las conexiones entre los servicios de software para los fines de la creación de cualquier sistema de software.

razonamiento podría deducir, dado el 176

12.5 La semántica de poner en el Web

La única característica que distingue a la Web Semántica de la World Wide Web es la representación y la la relación no se indica explícitamente en los metadatos disponibles o en el esquema. Un sistema de utilización de significado o semántica. Una representación común para la semántica es una ontología. Una ontología consiste en un conjunto de ideas o conceptos y la colección de las relaciones entre esos conceptos.

Una ontología se puede utilizar para identificar las ideas que están relacionadas entre sí y para proporcionar la estructura y las reglas para un motor de razonamiento para hacer inferencias sobre esas ideas. Una ontología modelos tanto de abstracción y relaciones de agregación. ontologías modelo de dominio especí fi cas relaciones más complejas sobre los individuos y las clases de la ontología también. Ontologías también pueden proporcionar información sobre los conceptos que son equivalentes a otros conceptos. Cuando adecuadamente complejo, una ontología puede proporcionar la correspondencia entre diferentes vocabularios de metadatos, haciendo que la integración de los procesos de software mucho más simple.

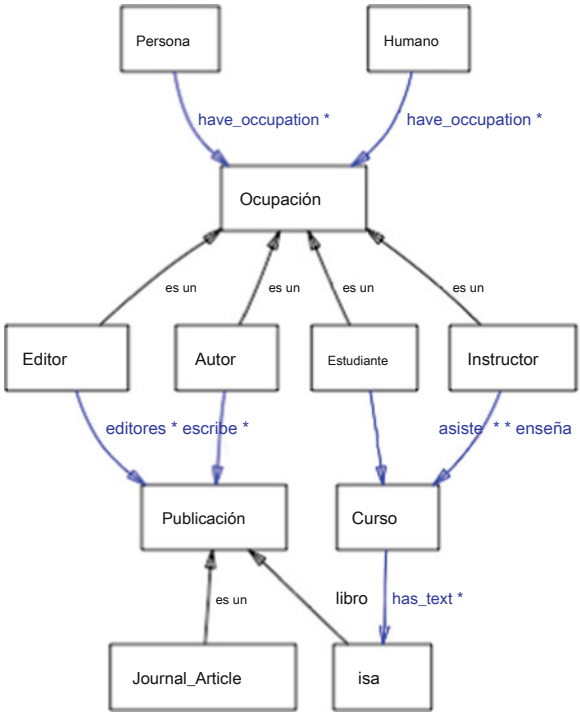
lógica para identificar automáticamente una relación entre un cliente y un asesor fi nanciero, incluso cuando

Por ejemplo, considere los fragmentos ontología representados en la Fig. 12.3 . los ontología muestra que Los seres humanos y personas tener ocupaciones y que ciertos tipos de ocupaciones tener relaciones con otros conceptos en la ontología. Ambos estudiantes y instructores tienen que ver con cursos y ambos autores y editores tienen que ver con Publicaciones. Esta ontología podría ser utilizado por un sistema automatizado para identificar entidades relacionadas o identificar el uso de conceptos equivalentes (tales como Humano y Persona en este ejemplo). La ontología proporciona axiomas lógicos a un sistema de razonamiento, que luego pueden hacer inferencias sobre la información.

Dentro de la Web Semántica, el búho es una representación común de los axiomas y conceptos de dominio.

Considere una vez más el ejemplo de la transacción financiera. Una ontología podría proporcionar la

Fig. 12.3 ejemplo ontología



reglas o entrenamiento correcto, que un cliente y consultor se conocen el uno al otro si han colaborado en un número fi cado de las transacciones. Una norma adicional podría indicar que si han colaborado en más de un tipo de transacción, que son bien conocidos entre sí.

En conjunto, los datos de la transacción financiera, los metadatos y la ontología conforman una base de conocimientos que no sólo proporciona información sobre las transacciones fi nancieras y los clientes, sino que también se pueden utilizar para identificar las relaciones entre los seres humanos especí fi cas. Información acerca de las relaciones cliente-consultor podría ser útil a alguien analizar las transacciones financieras con el fin de identificar los conjuntos o grupos de personas que llevan a cabo las clases especí fi cos de las transacciones (es decir, las transacciones que ocurren en un período de tiempo determinado), o tal vez para las organizaciones que necesitan para determinar el alcance de determinados consultores financieros.

Una ontología también puede contener reglas que limitan las relaciones. Supongamos que el ejemplo ontología contenía una regla que se opone a la misma persona de ser el cliente y el empleado para una transacción. La ontología podría utilizarse, en conjunción con un motor de razonamiento, para detectar errores en la información o para evitar errores en la entrada de datos. Ontologías proporcionan un significado para los metadatos que forman la columna vertebral de la Web Semántica.

XML, RDF, OWL y son las tecnologías básicas que apoyan la Web Semántica, que ahora está empezando a aparecer en la corriente principal de la web y en aplicaciones industriales

aplicaciones. El Hacker Semántica [1] es un ejemplo de una demostración independiente de las posibilidades de la Web Semántica para el descubrimiento de información. Ontoprise [2] utiliza tecnologías de Web Semántica para desarrollar sistemas de solución de problemas y la validación de diseños que funcionan como sistemas expertos con más flexibilidad en la definición y mantenimiento de datos y reglas. Sus clientes incluyen a los fabricantes de automóviles, fabricantes de robots industriales y empresas de inversión.

Una de las di fi cultades para la rápida adopción de las tecnologías de la Web Semántica fue la di fi cultad en la creación y desarrollo de materiales. El dominio de XML, RDF, OWL y requiere un alto nivel de conocimientos técnicos y un compromiso no puede signi fi tiempo. Esta barrera para su uso se ha desacelerado la adopción de las tecnologías y también enmascarado gran parte del progreso en el desarrollo de sitios Web Semántica detrás de prototipos y aplicaciones de prueba de concepto. Afortunadamente, en los últimos años que ha cambiado.

En el último año o así, la atención se ha desplazado de las organizaciones individuales que proporcionan especificidad c sitios web habilitados semánticamente a los vendedores que envuelven las tecnologías asociadas a la Web Semántica en los sistemas llave en mano para la publicación de determinados tipos de información. Por ejemplo, AllegroGraph [3] proporciona un sistema de base de datos y lenguaje de consulta para la gestión de datos RDF, SPARQL y consultas utilizando los servicios de razonamiento en los datos, lo que libera a los desarrolladores potenciales de la necesidad de construir una comprensión profunda de esas tecnologías. Thetus [4] proporciona un sistema para hacer el modelado del conocimiento de toda la empresa utilizando la tecnología de Web Semántica. Con el aumento de los proveedores de herramientas de publicación y la autoría, la incidencia de las aplicaciones y los sitios web semánticas basadas en la Web seguirá aumentando.

12.6 La semántica para ICDE

El sistema ICDE se beneficiaría del uso de ontologías para apoyar el intercambio de información y tareas de integración de herramientas de terceros. Como insinuado en la introducción capítulo, una tarea dentro análisis transacción financiera que se beneficiaría enormemente de una descripción de la semántica sólido es la identificación de vocabularios consistentes. Se muestra en la Fig. 12.4 es una porción de una ontología financiera fi originalmente creado por Teknowledge [5] como parte de la ontología SUMO. El fragmento de la ontología muestra diferentes tipos de transacciones financieras fi dispuestos en una jerarquía abstracto.

Supongamos que esta ontología está disponible para el sistema ICDE, y un usuario ICDE estaba analizando el ejemplo presentado en la Fig. 12.2 . En esos datos, downtick es el

1 http://www.semantichacker.com/

2 http://www.ontoprise.de/de/en/home/products/semanticguide.html

3 http://www.agraph.franz.com/allegrograph/

4 http://www.thetus.com/

5 http://www.teknowledge.com/

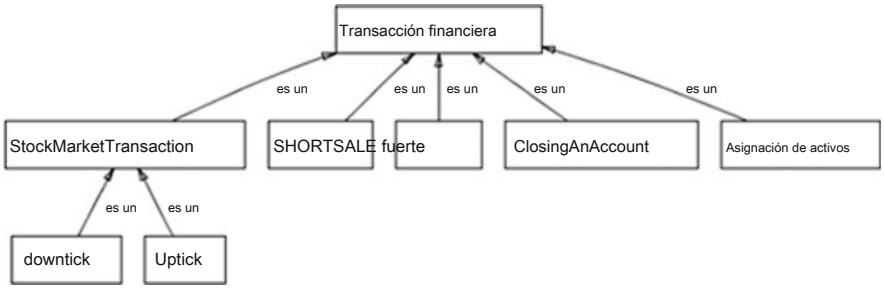


Fig. 12.4 Una ontología transacción financiera sencilla fi

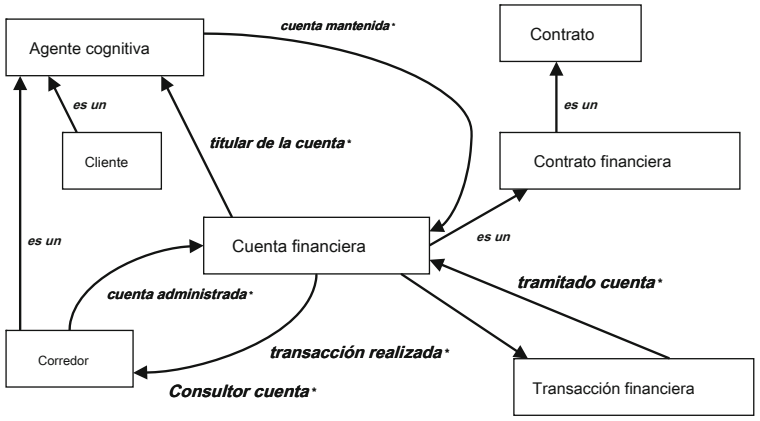


Fig. 12.5 Reglas en una ontología

etiqueta XML para el ID de transacción, una opción que podría evitar que otras herramientas de terceras partes del ICDE de hacer uso de los datos debido a que la etiqueta XML no es estándar. Sin embargo, el uso de la ontología y un motor de razonamiento, es sencillo determinar que downtick es un tipo de Transacción financiera y que la información debe ser compartida con cualquier herramienta que están interesados en los datos sobre las transacciones financieras.

Ontologías podrían proporcionar mucho más que servicios tesauro para herramientas ICDE. Una ontología OWL puede codificar reglas complejas sobre las relaciones entre los individuos de un tipo particular conceptual, lo que permitiría motores de razonamiento para hacer deducciones acerca de los elementos de datos individuales.

Considere el fragmento ontología se muestra en la Fig. 12.5 . Esto demuestra que la ontología contiene reglas que describen las relaciones entre cuentas, los titulares de cuentas, transacciones y corredores. Un motor de razonamiento puede usar estas descripciones para deducir relaciones entre un cliente particular y un corredor o para deducir que un corredor particular tenía una participación probable con una transacción individual, incluso cuando los datos que están siendo analizados no contenía ninguna vinculación fi específico entre las dos entidades.

Este tipo de ontología compartida podría permitir a colaborar herramientas de terceros ICDE partido para ayudar a la notificación de usuario previamente conexiones invisibles dentro de los datos. Por ejemplo, supongamos que una herramienta ayudó a seleccionar un usuario y analizar determinados tipos de transacciones financieras. Otra herramienta asistido al usuario a identificar las redes sociales de los individuos sobre la base de intereses comunes en las cuentas. Individualmente, ninguna de estas dos herramientas sería descubrir relaciones entre un asesor y un tipo particular de transacción, pero los resultados individuales de las dos herramientas se podrían combinar (posiblemente por un tercio de la herramienta) para descubrir las relaciones implícitas.

12.7 Servicios de Web Semántica

servicios web y arquitecturas orientadas a servicios fueron presentados en el capítulo anterior como un paso significativo hacia una solución sencilla para los problemas de interoperabilidad que normalmente afectan a las aplicaciones empresariales. Los servicios Web también juegan un papel en la Web Semántica. Dado que las aplicaciones de Web Semántica aumentan en complejidad, y como consumidores de información se vuelven más exigentes, el foco está pasando de la información semánticamente direccionable a los servicios semánticamente direccionables que permiten la creación de sistema de software personalizado, o servicios de la Web Semántica automatizados.

Las herramientas actuales proporcionar la capacidad para describir servicios Web, pero no tienen medios adecuados para la categorización y la utilización de esas descripciones. Las categorizaciones disponibles, tales como **WSIndex [6] y Ping de la Web Semántica, [7] están diseñados principalmente para uso humano en lugar de la máquina.** composición del sistema automatizado es el tema de prototipos de prueba de concepto en el momento, pero pocos sistemas operativos.

Sin embargo, los servicios web se construyen típicamente con descripciones de metadatos generosas, que es el componente clave de la Web Semántica. Al igual que con todos los metadatos, la di fi cultad en su uso para la composición dinámica radica en la comprensión de la semántica. Como era de esperar, una comunidad de investigación importante se centra en la aplicación de ontologías y tecnologías de Web Semántica para definir un dominio llamado los servicios de la Web Semántica. los servicios de la Web Semántica proporcionan un mecanismo para crear, localizar y utilizar semánticamente ricas descripciones de los servicios. Una de las tareas más importantes para esta comunidad es la estandarización de la descripción de la semántica asociada con descripciones de los servicios Web. Una vez que la semántica son claras, descripciones de servicios Web se pueden utilizar para crear las especificaciones para servicios compuestos, para representar la lógica de negocio a un nivel más abstracto,

Uno de los lenguajes subyacentes para la anotación semántica de los servicios Web es SAWSDL **(Anotaciones semánticas para Web Services Description Language). [8]**

---

6 http://www.wsindex.org

7 http://www.pingthesemanticweb.com/

8 http://www.w3.org/2002/ws/sawsdl/

SAWSDL no especifica la ontología sino que proporciona el lenguaje para identificar los conceptos ontológicos asociados a un servicio web dentro de la descripción del servicio. SAWSDL describe la definición de anotaciones para un pequeño subconjunto de los posibles componentes de una descripción WSDL. SAWSDL es ontología agnóstico, en que la especificación no hace referencia a un idioma preferido o codificación para ontologías.

Idiomas para la descripción de ontologías acerca de los servicios Web incluyen OWL-S, una variante fi co-servicio específico de búho y los Servicios Web Modelado Ontología (WSMO). Estos lenguajes permiten la integración de anotación semántica con la Web Services Description Language (WSDL). Integración con WSDL es importante ya que la mayoría de los servicios web existentes utilizan WSDL como base para la descripción del servicio. La creación de servicios de Web Semántica para el público en general, sin embargo parece bastante lejos por el momento. Existen buenas prototipos y la comunidad de la Web Semántica está llegando lentamente a un acuerdo acerca de las lenguas y de fi niciones necesarias para realizar servicios de Web Semántica.

Las tecnologías de la pena observar, sin embargo, ya que los éxitos en la construcción y utilización de servicios Web Semántica cambiará la forma en que se creó el software. El estado actual de los servicios de la Web Semántica se muestra prometedor para la integración de la empresa, pero en la actualidad carecen de la capacidad para el descubrimiento y composición de los servicios automatizados. No obstante, parece inevitable que los Servicios Web Semántica pronto definen un mecanismo automatizado para el hallazgo y los servicios que componen y cambiar la forma de pensar acerca de los sistemas de software.

12.8 El optimismo Continuación

La Web Semántica ha gozado de inmensa publicidad en los últimos años. Muchas descripciones de los proyectos de investigación se han ajustado rápidamente para reflejar incluso el más pequeño de conexión a la Web Semántica en un esfuerzo para tomar ventaja de que la popularidad. Por supuesto, esto se traduce en un aumento en el alcance de la investigación que dice ser la investigación de la Web Semántica, la reducción de la concentración de actividad para abordar los objetivos importantes de semánticamente ricos, los metadatos comprensible máquina para datos y procesos.

Mientras que muchos creen en las tecnologías, la cautela general parece prevalecer. En la superficie, la Web Semántica se parece a una refactorización de los proyectos de inteligencia arti fi ciales que pasaron de moda hace varios años. Sin embargo, la necesidad de que las representaciones semánticas de software y sistemas de información está ampliamente reconocida, y la demanda de soluciones reales está creciendo. Esta vez, los objetivos de investigación están más alineados con las necesidades del público, y la tecnología podría ganar aceptación.

La Web Semántica tiene todos los problemas de gestión de datos asociados con cualquier sistema de información general. ¿Quién tomará el tiempo para proporcionar todos los metadatos detallada sobre los servicios e información existentes? ¿Quién controla la información y los servicios de integridad, autenticidad y exactitud? ¿Cómo están las leyes y los problemas de privacidad abordados cuando se calcula se compone de servicios distribuidos? proveedores de servicios Web surgirán como una nueva categoría de negocio, pero ¿cómo van a

vigilar y regular? A medida que los sistemas se construyen que se basan en los metadatos de calidad, su mantenimiento y conservación se convertirán en cuestiones operacionales vitales.

A pesar del carácter prototípico de la mayoría de los sistemas operativos hasta el momento, la Web Semántica coloca nuevas técnicas, nuevas aplicaciones y experiencias importantes en la caja de herramientas de los arquitectos de software. La Web Semántica es simplemente un conglomerado de herramientas y tecnologías que cooperan, pero precisamente por la articulación flexible entre las tecnologías de la Web Semántica ofrece una caja de arena flexible para el desarrollo de nuevos marcos y arquitecturas.

Y, si uno mira más allá del bombo, los objetivos de la comunidad de la Web Semántica son los mismos que los objetivos de la arquitectura de software distribuido: crear débilmente acoplado,, e fi ciente de software fiable que responda a las necesidades de los usuarios. A través de los mecanismos de fi nidas formal de razonamiento para con los metadatos, la Web Semántica proporciona la base para la creación de software que es verdaderamente sensible a las necesidades de los usuarios, sus tareas y su contexto físico.

Los desarrolladores de software e investigadores están respondiendo rápidamente a las necesidades de la computación semántica. El Semantic Web Conference 2008 organizó un itinerario de investigación, una Web Semántica “en uso” de la pista, y talleres y tutoriales sobre todo, desde la seguridad de los sistemas de razonamiento. El tema es activa tanto en la industria y en el mundo académico. La arquitectura de servicios Web Semántica identificado mensaje es la mediación, la seguridad, la composición de procesos, negociación y contratación, y la formulación mensaje como aspectos importantes de la Web Semántica, y cada uno de éstos se está explorando y prototipos. Desarrollos como SAWSDL y ontologías servicio (OWL-S y WSMO) son prometedores como descripción del proceso y composición idiomas. La Web Semántica y la arquitectura de software sigan trayectorias que convergen rápidamente en un nuevo semánticamente impulsado, modo de construcción de software,.

12.9 Lectura adicional

Tres libros generales sobre la Web Semántica son:

Liyang Yu Introducción a la Web Semántica y la Web Semántica Servicios Chapman & Hall / CRC. 2007.

Pascal Hitzler, Sebastian Rudolph, Markus Kroetzsch, Fundamentos de la Semántica Tecnologías Web, Chapman & Hall / CRC. 2009.

Michael C. Daconta, Leo J. Obrst, Kevin T. Smith, La Web Semántica: Una guía para el futuro de XML, Servicios Web, y Gestión del Conocimiento, Wiley 2010.

Nigel Shadbolt, Wendy Hall, y de Tim Berners-Lee 2006 revisitación del la Ciencia artículo original **estadounidense La Web Semántica arroja luz sobre la visión de la gente en el frente de batalla y lo que creen** que se requiere para hacer realidad la promesa de la Web Semántica.

Shadbolt, N., Berners-Lee, T., y Hall, W. 2006. La Web Semántica Revisited. IEEE Intelligent Systems 21, 3 (May. 2006), 96-101. 182

David Provost ha revisado recientemente una serie de organizaciones de la industria de la Web Semántica y publicado su informe bajo Creative Commons License. Se titula

En la cúspide, un Examen Global de la Industria de la Web Semántica y está disponible en:

http://www.davidprovost.com/

El sitio Web del W3C es una fuente de gran información sobre la Web Semántica:

http://www.w3.org/2001/sw/

Especí fi cas detalles sobre algunas de las tecnologías se pueden encontrar en los siguientes lugares en línea:

BÚHO	http://www.w3.org/2007/OWL/wiki/OWL_Working_Group
SAWSDL	http://www.w3.org/2002/ws/sawsdl/
RDF	http://www.w3.org/RDF/
WSMO	http://www.cms-wg.sti2.org/home/
OWL-S	http://www.daml.org/services/owl-s/

Una herramienta para la construcción de ontologías se puede descargar libremente desde http: // www. protege.stanford.edu/ . Es una buena herramienta para explorar cómo las ontologías pueden ser construidos y utilizados:

capítulo 13
Arquitecturas Orientada a Aspectos

Yan Liu

13.1 Aspectos para el Desarrollo del ICDE

El entorno 2.0 ICDE tiene que cumplir con ciertos requisitos de rendimiento para las recuperaciones de datos API. Para tratar de garantizar este nivel de rendimiento, el comportamiento real de una aplicación ICDE debe ser monitoreado. La supervisión del rendimiento permite que las acciones correctivas a ser tomadas por el equipo de desarrollo si el nivel de rendimiento requerido no se cumple.

Sin embargo, v2.0 ICDE es un sistema grande, multihilo y distribuido, que comprende tanto off-the-shelf y componentes de encargo por escrito. Tales sistemas son culto fi notoriamente dif para controlar y aislar la causa raíz de los problemas de rendimiento, especialmente cuando se ejecuta en entornos de producción.

La estrategia tradicional para el control de rendimiento de las aplicaciones y la localización de los componentes causando cuellos de botella es para instrumentar el código de la aplicación con llamadas para registrar el tiempo y la utilización de recursos. Sin embargo, este enfoque lleva a duplicar el código que se inserta en diversos lugares de la fuente. Como siempre, código duplicado crea código de hinchazón, es propenso a errores y hace que sea más difícil mantener la aplicación como la aplicación ICDE evoluciona.

El ICDE teamwas consciente de los problemas de ingeniería de la inserción de código de monitorización del rendimiento a lo largo de la base de código ICDE. Por lo tanto, que buscaban una solución que pudiera separar el código de supervisión del rendimiento de la implementación de la aplicación de forma modular, más fácil de mantener. Aún mejor sería si fuera posible inyectar el código de supervisión del rendimiento en la aplicación sin la necesidad de volver a compilar el código fuente.

Por lo tanto, el equipo ICDE comenzó a mirar a los enfoques y tecnologías basadas en el aspecto para hacer frente a su problema de supervisión del rendimiento. programación de códigos de estructuras (AOP) orientada a aspectos en los módulos conocidos como aspectos. Los aspectos se fusionaron luego al compilar o en tiempo de ejecución para formar una solicitud completa.

El resto de este capítulo se proporciona una visión general de AOP, sus elementos esenciales y soporte de herramientas. También se analiza la influencia de los enfoques basados aspecto sobre arquitectura y diseño. Por último, el capítulo se describe cómo el sistema ICDE podría aprovechar las técnicas basadas en aspecto a supervisar el rendimiento de aplicaciones de una manera flexible, modular y fácil de mantener altamente fl.

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3_13, # Springer-Verlag Berlin Heidelberg 2011

13.2 Introducción a la Programación Orientada a Aspectos

programación orientada a aspectos (AOP) es un enfoque de diseño de software inventado en Xerox PARC en la década de 1990. [1] El objetivo de AOP es dejar que los diseñadores y desarrolladores separar mejor las "preocupaciones transversales" que un sistema de software debe abordar. preocupaciones transversales son elementos de comportamiento de un sistema que no se pueden localizar fácilmente a los componentes mercantiles en la arquitectura de una aplicación. preocupaciones transversales comunes son el manejo de errores, los controles de seguridad, registro de eventos y manejo de transacciones. Cada componente de la solicitud debe incluir típicamente código especí fi ca para cada preocupación transversal, por lo que el código del componente más complejo y más difícil de cambiar.

Para abordar las preocupaciones transversales, AOP proporciona mecanismos para la identificación sistemática, la separación, la representación y la composición. preocupaciones transversales están encapsulados en módulos separados, llamados "aspectos", por lo que la localización se puede lograr.

Springer-Verlag, (1997) 220-242. 186

AOP tiene una serie de beneficios potenciales. En primer lugar, ser capaz de identificar y representar de forma explícita las preocupaciones transversales ayuda a los arquitectos consideran el comportamiento transversal en términos de aspectos en una etapa temprana del ciclo de vida del proyecto. En segundo lugar permite a los desarrolladores reutilizar fácilmente el código de un aspecto en muchos componentes, y por lo tanto reduce el esfuerzo de utilizar (a menudo esto significa copiar) el código. En tercer lugar, AOP promueve una mejor modularidad y encapsulación como código de componente es sucinta y despejada.

Estructuración de las aplicaciones con los aspectos y la ejecución directa del diseño utilizando lenguajes de programación orientada a aspectos, tiene el potencial para mejorar la calidad de los sistemas de software. Aspectos pueden hacer posible que los sistemas de software grandes y complejos que se incluyen y recomponen en las ofertas simples y de mayor calidad. Para ver cómo funciona esto, veamos este enfoque en más detalles.

Orientada a Aspectos, Actas de la Conferencia Europea sobre programación orientada a objetos, vol. 1241.

13.2.1 Las preocupaciones transversales

Separación de intereses es un principio fundamental de la ingeniería de software. Este principio ayuda a gestionar la complejidad del desarrollo de software mediante la identificación, la encapsulación y la manipulación de aquellas partes del software correspondiente a una preocupación particular. Un "preocupación" es un requisito especí fi co o consideración que debe ser tratado con el fin de satisfacer el objetivo general del sistema.

---

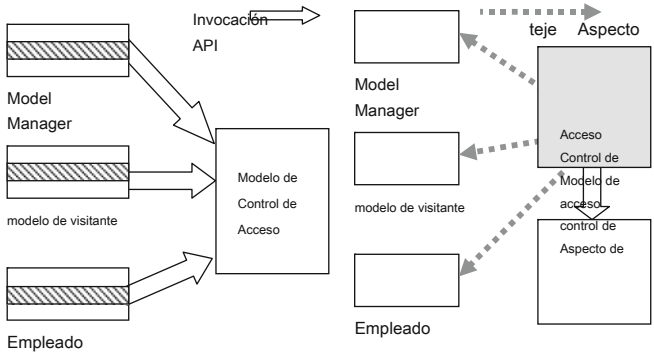
[1] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Videira Lopes, C., tier Loing, J.-M., y Irwin, J, Programación

Cualquier aplicación se compone de múltiples preocupaciones funcionales y no funcionales. preocupaciones funcionales son relevantes para el uso real de la aplicación, mientras que las preocupaciones no funcionales se refieren a los atributos de calidad globales del sistema, tales como el rendimiento, las transacciones y la seguridad. Incluso las aplicaciones que están diseñadas de una manera altamente modular sufren de enredos de los aspectos funcionales y no funcionales. Por ejemplo, la lógica de caché para mejorar el rendimiento de base de datos podría estar integrada en la lógica de negocio de muchos componentes diferentes, mezclando así que se enreden o preocupaciones funcionales y de rendimiento. Otros ejemplos de preocupaciones transversales incluyen la supervisión del rendimiento, control de transacciones, autorización de servicio, el control de errores, el registro y la depuración. El manejo de estas preocupaciones se extiende a través de múltiples módulos de aplicación,

13.2.2 Las preocupaciones con Gestión de los Aspectos

El uso de técnicas de diseño convencionales, una preocupación transversal puede ser en módulos mediante una interfaz para encapsular la aplicación de la preocupación de sus componentes de cliente que invocan. Aunque la interfaz reduce el acoplamiento entre los clientes y la aplicación de la preocupación, los clientes todavía tienen que integrar código para llamar a los métodos de interfaz desde dentro de su lógica de negocio. Esto contamina la lógica de negocio.

Con un diseño y programación orientada a aspectos, cada una preocupación transversal se lleva a cabo por separado en un componente conocido como un aspecto. En la Fig. 13.1 , La diferencia entre la aplicación de una preocupación de registro utilizando la programación convencional y AOP se demuestra. El aspecto de fi ne puntos de ejecución de los componentes del cliente que requieren la aplicación de la preocupación transversal. Para cada



1. (a) modelo convencional *2. (b) Modelo AOP*

Fig. 13.1 Implementación de una preocupación tala

punto de ejecución, el aspecto y luego define el comportamiento necesarios para implementar el comportamiento aspecto, como llamar a una API de registro.

Es importante destacar que los módulos cliente ya no contienen ningún código para invocar la aplicación aspecto. Esto lleva a los componentes del cliente que no están contaminados por las llamadas a implementar una o más preocupaciones.

Una vez de fi nido, el uso de un aspecto es especificados en las reglas de composición. Estas reglas de composición son introducidos en una utilidad de programación conocido como un “tejedor”. Un tejedor transforma el código de la aplicación, componiendo el aspecto con sus clientes que invocan. lenguajes de programación orientada a aspectos tales como AspectJ proporcionan herramientas de tejer, y por lo tanto son necesarios para aplicar eficazmente diseños orientadas a aspectos lenguajes y herramientas de AOP.

13.2.3 Sintaxis AOP y Programación Modelo

“Transversal” es una técnica de AOP para permitir la identificación fi de las preocupaciones y la estructuración de ellos en módulos de manera que pueden ser invocados en diferentes puntos a lo largo de una aplicación. Existen dos variedades de corte transversal, a saber, estáticas y dinámicas. dinámica transversal modi fi ca el comportamiento de ejecución de un objeto tejiendo en un nuevo comportamiento en puntos especí fi cos de interés. transversal estático altera la estructura estática de un componente mediante la inyección de métodos adicionales y / o atributos en tiempo de compilación. Las construcciones básicas del lenguaje y la sintaxis utilizados para definir transversal en AOP son:

- Un “punto de unión” es una identi fi punto de ejecución capaz en una aplicación, tal como una llamar a un método o una asignación a una variable. Unirse puntos son importantes, ya que son los comportamientos de aspecto donde se tejen en la aplicación.
- Un “punto de corte” identifica las unirse a un punto en el programa en el que un eje transversal preocupación debe ser aplicada. Por ejemplo, la siguiente define un punto de corte cuando el valor ajustado método de la Valores clase se llama:

```
de registro de punto de corte (String msg): args (msg)
ejecución (void Stock.setValue (float))
```

- Un “consejo” es una pieza de código que implementa la lógica de una preocupación transversal. Se ejecuta cuando se alcanza un especí pointcut fi ed.
- Una “introducción” es una instrucción transversal que puede hacer cambios a estáticas los componentes de aplicación. Una introducción puede ser, por ejemplo, añadir un método a una clase en la aplicación.
- Un aspecto en el AOP es equivalente a una clase en la programación orientada a objetos. Eso encapsula puntos de corte y asesoramiento asociado e introducciones.

En la Fig. 13.2 se ilustra la relación entre estos términos AOP. 188

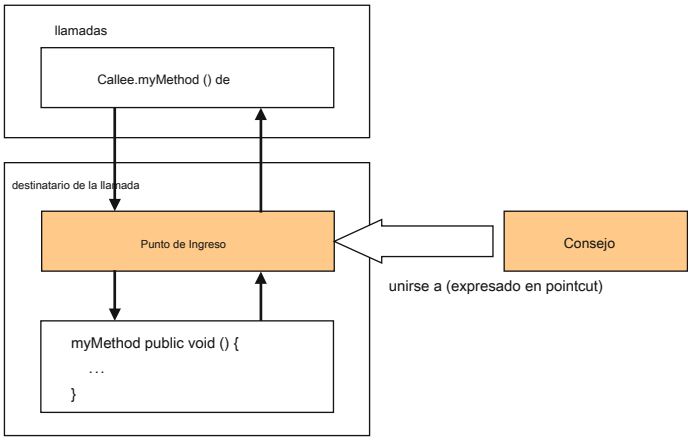


Fig. 13.2 La anatomía de AOP

13.2.4 Weaving

Al darse cuenta de un diseño orientado a aspectos requiere apoyo lenguaje de programación para poner en práctica los aspectos individuales. El lenguaje también de fi ne las reglas para la aplicación de tejer un aspecto con el resto del código de la aplicación. Tejer puede seguir una serie de estrategias, a saber:

1. Una fuente especial de código de preprocesador ejecutada durante la compilación
2. Un postprocesador que los parches de archivos binarios
compilador 3. Un AOP-consciente de que genera tejida binario fi les
4. Cargar en tiempo tejer (LTW); por ejemplo, en el caso de Java, tejiendo el asesoramiento pertinente mediante la carga de cada clase de consejo en la JVM
5. tiempo de ejecución de tejer (RTW); interceptando cada unión punto en el tiempo de ejecución y la ejecución de todos los consejos pertinentes. Esto también se conoce como “hotswapping” después de la clase es cargada

La mayoría de los lenguajes AOP soporte en tiempo de compilación tejer (CTW) usando uno de los primeros tres opciones. En el caso de Java, la forma en que normalmente funciona es que el compilador de Java genera la clase estándar binario archivos, que cualquier JVM estándar puede ejecutar. Entonces el . clase archivos son modi fi cado en base a los aspectos que se han de fi nido. CTW no es siempre la mejor opción, sin embargo, ya veces simplemente no es posible (por ejemplo, con Java Server Pages).

LTW ofrece una mejor solución con mayor flexibilidad. En el caso de Java, LTW requiere que el cargador de clases JVM para poder transformar o clases de instrumentos en tiempo de ejecución. el JDK [2] v5.0 admite esta función a través de un mecanismo estándar simple. LTW debe procesar el código de bytes de Java en tiempo de ejecución y crear estructuras de datos (esto puede

[2] Kit de desarrollo de Java.

ser lento) que representan el código de bytes de una clase particular. Una vez que todas las clases se cargan, LTW no tiene ningún efecto sobre la velocidad de la ejecución de la aplicación. AspectJ, [3]

JBoss AOP [4] y AspectWerkz [5] Ahora apoyar LWT.

RTW es una buena opción si aspectos deben estar habilitadas en tiempo de ejecución. Sin embargo, al igual que LTW, RTW puede tener inconvenientes en términos de rendimiento en tiempo de ejecución, mientras que los aspectos se tejieron en.

13.3 Ejemplo de un aspecto de la caché

En esta sección vamos a utilizar un ejemplo sencillo para ilustrar el modelo de programación de AOP. [6] Esta sencilla aplicación calcula el cuadrado de un entero dado. Con el fin de mejorar el rendimiento, si un valor de entrada particular, se ha encontrado antes, su valor cuadrado se recupera de una memoria caché. La memoria caché es una preocupación transversal, no es una parte esencial de calcular el cuadrado de un número entero.

El ejemplo se implementa utilizando AspectJ y se muestra en la Fig. 13.3 . El caché es implementado como un aspecto en Cache.aj y separada de la implementación de la aplicación de núcleo, Application.java. El método calculateSquare es un punto de unión y es identi fi cado por el punto de corte calcular en el Cache aspecto, como en el siguiente:

```
Calcular el punto de corte (int i): args (i)
    && (ejecución (int Application.calculateSqure (int)));
```

La implementación de la función de caché, recuperando un valor de una java.util. Tabla de picadillo, se proporciona dentro de la alrededor Consejo. Tenga en cuenta que este consejo se aplica sólo a la clase Solicitud. El aspecto caché se teje en el código de la aplicación en tiempo de compilación usando un compilador AspectJ.

La siguiente salida de la ejecución del programa demuestra el consejo se invoca en el punto de unión.

```
aspecto de la caché se invoca para el parámetro 45 El cuadrado de
45 es 2,025
aspecto de la caché se invoca para el parámetro 64 El cuadrado de
64 es 4,096
aspecto de la caché se invoca para el parámetro 45 El cuadrado de
45 es 2,025
aspecto de la caché se invoca para el parámetro 64 El cuadrado de
64 es 4,096
```

3 http://www.eclipse.org/aspectj/

4 http://www.jboss.org/products/aop

5 http://www.aspectwerkz.codehaus.org/

6 Chapman, M., Hawkins, H. Las aplicaciones Java orientadas a aspectos con Eclipse y AJDT, IBM developerWorks, http://www-128.ibm.com/d

```
// El código fuente de Application.java
El almacenamiento en caché el paquete;

Aplicación public class {
    principales (args String []) {public static void
       System.out.println ( "El cuadrado de 45 es" + calculateSquare (45)); System.out.println ( "El cuadrado de 64
      es" + calculateSquare (64)); System.out.println ( "El cuadrado de 45 es" + calculateSquare (45));
      System.out.println ( "El cuadrado de 64 es" + calculateSquare (64)); } Private static int calculateSquare (int
      numero) {

       tratar {
           Thread.sleep (6000);
       } Catch (es decir InterruptedException) {} número de
    devolución * número; }}



// El código fuente de Cache.aj
El almacenamiento en caché el paquete;
java.util.Hashtable importación; Caché

aspecto público {
      valueCache Hashtable privado; Calcular el punto de corte (int
      i): args (i)
                && (ejecución (int Application.calculateSquare (int))); int alrededor (int i): el cálculo de (i) {

          System.out.println ( "Caché aspecto se invoca para el parámetro" + i); si (valueCache.containsKey (new
          Integer (i))) {
              retorno ((entero) valueCache.get (new Integer (i))) intValue ().; } Cuadrado int = proceder (i);


          valueCache.put (new Integer (i), new Integer (cuadrado)); volver cuadrado; } Caché
        pública () {

          valueCache = new Hashtable (); }}
```

Fig. 13.3 Un aspecto caché implementada utilizando AspectJ

13.4 Arquitecturas Orientada a Aspectos

Un aspecto relacionado con la calidad de un sistema atribuye en gran medida influye en la arquitectura de la aplicación, y muchos de estos aspectos son básicamente imposible de localizar. Por ejemplo, para garantizar el rendimiento de una aplicación débilmente acoplados, la consideración se debe pagar al comportamiento de los componentes individuales y sus interacciones con los otros. Por lo tanto, las preocupaciones tales como el rendimiento tienden a cortar transversalmente la arquitectura del sistema a nivel de diseño, y que no pueden ser simplemente capturados en un solo módulo.

AOP proporciona una solución para el desarrollo de sistemas mediante la separación de las preocupaciones transversales en módulos y sin apretar el acoplamiento de estas preocupaciones a requisitos funcionales. Además, se han propuesto las disciplinas de diseño como el diseño orientado a aspectos (AOD) y el desarrollo de software orientado a aspectos (AOSD) para extender los conceptos de AOP a etapas anteriores del ciclo de vida del software. Con AOD y AOSD, la separación de las preocupaciones se dirige en dos niveles diferentes.

En primer lugar a nivel de diseño, tiene que haber una clara identificación y definición de la estructura de los componentes, aspectos, puntos de unión y su relación. Aspecto de diseño y modelado son las principales actividades de diseño en este nivel. preocupaciones individuales tienden a estar relacionada con múltiples artefactos arquitectónicos.

Por ejemplo, una preocupación para el rendimiento puede estar asociado con un conjunto de casos de uso en los requisitos de arquitectura, un número de componentes en el diseño y algunos algoritmos para ef fi cientemente ejecución componentes lógicos específicos. Los requisitos para cada aspecto necesitan ser extraído de la declaración del problema original y la arquitectura tiene que incorporar aquellos aspectos e identificar su relación con otros componentes. También es importante identificar posibles con fl ictos que surgen cuando los aspectos y componentes se combinan en este nivel. Para ser eficaz, este enfoque requiere que ambas metodologías de diseño y soporte de herramientas para modelar aspectos.

En segundo lugar, a nivel de aplicación, estos aspectos arquitectónicos se deben asignar a una aplicación de aspecto y teje en la ejecución de otros componentes. Esto requiere no sólo la expresividad de un lenguaje de AOP que puede proporcionar la semántica para implementar puntos de unión, sino también una herramienta de tejido que puede interpretar las reglas de tejido y combinar las implementaciones de aspectos.

distribuidos, IEEE Computer Society, (2003), 14 (11): 1058 - 1073. 192

13,5 aspectos arquitectónicos y Middleware

Como se explica en el capítulo. 4, las tecnologías de middleware basadas en componentes tales como JEE proporcionan servicios que apoyan, por ejemplo, procesamiento distribuido transacción, seguridad, servicios de directorio, servicios de integración, la agrupación de conexiones de base de datos, y así sucesivamente. Los diferentes temas tratados por estos servicios son también las preocupaciones no funcionales primarias seleccionadas por AOSD. En este caso, tanto la tecnología de componentes y AOP abordan el mismo tema de la separación de las preocupaciones.

No es sorprendente entonces, middleware es uno de los dominios más importantes para la aplicación de AOP. La investigación sobre la minería aspecto [7] muestra que el 50% de las clases en tres implementaciones CORBA ORB son responsables de la coordinación con un aspecto particular. AOP se ha utilizado en tales casos refactorizar efectivamente un ORB CORBA y modularizar su funcionalidad.

A raíz de estos esfuerzos, se han hecho intentos para introducir AOP para encapsular los servicios de middleware para construir altamente con fi gurable arquitecturas de middleware. Distribución, de persistencia y de transacción aspectos para los componentes de software que utilizan AspectJ se han implementado con éxito, y se extiende AspectJEE AspectJ para implementar el modelo EJB y varios servicios JEE. En el mundo de los productos de código abierto, JBoss AOP ofrece una amplia biblioteca de aspecto para el desarrollo de la aplicación basada en Java usando técnicas de AOP.

[7] Zhang, C., Jacobsen, H. Refactorización middleware con aspectos. En IEEE Transactions on sistemas paralelos y

El principal problema en la aplicación de AOP para construir marcos de middleware es que los servicios de middleware no son generalmente ortogonal. Colocación de un servicio (de aspecto) a un componente sin entender su interacción con otros servicios no es sensible, como los efectos de los servicios pueden interactuar unos con otros.

Por ejemplo, aspectos se utilizan comúnmente para tejer comportamiento transaccional con código de aplicación. transacciones de base de datos se pueden cometer utilizando ya sea una fase o de dos fases (para transacciones distribuidas) cometer protocolos. Para cualquier transacción individual, se ejecuta sólo un protocolo, y por lo tanto sólo un aspecto, y de infinitamente no ambos, debe ser tejida para cualquier punto de unión. En general, el manejo de los aspectos que interactúan es un problema di fi culto. Ya sea un error en tiempo de compilación o una excepción de ejecución debe aumentarse si los dos aspectos que interactúan comparten un punto de unión.

13.6 Estado-of-the-Art

Los recientes esfuerzos de investigación y desarrollo se han dedicado a diferentes tecnologías y prácticas orientadas por aspectos. Estos incluyen el lenguaje AOP especi fi cación, soporte de herramientas para el modelado de aspecto y la generación de código, y la integración con las tecnologías emergentes, tales como la programación basada en metadatos. Vamos a discutir cada uno de estos.

13.6.1 orientada a aspectos de modelado en UML

Existen varios enfoques para apoyar el modelado de aspecto de AOD y AOSD. La mayoría de estos enfoques se extienden UML por de fi nir un nuevo UML per fi l de AOSD. Esto permite que las extensiones de UML con los conceptos de aspecto a ser integrados en las herramientas CASE existentes que apoyan la norma UML.

Una ventaja de modelado orientado aspecto es el potencial para generar código para aspectos de los modelos de diseño. En el modelado orientado aspecto y la generación de código, el código de aspecto y el código nonaspect se genera por separado. El uso de Model Driven Architecture (MDA) se acerca, herramientas utilizan una transformación definición para transformar un modelo independiente de la plataforma (PIM) en uno o más modelos de plataforma especí fi cos (PSM), a partir del cual puede tener lugar la generación automática de código de aspecto y el tejido. tecnologías de MDA se explican en detalle en el siguiente capítulo.

Herramientas 13.6.2 AOP

El modelo fundamental de AOP es el modelo de punto de unión. Todas las herramientas de AOP emplean este modelo para proporcionar un medio de identificar donde se aplican las preocupaciones transversales.

Sin embargo, diferentes herramientas implementan la especificación del modelo cs aspecto a su manera, e introducen nuevos mecanismos para la semántica y los aspectos de tejido.

Por ejemplo, en JBoss AOP, consejos se implementan a través de “interceptores” utilizando Java reflexión y puntos de corte se definen en un archivo XML fi l que describe el lugar para tejer en un consejo de forma dinámica en tiempo de ejecución. En AspectJ, ambos consejos y puntos de corte se de fi nen en una clase de aspecto y tejió estáticamente.

Esta diversidad en las herramientas de AOP es un problema para el desarrollo de software utilizando aspectos, debido a las diferencias semánticas de los diferentes modelos de AOP y las diferentes maneras en un aspecto se teje con otras clases. No es posible simplemente volver a desarrollar un aspecto existente con el fin de que sea tejida con otros aspectos desarrollados con otro modelo AOP.

Para hacer frente a este problema, AspectWerkz [8] utiliza el código de bytes modi fi cación para tejer clases de Java en el proyecto de acumulación de tiempo, el tiempo de carga de clase o en tiempo de ejecución. Se engancha en el uso de las API de nivel JVM estándar, y tiene un poderoso modelo se unen a punto. Aspectos, consejos e introducciones están escritos en Java y llano clases de objetivos pueden ser POJOs regulares. Aspectos pueden definirse ya sea utilizando Java 5 anotaciones, Java 1.3 / 1.4 doclets personalizados o un sencillo XML definición fi l. (En cierto estilo orientada a aspectos, AspectWerkz se teje en la liberación v5.0 AspectJ en 2006).

13.6.3 Las anotaciones y AOP

El modelo de punto de unión puede utilizar las propiedades de los elementos de programa tales como firmas de métodos para capturar unirse puntos. Sin embargo, no puede unirse a la captura de puntos necesarios para poner en práctica ciertas preocupaciones transversales, como la transacción y la seguridad basada en roles, ya que no hay información en el nombre o la firma de un elemento que sugiera la necesidad de comportamientos transaccionales o autorizaciones relacionadas. Adición de metadatos a los sistemas de AOP tanto, es necesario proporcionar una solución para estos casos.

En el contexto de lenguaje de programación, los metadatos conoce como “anotaciones”, la captura de información adicional asociado a los elementos tales como los métodos, campos, clases y paquetes de programas. La v5.0 JSE y el C # / VB .NET idiomas proporcionan estándares del lenguaje para adjuntar anotaciones a los elementos del programa. Un buen ejemplo de anotaciones que aplican está declarando transacciones en los marcos JEE y .NET. Por ejemplo, la siguiente anotación declara el atributo de transacción del método actualizar() en EJB 3.0:

```
@TransactionAttribute
        (TransactionAttributeType.REQUIRED) actualización public
  void (doble nuevovalor)
        throws Exception
```

[8] http://www.aspectwerkz.codehaus.org/

13.7 Supervisión del rendimiento del ICDE con AspectWerkz

Cuando se ejecuta en la producción, es deseable poder inyectar código de monitorización del rendimiento en componentes ICDE sin recompilar la aplicación completa. Usando aspectos, esto se puede lograr usando LTW. Por lo tanto el equipo ICDE comienza a diseñar una arquitectura basada en aspecto usando AspectWerkz como se muestra en la Fig. 13.4 . En esta arquitectura, la instrumentación de rendimiento para diferentes componentes ICDE se encapsula en un aspecto dedicado que se puede inyectar en la aplicación ICDE. Esto es necesario porque los indicadores que deben ser grabados son de naturaleza diferente. Por ejemplo, la supervisión del rendimiento de un servidor JMS medidas tanto la tasa de procesamiento de mensajes y el tamaño del mensaje, mientras que la instrumentación de las sentencias SQL mide el tiempo de respuesta.

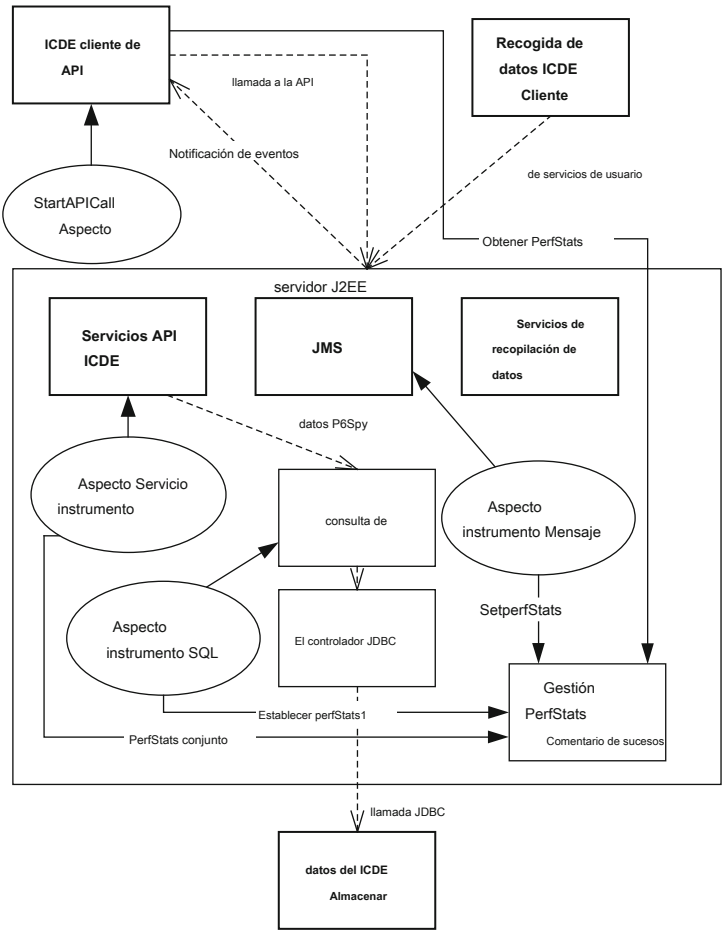


Fig. 13.4 arquitectura ICDE 2,0 a base de aspecto para la supervisión del rendimiento ICDE

Con el fin de instrumentar el tiempo de respuesta de la consulta de base de datos, un componente de código abierto, P6Spy, [9] se utiliza. Esto actúa como una capa entre el conjunto de conexiones JEE y los controladores JDBC, la captura de las sentencias SQL emitidas por la aplicación JEE. Un aspecto también se debe aplicar a este componente para recuperar la información de instrucción SQL.

Una vez que se captura todos los datos de rendimiento, hay una variedad de opciones para que esté disponible para su procesamiento posterior. Se puede simplemente escribe en un registro de fi le periódicamente o cargado en una base de datos. Una solución flexible fi ciente y e fl para proporcionar acceso directo a los datos de rendimiento del sistema en vivo es utilizar un protocolo estándar como Java Gestión de extensión (JMX) [10] que las herramientas de gestión existentes JEE pueden visualizar y realizar un seguimiento.

```
InstrumentSQLAspect clase pública
{LogJdbcQueries objeto público (finales puntos de intersección de puntos de inflexión)

                        lanza Throwable
{ // Información de Acceso Tipo de tiempo de ejecución

    MethodRtti rtti = (MethodRtti) joinPoint.getRtti (); Cadena de consulta = (String)
    rtti.getParameterValues () [0]; Larga horaInicio = System.currentTimeMillis ();

    // ejecutar el método
    resultado final Object = joinPoint.proceed (); Larga endTime =
    System.currentTimeMillis ();
    // registrar la información de temporización para esta ejecución de instrucciones SQL
    perfStatsManager.log (consulta "Estado", endTime-horaInicio); return resultado; }


logValuesInPreparedStatement public Object (final de puntos de intersección de puntos de inflexión) throws
Throwable
{MethodRtti rtti = (MethodRtti) joinPoint.getRtti ();

    índice entero = (entero) rtti.getParameterValues () [0]; Valor de objeto = rtti.getParamterValues () [1];
    consulta String = “index =” + index.intValue () + “value =”

                                    +   value.toString ();
    Larga horaInicio = System.currentTimeMillis ();
    // ejecutar el método
    Resultado final Object = joinPoint.proceed (); Larga endTime =
    System.currentTimeMillis ();
    // registrar la información de temporización para este PreparedStatement // ejecución

    perfStatsManager.log (consulta, “PreparedStatement”, endTime- horaInicio); return resultado; }};
```

Fig. 13.5 sentencia SQL aplicación aspecto de instrumentación

9 http://www.p6spy.com/

10 http://www.java.sun.com/products/JavaManagement/

```
<Aspectwerkz>
    <Sistema id = "ICDE">
      <Paquete name = "com.icde.perf.aop"> <clase aspecto =
        "InstrumentSQLAspect"
              implementación de modelo = "perThread"> <pointcut name
          = "Estado" expresión =
              "Ejecución (* java.sql.Connection + .prepare * (..))" /> <pointcut name = expresión
          "PreparedStatement" =
              "Ejecución (java.sql.PreparedStatement vacío + .set * (..))" /> <asesoramiento name =
          "logJdbcQueries (final de puntos de intersección de puntos de inflexión)"
              type = "alrededor de" bind-a = "Estado" />
              <Nombre de asesoramiento = "logValuesInPreparedStatement (puntos de inflexión definitivo
              joinpoint)" type = "alrededor de" bind-a = "PreparedStatement"/> </ aspecto> </ package> </
    sistema> </ Aspectwerkz>
```

Fig. 13.6 InstrumentSQLAspect XML definición fi l

Para ilustrar el diseño, la implementación y el despliegue de los aspectos AspectWerkz, vamos a describir en detalle el InstrumentSQLAspect. Para medir los tiempos de respuesta instrucción SQL, tenemos que localizar todas las llamadas a métodos donde un java.sql.Statement se crea y se inyecta código de tiempo inmediatamente antes y después de ejecutar la consulta SQL. También tenemos que rastrear todas las llamadas a métodos que un valor se encuentra en una java.sql.PreparedStatement ejemplo. El fragmento de código resultante para el InstrumentSQLAspect se ilustra en la Fig. 13.5 .

El siguiente paso es compilar los aspectos como una clase Java normal con las bibliotecas AspectWerkz. Las reglas de tejido para unir el asesoramiento al punto de corte se especi fi ca en el aop.xml fi le como se muestra en la Fig. 13.6 . [11]

LTW para AspectWerkz se logra mediante la carga de la biblioteca AspectWerkz para la v5 JDK. La aplicación ICDE entonces se puede arrancar normalmente y el código de aspecto se tejió en al tiempo de carga

En resumen, el uso de técnicas de AOP, código de instrumentación puede ser separado y aislado en aspectos. La ejecución de los aspectos puede ser tejida en el sistema en tiempo de ejecución y sin la necesidad de volver a compilar todo el sistema.

13.8 Conclusiones

AOP se introdujo originalmente como un mecanismo de programación para encapsular funcionalidad entrecruzada. Su éxito ha visto técnicas orientadas a aspectos se acostumbran en varios dominios de aplicación, tales como marcos de middleware. También tiene

[11] Tenga en cuenta que como contenedores JEE son multi-hilo, y las solicitudes individuales son manejados por hilos mantenidas en un grupo de subprocesos, el aspecto se despliega en perThread modo.

dado lugar a técnicas de modelado y diseño que influyen en la arquitectura de un sistema de software construido usando técnicas orientadas a aspectos.

AOP trae tanto oportunidades como desafíos para la arquitectura de software. En los dominios limitados, AOP ha demostrado una gran promesa en la reducción de la complejidad del software a través de proporcionar una separación clara y modularización de preocupaciones. áreas fructíferas incluyen una mayor integración de AOP y middleware para aumentar la flexibilidad de las plataformas de middleware con fi gurar. Incluso en este ejemplo, sin embargo, los problemas siguen siendo difíciles, es decir, la coordinación de múltiples aspectos para hacer frente a los conflictos, como las preocupaciones transversales no son completamente ortogonales.

orientada a aspectos de diseño e implementación requiere el apoyo de herramientas e fi ciente de AOP. Con este tipo de herramientas, en curso de investigación y desarrollo sigue intentando ofrecer mejores soluciones en diversas áreas, a saber:

- **Mantenimiento: El diseño de sistemas orientados a aspectos de calidad significa prestar atención**

  para de fi nir pointcuts robustos y de manera prudente mediante la herencia aspecto. Pointcuts que la captura se unen a más puntos de los esperados o pasar por alto algunos puntos de combinación que desee puede conducir a implementaciones frágiles como el sistema evoluciona. En consecuencia se necesita una herramienta deficiente depuración ef para detectar el punto defectuoso y la implementación punto de corte se unen.

- **Actuación: El uso de AOP introduce los gastos generales de funcionamiento adicionales en aplica-**

  ciones, tanto durante el proceso de tejido y, potencialmente, en tiempo de ejecución. La sobrecarga de AOP debe minimizarse para proporcionar una buena acumulación y el rendimiento en tiempo de ejecución.

- **Integración: La reutilización de los aspectos que no se ha explorado su fi cientemente, de manera que**

  Los diseñadores pueden utilizar las bibliotecas de los aspectos en lugar de desarrollar cada aspecto de cero. Como cada herramienta AOP sólo proporciona implementaciones de aspecto específica para su propio modelo de AOP, un aspecto implementado por un modelo AOP no puede ser fácilmente teje en un sistema con aspectos utilizando un modelo de AOP diferente. Esto es potencialmente un obstáculo serio para la adopción de aspecto orientación en una amplia gama de aplicaciones de software.

y la primavera, en cuanto a sus mecanismos de lenguaje y entornos de desarrollo es: 198

En resumen, las técnicas orientadas a aspectos se desarrollan y maduran, y demostrando ser útiles en varios dominios de aplicación y la herramienta. Estos incluyen la seguridad, la explotación forestal, la vigilancia, las transacciones y almacenamiento en caché. Ya sea aspecto orientación se convertirá en un importante paradigma de diseño y desarrollo es mucho más abierto al debate. Sin embargo, parece inevitable basado en la adopción corriente que seguirán siendo infundida gradualmente en la corriente principal de la ingeniería de software técnicas orientadas a aspectos.

13.9 Lectura adicional

Una buena comparación de cuatro herramientas de Java AOP, a saber AspectJ, AspectWerkz, JBoss AOP AOP

M. Kersten, AOP herramientas de comparación. IBM developerWorks, http://www-128.ibm.
com / developerworks / library / j-aopwork1 /

Una fuente de información de amplio alcance sobre aspectos se mantiene en el wiki AOSD en:

http://www.aosd.net/wiki/index.php?title ¼ Pagina principal

El (en desuso) página de inicio para Aspectwerkz es

http://www.aspectwerkz.codehaus.org/

AspectJ documentación se puede encontrar en:

http://www.eclipse.org/aspectj/docs.php

Los buenos guías prácticas a AspectJ y aspectos de las aplicaciones de base de datos son:

Ramnivas laddad, Aspectj en Acción: Empresa AOP con aplicaciones de primavera,
Manning Publications, 2009. Awais Rashid, Sistemas de base de datos orientada a aspectos, Springer-Verlag,
2009.

capítulo 14
Arquitectura basada en modelos

Zhu Liming

14.1 Desarrollo Model-Driven para ICDE

Uno de los problemas que acechan en la parte posterior de la mente del equipo de desarrollo del ICDE está relacionado con la planificación de la capacidad para las nuevas instalaciones del ICDE. Cuando una instalación ICDE soporta múltiples usuarios, la carga de solicitudes se convertirá en alto, y el hardware que se ejecuta en la plataforma debe ser lo suficientemente potente como para soportar esta carga de solicitudes. Si el hardware se satura, no será capaz de procesar todos los eventos generados por el usuario, y los datos importantes se pueden perder. La situación se ve agravada por las siguientes cuestiones:

- Diferentes dominios de aplicación y diferentes instalaciones individuales dentro de cada dominio usará ICDE de diferentes maneras, y por lo tanto generar diferentes cargas solicitud por usuario.
- Diferentes instalaciones desplegarán ICDE en diferentes plataformas de hardware, cada capaz de soportar un número diferente de usuarios.
- La plataforma ICDE será portado a diferentes servidores de aplicaciones JEE, y cada una de ellas tiene diferentes características de rendimiento.

Todos estos problemas están relacionados con la actividad de ingeniería de software de planificación de capacidad. La planificación de capacidad se refiere a qué tan grande, en términos de recursos de hardware y software, la instalación deberá cumplir en apoyo de su petición de carga esperada. técnicas de modelado matemático a veces pueden ser utilizados para predecir la capacidad de una plataforma para componentes y redes estandarizadas. [1] Sin embargo, más típicamente, las pruebas de referencia se ejecutan en un prototipo o una aplicación completa para probar y medir cómo realiza el despliegue de hardware / software combinado.

La única manera realista el equipo ICDE podría anticipar para llevar a cabo la planificación de capacidad era para ejecutar una carga de prueba en las plataformas de despliegue fi cas. Para cada instalación, el equipo tendría que:

[1] Por ejemplo, el Gestor de capacidad de Microsoft y su apoyo a las implementaciones de Exchange.

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3_14, # Springer-Verlag Berlin Heidelberg 2011

- Instalar ICDE en la plataforma de hardware de destino, o uno que sea lo más cerca posible en especí fi cación a la plataforma de despliegue esperado.
- Desarrollar las solicitudes de prueba de muestra generados por el software robots para generar una carga en la plataforma, y medir cómo responde. Las solicitudes de prueba deben reflejar el uso esperado per fi l de los usuarios que operan en esa instalación ICDE.

Así, para cada instalación, se debe desarrollar un conjunto de pruebas, cada una de las cuales se ejecutará una serie de peticiones en la plataforma ICDE y medir el tiempo de respuesta y el rendimiento. Esto se muestra en la figura. 14.1 .

Como era de esperar, la ICDE teamwere muy interesado en hacer todo este ejercicio de planificación de calidad de e fi ciente y sin dolor como sea posible. Esto significaría reducir al mínimo la cantidad de desarrollo de sitio-específico. Así, por ejemplo, en lugar de escribir un robot de prueba específico para cada instalación, que les gustaría para definir los datos de carga de prueba y prueba externa al código, y de alguna manera esta entrada en el robot de interpretar. Ellos también como los resultados de rendimiento de ejecuciones de prueba a ser producidos y clasifica automáticamente en forma de gráficos para facilitar el análisis.

Para lograr esto, el equipo decidió explotar métodos arquitectura dirigida por modelos y tecnologías de desarrollo de apoyo. enfoques modelo impulsado animan a los componentes de un sistema de software que se describen en los modelos UML. Estos modelos son entonces de entrada en los generadores de código que producen automáticamente código ejecutable que corresponde al modelo. El equipo esperaba que pudieran desarrollar un modelo único de un robot de prueba ICDE. Entonces, simplemente cambiando los parámetros en el modelo, que podrían generar una prueba de carga fi c instalación especí en la prensa de un botón.

En este capítulo se describen los elementos esenciales de los enfoques basados en modelos de arquitectura. A continuación, muestra cómo el equipo ICDE podría utilizar técnicas dirigidas por modelos para automatizar el desarrollo, despliegue y resultados reunión de una instalación ICDE para fines de planificación de capacidad e fi cientes.

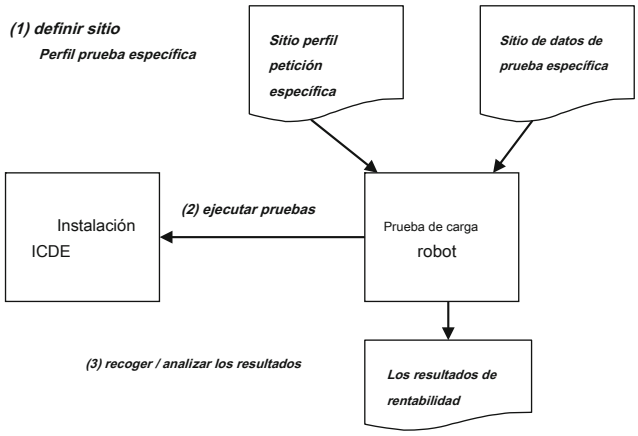


Fig. 14.1 La planificación de capacidad para instalaciones ICDE 202

14.2 ¿Cuál es la MDA?

Un tema recurrente en la evolución de la ingeniería de software es el uso en curso de más lenguajes formales abstractos de soluciones de modelado. En el desarrollo de software de corriente tanto, las descripciones abstractas, por ejemplo, en Java o C #, son transformados por herramientas en formas ejecutables. El desarrollo de soluciones en notación abstracta aumenta la productividad y reduce los errores porque la traducción de lo abstracto a formas ejecutables está automatizado mediante herramientas de traducción como compiladores.

Por supuesto, algunas personas creen que el nirvana de los lenguajes de programación abstractos es Java, C # o cualquiera de sus contemporáneos modernos. De hecho, la historia de la investigación de lenguajes de programación está lleno de muchas propuestas de nuevos lenguajes de desarrollo, algunos de propósito general, algunas restringido a los dominios de aplicación estrechas. Una pequeña minoría nunca ver la luz del día en “developerland”. Esto no impide que la búsqueda continúe sin embargo.

arquitectura basada en modelos (MDA) es una tecnología reciente que lleva el paquete en términos de herramientas más abstractos específicos fi cación y desarrollo (y el uso de nuevas siglas) dirigidos al mercado de TI. **MDA está definido por el OMG [2] como " una aproximación al sistema informático las especi fi caciones que separa la especificación de la funcionalidad de la especi fi cación de la aplicación ”.**

Como su nombre indica, un “modelo de aplicación” es la fuerza impulsora detrás de la MDA. Amodel en MDA es una especificación formal de la función, estructura y / o el comportamiento de una aplicación o sistema. En el enfoque de MDA, un sistema de TI es primero analizada y específica ed como un “Independent Modelo Computation” (CIM), también conocido como un modelo de dominio. La CIM se centra en el medio ambiente y los requisitos del sistema. Los detalles de cálculo y aplicación del sistema están ocultos en este nivel de descripción, o están aún por determinar.

Como la fig. 14.2 espectáculos, la CIM se transforma en una “Plataforma Independiente Modelo” (PIM) que contiene información computacional para la aplicación, pero no hay información específico a la tecnología de la plataforma subyacente que se utilizará para implementar, finalmente, el PIM. Finalmente, un PIM se transforma en una “Plataforma Speci fi c Model” (PSM), que incluye descripciones detalladas y elementos especí fi cos a la plataforma de implementación específica.

Una “plataforma” en MDA se define como cualquier conjunto de subsistemas y tecnologías que proporcionan un conjunto coherente de funcionalidades a través de interfaces y el uso especificado fi

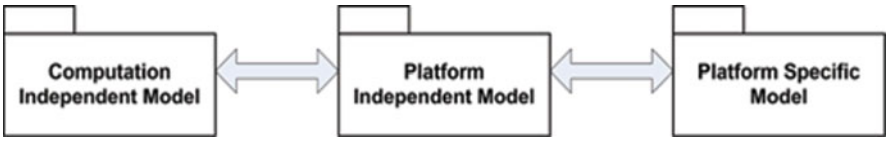


Fig. 14.2 la transformación del modelo de MDA

[2] Grupo de administración de objetos: http://www.omg.org

patrones. por lo tanto, una plataforma MDA es un concepto muy amplio. Plataformas a menudo se refieren a la tecnología de conjuntos específicos fi de subsistemas que se definen por una norma, tal como CORBA o JEE. Plataformas también pueden referirse a una plataforma específica fi proveedor que es una implementación de un estándar, al igual que la plataforma WebLogic de BEA JEE, o una tecnología patentada como la plataforma Microsoft .NET.

MDA es apoyado por una serie de normas OMG, incluyendo el UML, MOF (Meta-Object Facility), XMI (XML Metadata Interchange), y CWM (Common Metamodelo de depósito). MDA también incluye directrices y normas cambiantes de apoyo en la transformación de modelos y servicios generalizados. Las normas de la MDA en conjunto definen cómo un sistema puede ser desarrollado siguiendo un enfoque basado en modelos y el uso de herramientas MDA compatibles. Cada estándar MDA tiene su papel único en el panorama general de la MDA.

En MDA, los modelos deben ser indicado por un lenguaje de modelado. Esto puede ir desde lenguajes de modelado genéricas aplicables a múltiples dominios (por ejemplo, UML) a un específico lenguaje de modelado de dominio c. El MOF proporciona facilidades para especificar cualquier lenguaje de modelado utilizando las instalaciones de modelado conocido de MOF, como se muestra en la Fig. 14.3 .

El MOF también proporciona mecanismos para determinar cómo cualquier modelo definido en un lenguaje de modelado se puede serializar en documentos XML o ser representado por interfaces programables. Cualquier lenguaje de modelado existente puede hacerse compatible MDA mediante la creación de una representación MOF de la lengua.

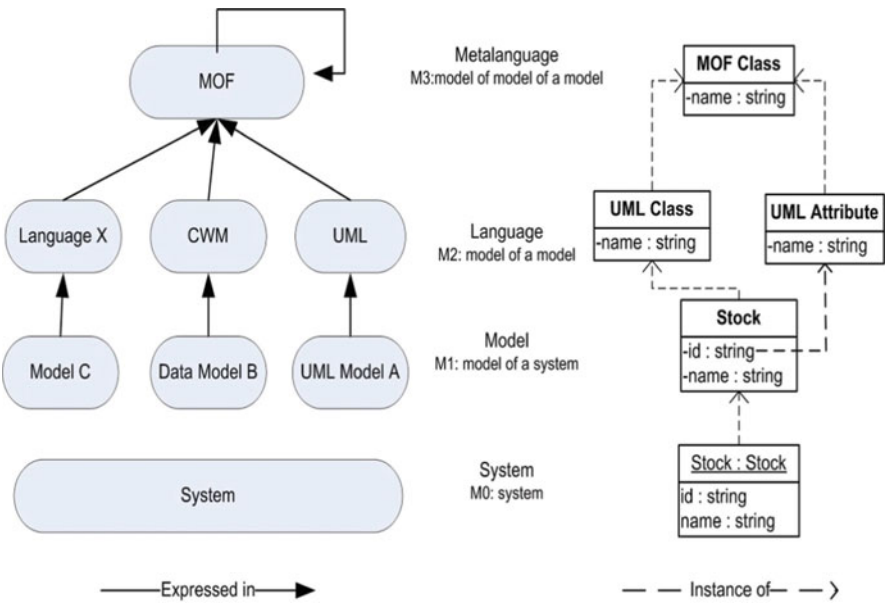


Fig. 14.3 El papel del MOF en MDA 204

El UML y CWM son dos lenguajes de modelado fi nidas relativamente genéricos MOF-DE y se incluyen en el paquete de normas MDA. UML se centra en el modelado de objetos y CWM se centra en el modelado de datos.

El estándar XMI en MDA es una aplicación que se puede utilizar para definir la forma de un esquema XML e instalaciones de serialización XML relacionados se pueden derivar de un lenguaje de modelado metamodelo se especi fi ca utilizando el MOF. Por ejemplo, el OMG ha aplicado a XMI el metamodelo UML para llegar a un esquema XML para la representación de modelos UML. En consecuencia, el esquema XML para los modelos UML puede ser utilizado por los proveedores de herramientas de modelado UML para intercambiar modelos UML.

Así, a partir de modelos de dominio de negocios, a los modelos de análisis, para diseñar modelos y modelos de código finalmente, los principios de MDA cubren todas las fases de los procesos de desarrollo de software, artefactos y herramientas. En las siguientes secciones, vamos a discutir los ts generales bene fi de MDA y dar algunos ejemplos.

14.3 ¿Por qué MDA?

Modelos desempeñan un papel central en MDA. Pero ¿por qué es exactamente lo que necesitamos modelos? Aquí está la respuesta.

Los modelos proporcionan abstracciones de un sistema que permiten a las diversas partes interesadas para razonar sobre el sistema desde diferentes puntos de vista y niveles de abstracción. Los modelos se pueden utilizar de muchas maneras, por ejemplo, para predecir las cualidades (por ejemplo, rendimiento) de un sistema, validar diseños frente a los requisitos, y para comunicar las características del sistema a los analistas de negocios, arquitectos e ingenieros de software. Y lo más importante en el mundo de la MDA, que pueden ser utilizados como modelo para la implementación del sistema.

Los tres objetivos primarios de MDA son la portabilidad, la interoperabilidad y la reutilización, logrado a través de la separación arquitectónica de preocupaciones. problemas de diseño críticos relativos a la CIM, PIM y PSM son de naturaleza muy diferente y pueden evolucionar de forma independiente el uno del otro. Múltiples CIM, PIM y PSM pueden existir para una aplicación, reflectante diferentes niveles de refinamiento fi y puntos de vista. Vamos a ver cómo se logran estos objetivos primarios en MDA.

14.3.1 Portabilidad

Portabilidad se consigue mediante la separación de modelo y de transformación. modelos de alto nivel no contienen plataforma de bajo nivel y los detalles técnicos. Como la fig. 14.4 ilustra, cuando las plataformas subyacentes cambian o evolucionan, los modelos de nivel superior pueden ser transformados a una nueva plataforma directamente, sin ninguna remodelación.

Portabilidad también se consigue haciendo que los modelos movible a través de diferentes entornos de herramientas. Las normas de MOF y XMI permiten un modelo UML a ser serializado en documentos XML que se pueden importar en una nueva herramienta para diversos propósitos de modelado y análisis.

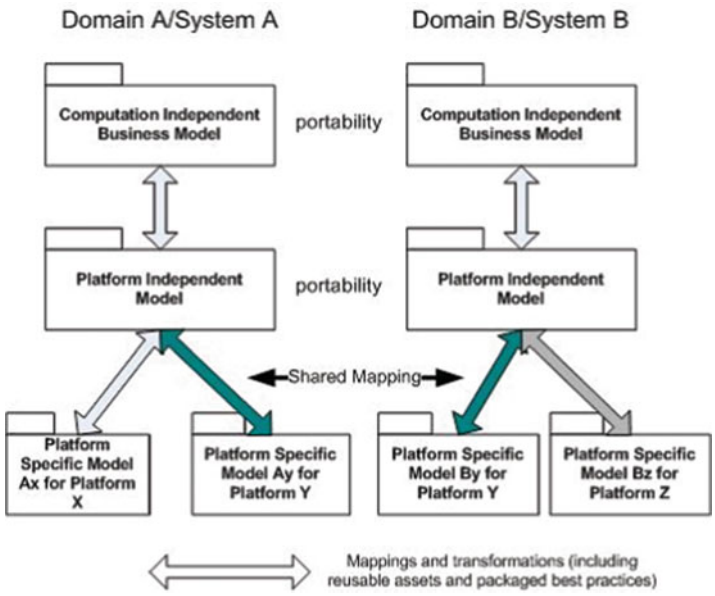


Fig. 14.4 asignaciones de modelo MDA 206

14.3.2 interoperabilidad

Pocas veces existe una aplicación que no se comunica con otras aplicaciones. aplicaciones de nivel empresarial en particular necesitan comunicarse a través de fronteras organizativas internas y externas de una manera heterogénea y distribuida. La mayor parte del tiempo, que tienen un control limitado sobre los otros sistemas que necesita para interoperar con.

El uso de MDA, la interoperabilidad se consigue a través del mapeo modelo horizontal y la interacción (ver Fig. 14.5). Las primeras versiones de las directrices de MDA se refieren a la integración como la principal meta para la MDA, cuyo objetivo es mejorar la interoperabilidad de dos maneras:

- El problema de la interoperabilidad puede ser visto como un problema de modelo horizontal

  mapeo y la interacción. Por simplificación, supongamos que tenemos dos conjuntos de CIM / PIM / PSM para los dos sistemas, tal como se muestra en la Fig. 14.5 . La interacción entre CIMs de nivel superior y PSMs puede primera modelada y analizada. Estas asignaciones modelo transversales y las interacciones a continuación, se pueden asignar a los protocolos de comunicación detallada o bases de datos compartidas soportados por los modelos subyacentes. Desde existen transformaciones verticales explícitos entre los modelos en cada sistema, los elementos que intervienen en la asignación de alto nivel pueden rastrearse fácilmente o incluso traducirán automáticamente a elementos de nivel inferior.

- El mismo problema también puede ser visto como un problema de la re fi nición de un solo nivel alto

  modelo en varios modelos que opera a través de dos o más plataformas. Diferente

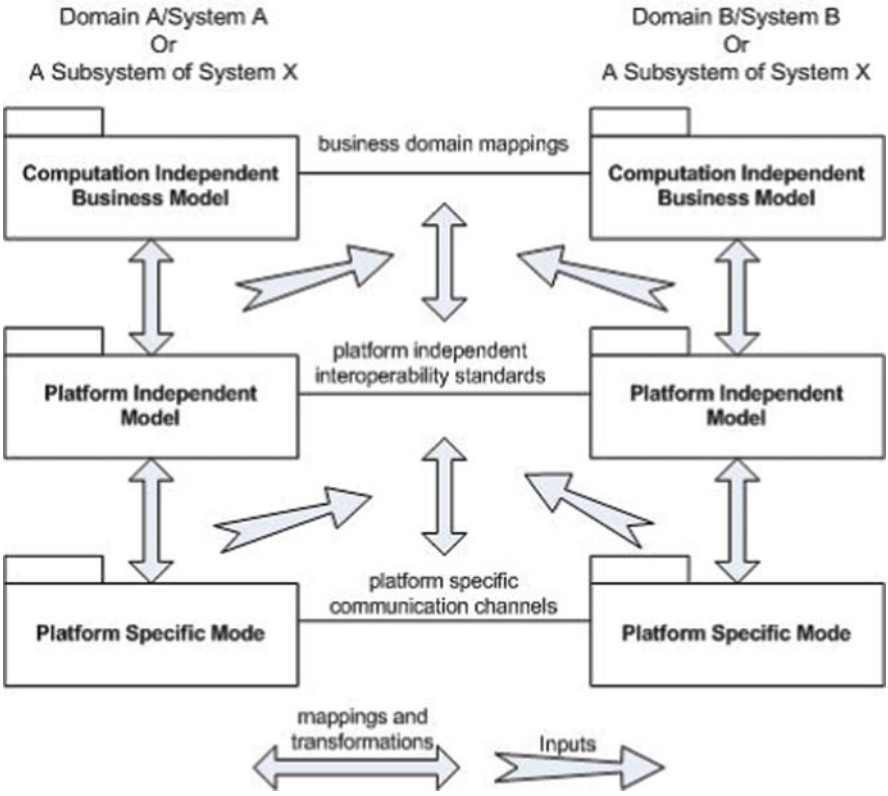


Fig. 14.5 mapeo modelo horizontal para la interoperabilidad

partes de los modelos de nivel superior son re definido en modelos específica a diferentes plataformas. Asociaciones en los modelos originales son re definido en los canales de comunicación o de bases de datos entre modelos de plataforma específica c compartido.

Con instalaciones metamodelado unificado y herramientas de transformación de modelos explícitos, estos dos enfoques se hacen factible en la práctica.

14.3.3 Reutilización

Reutilización es la clave para mejorar la productividad y la calidad. MDA fomenta la reutilización de modelos y mejores prácticas en el diseño de aplicaciones, especialmente en la creación de familias de aplicaciones como en las líneas de productos de software (véase el capítulo siguiente). MDA es compatible con la línea de productos de software se acerca con el aumento de los niveles de automatización. Por ejemplo, el PIM está destinado para su reutilización por mapeo a diferentes PSMs que una línea de productos apoya, y una plataforma de MDA está diseñado para su reutilización como un objetivo para múltiples aplicaciones en una línea de productos.

14.4 Prácticas y Herramientas de Estado-de-arte

Aunque es posible practicar las partes de la MDA sin el apoyo de herramientas, esto sólo se recomienda para los valientes y dedicados. Una gran parte de las normas está dirigido a los útiles y la interoperación herramienta. Algunas normas están destinadas a ser principalmente de lectura mecánica, y no para el consumo humano en general.

Dado que los patrones de MDA, en especial las directrices, son intencionalmente sugerente y no prescriptiva, ha habido una gran cantidad de herramientas para apoyar claming MDA, todos con muy diferentes características y capacidades. Algunas partes definido vagamente de de MDA han causado problemas en términos de interoperabilidad y reutilización herramienta de desarrollo artefacto. Sin embargo, el equilibrio correcto entre las normas preceptivas y no prescriptivas es difícil de determinar a priori y no requiere entradas del mundo real de usuarios de la industria.

Ahora vamos a discutir algunos ejemplos de herramientas de la comunidad de la plataforma JEE / Java debido a su relativamente amplia adopción de la MDA. La plataforma .NET también se está moviendo hacia enfoques basados en modelos estándar a través de su propio dominio especí fi co Idioma (DSL). Esto no es compatible con MDA aunque otros proveedores han desarrollado con éxito herramientas MDA para la plataforma .NET.

Aunque las herramientas descritas en el siguiente tienen sus raíces en las tecnologías JEE / Java, todos aquí tienen la capacidad de soportar otras plataformas. Los servicios de arquitectura e infraestructura de estas herramientas permiten que toda extensiones y “cartuchos” para ser construidos para soportar otras plataformas. Algunos de ellos simplemente tienen fuera de la caja de soporte para las tecnologías relacionadas JEE.

14.4.1 AndroMDA

AndroMDA [3] es una fuente abierta marco MDA. Tiene una arquitectura plug-in en el que las plataformas y componentes de apoyo se pueden intercambiar y salir en cualquier momento. Es muy explota proyectos existentes de código abierto para ambos propósitos especí fi cos de plataforma (por ejemplo, XDoclet para EJB) y servicios generales de la infraestructura (Apache Velocity para crear plantillas de transformación).

En AndroMDA, los desarrolladores pueden extender el lenguaje de modelado existente a través de las instalaciones conocidas como “metafacades”. La extensión se refleja como un per fi l de UML en las bibliotecas de modelado y plantillas en herramientas de transformación. El enfoque actual de AndroMDA es generar la mayor cantidad de código posible de un PIM marcada con UML valores etiquetados, sin tener un PSM explícita fi l (sólo existe en la memoria). Por lo tanto, no proporciona oportunidades para la inspección y manipulación PSM bidireccional entre PSM y PIM.

La razón de esto es principalmente debido a la compensación entre la complejidad de bidireccional PIM / trazabilidad PSM y los beneficios de mantener las MEP explícitas

---

3 http://www.andromda.org/

para diferentes plataformas. A nivel estereotipo de UML, este enfoque por lo general funciona bien porque la plataforma único general semántica independientes están involucrados, pero para la generación de código, las marcas a través de los valores etiquetados por lo general incluye información dependiente de la plataforma que contamina los PIM a un cierto grado.

14.4.2 ArcStyler

Arcstyler [4] es una de las herramientas comerciales líderes en el mercado de la MDA. Es compatible con el JEE, .NET y plataformas fuera de la caja. En adicional a UML per fi les, ArcStyler utiliza sus propios MDA “marcas” como una manera de introducir información dependiente de la plataforma en los PIM sin contaminar el modelo con datos de nivel de plataforma. Al igual que AndroMDA, ArcStyler soporta cartuchos extensibles para la generación de código. Los cartuchos ellos mismos también se pueden desarrollar dentro del entorno ArcStyler siguiendo los principios de MDA. La herramienta también es compatible con el modelo de transformación de modelos a través regla de transformación explícita externa archivos.

14.4.3 Eclipse Modeling Framework

El vínculo inseparable entre modelos MDA y el código creado a través de la generación de código requiere una gestión coherente de los modelos y el código en un solo IDE. Eclipse que modela el marco (EMF) es el sofisticado marco metamodelado y modelado detrás del IDE de Eclipse. Aunque EMF solamente se dio a conocer como un subproyecto Eclipse en 2003, tiene una larga historia como un motor de gestión de metadatos basado en modelos de IDE Visual Age de IBM.

EMF es en gran parte MDA compatible con sólo ligeras desviaciones respecto de algunas de las normas. Por ejemplo, la base de la lengua metamodelado de EMF se conoce como Ecore, que está cerca pero no idéntica a la MOF esencial (EMOF) en MOF 2.0. EMF por lo general puede cargar un metamodelo EMOF construido, y asignaciones y transformaciones se han desarrollado entre EMOF y Ecore.

EMF viene con mecanismos estándar para la construcción de metamodelos y persistiendo como interfaces programables, código y XML (ver Fig. 14.6). También se proporcionan un marco editor de modelos y un marco de generación de código. Sin embargo, los CEM no incluye ningún apoyo popular plataforma fuera de la caja, y no impresionó a la comunidad inicialmente como una herramienta MDA MDA-lista para usar plenamente fl filo para sistemas distribuidos basados en la plataforma.

Sin embargo, la estrecha integración de los CEM con el IDE Eclipse y la capacidad de aprovechar la arquitectura de Eclipse y las infraestructuras comunes apoya la integración de metadatos dispares a través de múltiples herramientas que cooperan en una común

4 http://www.interactive-objects.com/en/soa-governance.html

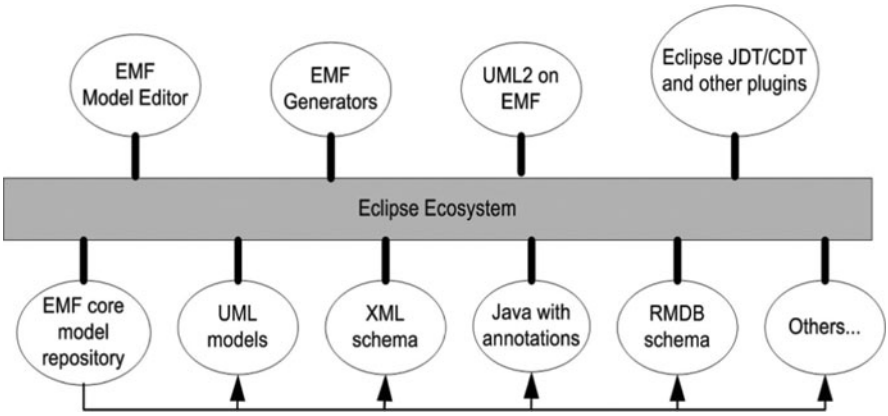


Fig. 14.6 El marco de modelado Eclipse 210

ecosistema Eclipse-basada. Esto eleva el nivel de interoperabilidad herramienta mientras está en gran medida compatible con las prácticas de MDA.

Este es también un ejemplo que demuestra que los basados en modelos principios y normas van más allá de la modelización del sistema, e incluyen el modelado de todos los aspectos de la construcción del sistema. Con poca fanfarria, IBM ha migrado muchas de sus herramientas de desarrollo Eclipse y gestiona sus metadatos a través de los CEM. proveedores de terceros también están desarrollando activamente herramientas basadas en EMF.

Debido a la normalización en curso de transformación de modelos y de los significantes aumentos de la producción de la generación de código, la mayoría de las herramientas existentes se centran en la generación de código a partir de modelos. El soporte para el modelo de transformación de modelos por lo general se carece. Esto se traduce en apoyo primitiva para la transformación bidireccional CIM-PIM-PSM. En general, sin embargo, el mercado está madurando MDA con herramientas tanto fuerza de la industria comercial y de código abierto emergente.

14.5 MDA y Arquitectura de Software

La mayoría de los modelos inMDA son esencialmente representaciones de una arquitectura de software. En un sentido amplio, los modelos de dominio y los modelos de sistemas son abstracciones y diferentes puntos de vista de los modelos de arquitectura de software. modelos de código generadas poseen las características de los modelos de arquitectura, junto con los detalles de implementación. El código puede de hecho ser utilizado en herramientas de ingeniería inversa para reconstruir la arquitectura de la aplicación.

Una arquitectura de software puede ser descrito en un lenguaje de descripción de la arquitectura (ADL). Ha habido muchas actividades cotidianas desarrolladas en los últimos años, cada uno con su expresividad se centraron en diferentes aspectos de los sistemas de software y dominios de aplicación. Muchas de las funciones ADL útiles han sido recientemente ya sea absorbido en revisiones de la UML, o específico ed como de peso ligero (a través de UML pro fi les) o (MOF) extensiones UML de peso pesado. Por lo tanto, el UML se utiliza inMDA como un ADL.

Algunos formalismos exóticas y características dinámicas de ciertas AVD todavía no se pueden expresar plenamente con UML. Pero la creciente MDA / UML piscina de experiencia en la industria, junto con la arquitectura de alta calidad y herramientas de modelado UML supera la desventaja de algunas limitaciones de modelado en la mayoría de los dominios.

14.5.1 MDA y requerimientos no funcionales

requisitos no funcionales (NFR) son una de las principales preocupaciones de la arquitectura de software. NFR incluyen requisitos relacionados con los atributos de calidad como el rendimiento, la capacidad modi fi, reutilización, interoperabilidad y seguridad. Aunque MDA no se ocupa de cada atributo de calidad individual directamente, promueve y ayuda a lograr estos atributos de calidad porque:

- Un cierto grado de interoperabilidad, reutilización y portabilidad está integrado en todos modelos a través de la separación inherente de las preocupaciones. Hemos explicado cómo se logran estos beneficios en las secciones anteriores.
- El Ministerio de Hacienda y UML per fi l de mecanismos permiten UML para ser extendido por modeling requisitos y elementos de diseño especí fi camente la orientación NFR. UML per fi les de NFR existen expresan, como del OMG per fi l para el funcionamiento, la programación y el tiempo.
- Junto con las extensiones de modelado para NFR requisitos y el diseño, explícitos reglas de asignación modelo animan a hacer frente a los atributos de calidad durante la transformación del modelo.

14.5.2 Transformación del modelo y de Arquitectura de Software

Una gran parte de la arquitectura de software de I + D se refiere a la forma de diseñar y validar las arquitecturas de software para que fi ll cumplir sus requisitos y se aplica fielmente al diseño. Uno de los principales obstáculos en el diseño de la arquitectura es la di fi cultad de diseñar una arquitectura que capta claramente cómo los diversos aspectos del diseño satisfacen los requisitos. Por esta razón, puede ser difícil para validar sistemáticamente si los modelos de arquitectura fi ll cumplir los requisitos, como la trazabilidad entre requisitos y elementos de diseño no se formaliza. Esto no ayuda a aumentar con fi anza que la arquitectura es fi cio para el propósito.

En MDA, todos los lenguajes de modelado son bien definido por sintaxis y la semántica en un metamodelo. El proceso de transformación de un modelo (por ejemplo, requisitos) a otro modelo (por ejemplo, diseño) es un proceso sistemático, siguiendo las reglas de transformación Ned explícitamente de fi. Este carácter explícito y el potencial de automatización puede mejorar en gran medida la calidad y e fi ciencia de la validación de un modelo de arquitectura.

El estándar de transformación de modelos que ha surgido de la OMG se conoce como “Query, Ver y Transformación” (QVT). En el momento de la escritura hay varios productos (comerciales y de código abierto) que dicen ser compatibles con el estándar QVT.

QVT define un método estándar para transformar los modelos de origen en los modelos de destino. Estos se basan en la idea de que el programa de transformación es en sí misma un modelo, y como consecuencia se ajusta a un metamodelo MOF. Esto significa que la sintaxis abstracta de QVT también se ajusta a un metamodelo MOF.

Si el QVT ganancias de tracción estándar generalizado, es posible que gran parte del conocimiento tácito, las mejores prácticas y patrones de diseño utilizados en el diseño de la arquitectura y la evaluación será formalmente codi fi cado como diversas formas de reglas de transformación bidireccionales. Estos crearán formas ricas de trazabilidad en modelos de arquitectura. De hecho, las transformaciones basadas en los patrones y las mejores prácticas que ya se han implementado en algunas herramientas, además de las asignaciones normales plataforma especí fi cos entre PIM y PSM.

14.5.3 SOA y MDA

BothMDA y SOA tratan de resolver el mismo problema de la interoperabilidad pero desde un punto de vista y el nivel de abstracción totalmente diferente. Una de ellas es desde la perspectiva general de modelado semántico; el otro es de los protocolos de comunicación y la perspectiva de la arquitectura de estilo. Después de MDA, es posible mapear consistentemente interacciones semánticas de alto nivel y las asignaciones entre los dos sistemas en los elementos de modelo de nivel inferior y canales de comunicación con servicios de apoyo necesarios.

beneficios de tener modelos analíticos que también son compatibles con la MDA son potencialmente enormes. 212

MDA también puede aumentar la productividad cuando las funciones de un sistema necesitan ser expuestos como servicios Web, uno de los requisitos más comunes en SOA. Si el sistema existente ya está modelada siguiendo las reglas de la MDA, exponiendo sus servicios es sólo una cuestión de la aplicación de reglas de transformación para la plataforma de servicios web. Por ejemplo, en AndroMDA, el cartucho “webservice” ofrece WSDL y WSDD fi le generación usando un simple UML per fi l. Para exponer la misma lógica de negocio como servicios web, los usuarios sólo tienen que cambiar el PIM de procesos de negocio (el objetivo final es tener ningún cambio) y utilizar el cartucho de “servicio web”.

En resumen, SOA puentes sistemas heterogéneos a través de protocolos de comunicación, servicios incluso en las directrices de fi ciales de MDA. Sin embargo, al igual que los modelos de transformación QVT, los generalizados y un estilo de arquitectura orientada a servicios asociados. MDA puede hacerse cargo de la perfecta integración semántica de alto nivel entre los sistemas y la transformación de los modelos de sistemas basados en instalaciones de nivel inferior SOA. Esta sinergia entre MDA y SOA podría significar que la próxima generación de servicios orientados mundo de la informática con una arquitectura flexible y altamente federada fl no está demasiado lejos.

14.5.4 Los modelos analíticos son demasiado Modelos

La importancia del uso de modelos analíticos para examinar las características de un sistema es a menudo ignorado,

De acuerdo con la MDA definición, un modelo se define como una descripción de un sistema en un fi lenguaje bien de nido. Esta definición se puede aplicar a una amplia gama de modelos. Por ejemplo, en ingeniería de rendimiento, podemos elegir ver un sistema como un modelo basado en la cola que tenga servidores y colas. En el análisis de la capacidad fi modi, podemos elegir ver un sistema como un modelo gráfico de la dependencia que tiene nodos para representar elementos conceptuales de aplicación, y bordes para representar relaciones de dependencia entre ellos.

Actualmente, estos modelos se expresan normalmente en sus propios lenguajes de modelado. Con el fin de construir un modelo de análisis de un modelo UML existente, ya sea que tenemos que hacer el modelado manual o una transformación bajo nivel debe llevarse a cabo basándose en el modelo UML representado en XML. Esto se muestra en la figura. 14.7 , Y tiene varias limitaciones:

- La transformación se basa únicamente en las instalaciones de transformación XML primitiva tales como XSLT. Depuración y mantenimiento es difícil sin mapeo semántico claro entre los dos modelos.
- Sin un mapeo semántico claro e instalaciones de ingeniería de ida y vuelta, es muy difícil de colocar los resultados obtenidos del modelo analítico de nuevo en el contexto del modelo original UML.
- El modelo de diseño original es probable que sea aún más el fuego y, finalmente, define implementado en el código. El modelo analítico es esencialmente también un modelo derivado del mismo modelo de diseño. Pero como el modelo analítico no es compatible con el estándar MDA, es aún más difícil de referencia cruzada el modelo analítico con todos los otros modelos derivados para la validación, calibración y otros fines.

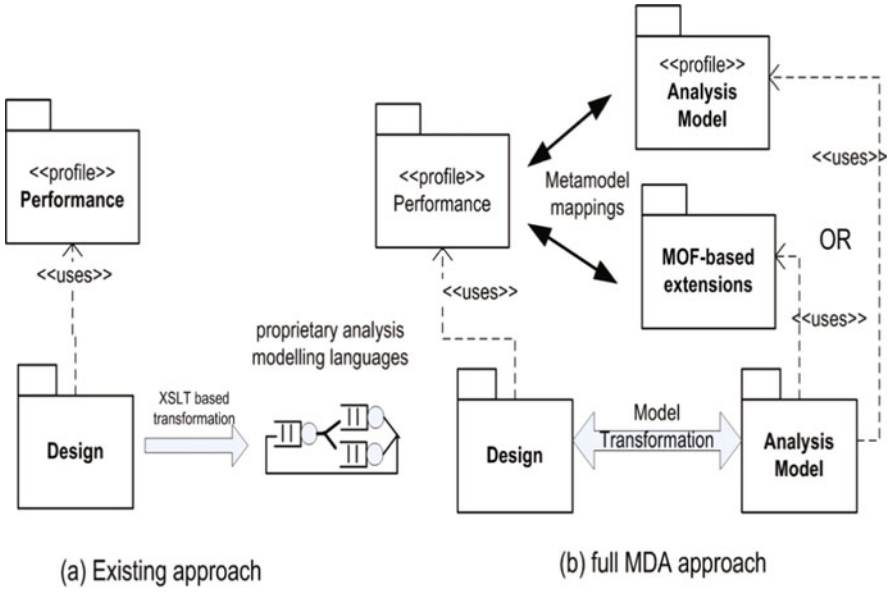


Fig. 14.7 transformación de modelos MDA para el análisis de modelo

14.6 MDA para la planificación de la capacidad del ICDE

Con el fin de llevar a cabo la planificación de capacidad para las instalaciones de la ICDE, el equipo ICDE necesita un conjunto de pruebas que podrían ser adaptado rápidamente para definir una carga de prueba fi co sitio-específica. A continuación, debería ser sencillo y rápido para ejecutar el conjunto de pruebas en el entorno de despliegue previsto, y reunir las estadísticas de rendimiento, tales como el rendimiento y tiempo de respuesta.

Después de un vistazo de cerca a sus requisitos de las pruebas de rendimiento, el equipo encontró que ICDE sus necesidades de desarrollo rápido a través de diferentes plataformas JEE eran susceptibles de aplicación de los principios de MDA, aprovechando su apoyo a la portabilidad, interoperabilidad y reutilización. Las razones son las siguientes:

- Para los diferentes servidores de aplicaciones JEE, sólo el código de plomería relacionados plataforma y los detalles de implementación difieren. UsingMDA, un modelo de aplicación genérica podría ser utilizado, y el código c plataforma específica y de plomería genera a partir del modelo. Esto aprovecha la portabilidad inherente a la MDA.
- La generación de código de plomería repetitivo y despliegue con fi guración es apoyado por muchos servidores de aplicaciones JEE por una serie de proyectos de código abierto de MDA. Estos cartuchos de generación de código se mantienen normalmente por una gran comunidad de usuarios activa, y son de alta calidad. Por lo tanto la capacidad de reutilizar estos cartuchos de herramientas MDA era muy atractiva.
- El equipo ICDE tiene una amplia experiencia en el rendimiento y la carga de la prueba. Por refactorización sus bibliotecas existentes en un marco reutilizable, gran parte de este se puede reutilizar fácilmente a través de plataformas JEE. Sin embargo, cada prueba c fi sitio-específico requerirá un código personalizado que se creará para capturar los requisitos del cliente. El uso de MDA, estas características del sitio-específico c pueden ser representados usando los estereotipos UML y los valores etiquetados, como una combinación de detalles de modelado e información con fi guración. A partir de esta descripción del diseño, el cartucho de generación de código MDA puede producir las características del sitio-específico C y enganche con estos en componentes reutilizables marco del equipo.

Por lo tanto, el equipo diseñó un ICDE UML per fi l y una herramienta para automatizar la generación de conjuntos de pruebas de rendimiento ICDE completos de una descripción del diseño. La entrada es un conjunto basado en UML de diagramas de diseño para la aplicación de referencia, junto con un cliente de pruebas de carga modelada en una versión de rendimiento a medida de la UML 2.0 Testing Pro fi le.[5] La salida es un conjunto de pruebas de despliegue que incluye el monitoreo, pro fi ling y los servicios públicos de información. La ejecución de la aplicación de referencia generada produce datos de rendimiento en formatos de fácil análisis, junto con gráficos de rendimiento generadas automáticamente. La herramienta está construida encima de un marco extensible de código abierto - AndroMDA. La estructura general de la generación de referencia y de trabajo proceso relacionado de flujo se presenta en el área de caja en la Fig. 14.8 . Un fragmento del modelo se representa en la Fig. 14.9 . La entrada de las pruebas de carga

punto es el ICDEAPIService. Es el componente de extremo delantero del sistema bajo

[5] http://www.omg.org/technology/documents/formal/test_pro fi le.htm

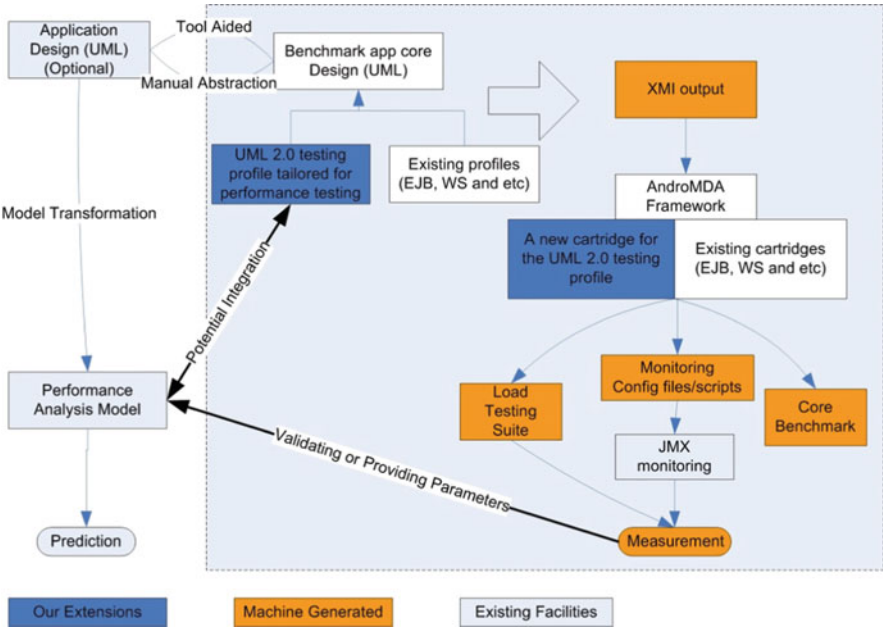


Fig. 14.8 Descripción general del generador de prueba de rendimiento basado en MDA del ICDE

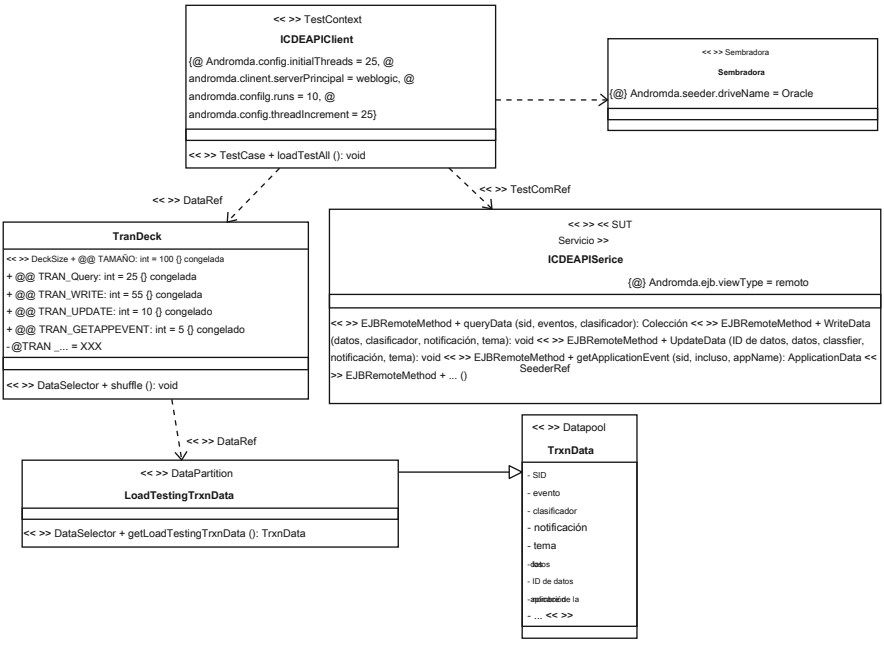


Fig. 14.9 modelo de prueba de rendimiento ICDE

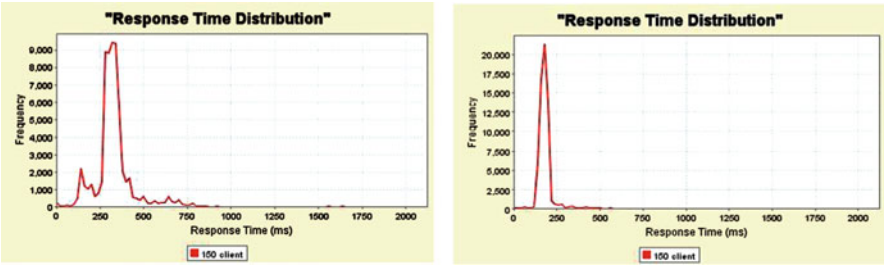


Fig. 14.10 resultados de tiempo de respuesta Ejemplo 216

prueba, que está marcado con el SUT << >> estereotipo. ICDEAPIClient es el << TestContext >> que consiste en un número de casos de prueba. Sólo el valor por defecto loadTestAll () caso de prueba se incluye con su incumplimiento genera la aplicación.

Todos los datos de prueba que se utilizarán para llamar a las API del ICDE se modela en el TrxnData clase. los TranDeck clase contiene valores que con fi gura la mezcla de transacción para una prueba con valores etiquetados, que se muestra en la Fig. 14.9 . Por ejemplo, las llamadas a la API ICDE queryData representa el 25% de todas las transacciones y WriteData representa el 55% para la prueba se define en este modelo. Estos datos son utilizados para generar aleatoriamente los datos de prueba que simula la carga de trabajo real de la instalación ICDE bajo prueba.

En la Fig. 14.10 , Salidas Ejemplo de ensayo se representan para la distribución del tiempo de respuesta para dos servidores de aplicaciones diferentes bajo una carga de trabajo de 150 clientes simultáneos.

La cantidad de tiempo que se ahorra el uso de MDA puede ser considerable. cartuchos de tecnología Communitymaintained generan automáticamente el código de error repetitivo y propenso plomería, y las mejores prácticas heredadas mediante el uso de los cartuchos de mejorar la calidad del software de pruebas de rendimiento. Por encima de todo, los principios de MDA elevar el nivel de abstracción del desarrollo conjunto de pruebas, por lo que es fácil y barato de modificar y ampliar.

Para más información sobre este trabajo, por favor refiérase a la referencia MDABench al final del capítulo.

14.7 Resumen y lectura adicional

MDA, como la industria amplia estandarización de modelo impulsado el desarrollo de software, está teniendo éxito y sigue evolucionando. MDA impactos sobre las prácticas de la arquitectura del software, ya que requiere el equipo de arquitectura para crear modelos formales de su aplicación utilizando rigurosamente de fi nida lenguajes de modelado y herramientas de apoyo. Esto representa esencialmente elevar el nivel de abstracción para los modelos de arquitectura. La industria del software ha estado elevando los niveles de abstracción en el desarrollo de software (por ejemplo, a partir del código máquina a lenguaje ensamblador a 3GLs de idiomas orientados a objetos y ahora a los modelos) para la mejor parte de cinco décadas. MDA es el último paso en esta dirección, y si lo logra los objetivos de la industria podría alcanzar nuevos niveles de productividad de desarrollo única soñado hoy.

Sin embargo, llama la MDA críticas por parte de muchas partes en cuanto a sus limitaciones, algunas de las cuales son, posiblemente, intrínseca y duro para mejorar sin una revisión a fondo. Microsoft ha optado por no cumplir con las normas de MDA y seguir su propio camino, de fi nir y utilizar su propio DSL como el lenguaje de modelado en su IDE de Visual Studio. Si bien esto puede astillar la comunidad de desarrollo y crear modelos y herramientas incompatibles, es probable que tenga resultados positivos para la comunidad de software en los próximos años, tanto en la promoción de los principios de desarrollo modeldriven general de Microsoft de OMG y.

La mejor referencia para toda la información relacionada con el estándar MDA es el sitio web de la OMG:

DIOS MIO, Guía de MDA versión 1.0.1. http://www.omg.org/mda/

Algunos buenos libros sobre MDA de autores destacados son:

Thomas Stahl, Markus Voelter, Software Development Model-Driven: Tecnología, Ingeniería, Administración, Wiley 2006.

David Steinberg, Frank Budinsky, Marcelo Paternostro, Ed Merks, EMF: Eclipse Marco de modelado, Addison Wesley Professional, 2ª edición, 2008. Michael Guttman, Juan Parodi, La vida real MDA: Solución de problemas de negocios con Driven Architecture modelo, Morgan Kaufman 2006.

SJ Mellor, S. Kendall, A. Uhl, D. Weise. MDA destilada. Addison-Wesley, 2004.

Para algunos detalles más sobre las herramientas de planificación de capacidad y rendimiento basadas en MDA, consulte:

L. Zhu, J. Liu, I. Gorton, NB Bui. Generación personalizada comparativas realizadas con MDA. en Actas de la 5ª Conferencia IEEE de Trabajo / IFIP en Arquitectura de Software, Pittsburgh, noviembre de 2005.

capítulo 15
Líneas de Producto Software

Marcar el Staples

15.1 líneas de productos para ICDE

El sistema ICDE es una plataforma para la captura y la difusión de información que se puede utilizar en diferentes dominios de aplicación. Sin embargo, como cualquier tecnología aplicable genéricamente horizontal, su gran atractivo es a la vez una fortaleza y debilidad. La debilidad se debe al hecho de que una organización usuario tendrá que adaptar la tecnología para adaptarse a su dominio de aplicación (por ejemplo, financiar), y hacer más fácil para sus usuarios aprender y explotar. Esto lleva tiempo y dinero, y por lo tanto, es un desincentivo para la adopción.

Reconociendo esto, el equipo de desarrollo decidió producir una versión adaptada de la plataforma ICDE por sus tres grandes dominios de aplicación, a saber, el análisis fi nanciero, análisis de inteligencia e investigación de la política del gobierno. Cada uno de los tres serían comercializados como productos diferentes, y contienen componentes especí fi cos que conforman la plataforma de base ICDE más fácil de usar en el dominio de la aplicación específica.

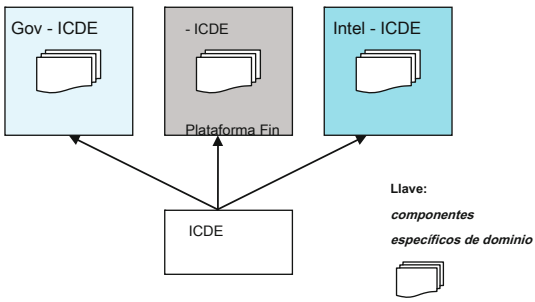
Para lograr esto, el equipo de lluvia de ideas varias estrategias que podrían emplear para minimizar el esfuerzo de diseño y desarrollo de los tres productos diferentes. La idea básica era que se asentaron en utilizar la plataforma de base ICDE sin cambios en cada uno de los tres productos. Ellos entonces crear componentes fi dominio especí adicionales en la parte superior de la plataforma base, y construir los productos resultantes mediante la compilación de la plataforma base con los componentes de dominio-específico. Esta arquitectura básica se representa en la Fig. 15.1 .

Lo que el equipo había hecho era tomar los primeros pasos para la creación de una arquitectura de línea de producto para su tecnología ICDE. Las líneas de productos son una forma de estructuración y gestión del desarrollo en curso de una colección de productos en una fi ciente altamente ef y de manera rentable. Las líneas de productos a lograr la reducción de costes y esfuerzo significativo a través de la reutilización de activos a gran escala de productos de software tales como arquitecturas, componentes, casos de prueba y documentación.

El equipo de desarrollo de productos ya ICDE bene fi cios de la reutilización del software en varias formas diferentes. Reutilizan algunas bibliotecas genéricas (como controladores JDBC para manejar el acceso de base de datos), y la totalidad de las aplicaciones de la plataforma (como la base de datos relacional en

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3_15, # Springer-Verlag Berlin Heidelberg 2011

ICDE 220
dominio-específico c para la plataforma
Fig. 15.1 El desarrollo de productos de



el almacén de datos ICDE). Las fuerzas del mercado están impulsando la introducción de las tres versiones adaptadas del producto ICDE. Pero si el equipo desarrolló cada uno de ellos por separado, podría triplicar su desarrollo o la carga de trabajo de mantenimiento. Por lo tanto, su plan es reutilizar los componentes básicos para la funcionalidad fundamental ICDE y crear componentes personalizados para la funcionalidad específico a cada uno de los tres mercados de productos. Se trata de un tipo de desarrollo de software línea de productos, y se debe reducir significativamente sus costes de desarrollo y mantenimiento.

El resto de este capítulo el desarrollo de líneas descripciones de productos y arquitecturas, y describe una serie de mecanismos de reutilización y de variación que se pueden adoptar para el desarrollo de la línea de productos.

15.2 Líneas de Productos Software

Generalizada reutilización del software es un “santo grial” de la ingeniería de software. Promete un mundo armonioso donde los desarrolladores pueden montar rápidamente soluciones de alta calidad a partir de un conjunto de componentes de software preexistente. La búsqueda de la reutilización de software efectivo en el pasado ha centrado estereotipada de “reutilización en el pequeño,” técnicas para reutilizar función propia, o bibliotecas de funciones para tipos de datos y tecnologías independientes del dominio explotar. clase de colección y librerías de funciones matemáticas son buenos ejemplos. Tales enfoques han demostrado ser beneficioso, pero no se han dado cuenta de la promesa completa de la reutilización del software.

La reutilización de software es fácil si sabes que ya lo hace exactamente lo que quiere. Pero el software que hace “casi” lo que quiere es por lo general completamente inútil. Por esta razón, para darse cuenta de los beneficios completos de la reutilización de software, tenemos que practicar “la variación de software” eficaz también. Los enfoques modernos a la reutilización de software, como el software Línea de Producto (SPL) desarrollo, soporte variación de software “en la gran”, con una base arquitectónica y una fi co enfoque de dominio especí. Línea de Producto (SPL) de desarrollo de software ha demostrado ser una forma efectiva de beneficiarse de la reutilización del software y la variación. Se ha permitido a muchas organizaciones a reducir los costes de desarrollo, reducir la duración del desarrollo, y aumentar la calidad del producto.

En el desarrollo de SPL, una colección de productos se desarrolla mediante la combinación de activos centrales reutilizados con activos fi co-productos personalizados específico que varían la funcionalidad

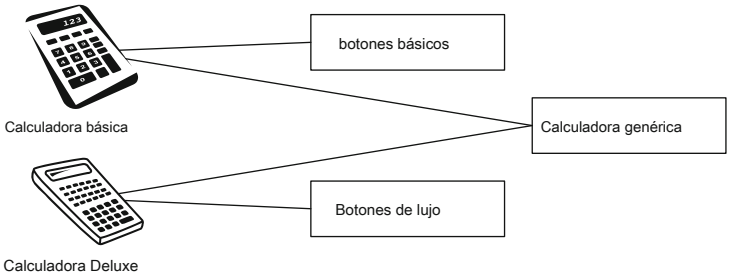


Fig. 15.2 Una vista esquemática de una línea de producto sencillo

proporcionada por los principales activos. Un ejemplo conceptual simple de una línea de productos se muestra en la Fig. 15.2 . En la imagen, dos productos de calculadoras diferentes se desarrollan, tanto con el uso de las mismas tablas internas núcleo de activos. Las diferentes funcionalidades de los productos de calculadoras se ponen a disposición de cada uno de sus activos personalizados, incluyendo los dos tipos diferentes de botones que proporcionan la interfaz individualizada a la funcionalidad genérica, reutilizados.

Desde esta perspectiva sencilla, desarrollo SPL es igual que el desarrollo de la línea de productos basada en hardware más tradicional, excepto que en el desarrollo de SPL, los productos son de software del curso! [1]

Para cualquier producto en un SPL, casi todo es implementado por los principales activos reutilizados. Estos activos centrales implementan la funcionalidad de base que es uniforme a través de productos en la SPL, así como proporcionar apoyo a características variables que pueden ser seleccionadas por los productos individuales. puntos de variación de activos básicos que proporcionan una interfaz para seleccionar de entre esta funcionalidad variable. activos personalizados fi co-productos especí ejemplifican puntos de variación de los activos básicos, y también pueden poner en práctica totalidad de características productspeci fi cos.

variación de software tiene una serie de funciones en el desarrollo de SPL. El papel más evidente es la de apoyar a las diferencias funcionales en las características de la SPL. variación de software también se puede utilizar para apoyar diferencias no funcionales (tales como el rendimiento, la escalabilidad, o la seguridad) en las características de la SPL.

SPL desarrollo no es simplemente una cuestión de arquitectura, diseño y programación. impactos en el desarrollo de SPL procesos en todo el ciclo de vida de desarrollo de software existentes, y requiere nuevas dimensiones de la capacidad de los procesos para la gestión de activos reutilizados, productos, y la presión sonora global en sí. El Instituto de Ingeniería de Software ha publicado directrices de práctica Línea de productos (ver lecturas adicionales al final del capítulo) para estos procesos y actividades que apoyan el desarrollo de SPL. Nos referiremos a estas áreas de práctica más adelante en este capítulo.

[1] Las líneas de productos también son ampliamente utilizados en el dominio de sistemas embebidos, donde los productos son una combinación de software / hardware.

15.2.1 Bene fi ción de Desarrollo SPL

Cuando una organización desarrolla un conjunto de productos que comparten muchos puntos en común, una presión sonora se convierte en un buen enfoque. Típicamente SPL de una organización aborda una amplia área de mercado, y cada producto en el SPL se dirige a un segmento de mercado fi específico. Algunas organizaciones también utilizan un SPL para desarrollar y mantener las variantes de un producto estándar para cada uno de sus clientes individuales.

El alcance de una línea de productos es la gama de posibles variaciones con el apoyo de los principales activos en un SPL. Los productos reales en un SPL estarán normalmente dentro del ámbito de SPL, pero los activos personalizados ofrecer la posibilidad de desarrollar una funcionalidad más allá del alcance normal de la SPL. Para maximizar el beneficio del desarrollo de SPL, el alcance SPL debe ser igual a ambos los mercados de interés para la empresa (para permitir nuevos productos dentro de esos mercados a desarrollarse de forma rápida y cientemente fi ciencia), y también toda la gama de funcionalidad requerida por los productos reales desarrollado por la empresa. Estas tres categorías diferentes de productos (mercados de interés de la compañía, el alcance SPL, y los productos reales desarrollados por la empresa) se representan en un diagrama de Venn en la Fig. 15.3 .

El más obvio beneficio del desarrollo SPL se incrementa la productividad. Los costes de desarrollo y mantenimiento de los activos básicos no son asumidos por cada producto por separado, sino que se extienden a través de todos los productos en el SPL. Las organizaciones pueden capturar estas economías de escala para beneficiarse del desarrollo de un gran número de productos. El enfoque de SPL escala bien con el crecimiento, ya que el coste marginal de agregar un nuevo producto debe ser pequeña.

Sin embargo, el desarrollo de SPL también tiene otras fi cativos beneficios significantes. Cuando los activos principales en un SPL están bien establecidos, el tiempo necesario para crear un nuevo producto en el SPL es mucho menor que con el desarrollo tradicional. En lugar de tener que esperar a la remodelación de la funcionalidad de los principales activos, los clientes sólo tienen que esperar a que el desarrollo de la funcionalidad que es única para sus necesidades.

Las organizaciones también pueden experimentar la calidad del producto beneficios del desarrollo SPL. En el desarrollo de productos tradicionales, un defecto podría repetir en muchos productos, pero en el desarrollo SPL, un defecto en un activo núcleo sólo tiene que ser fijo

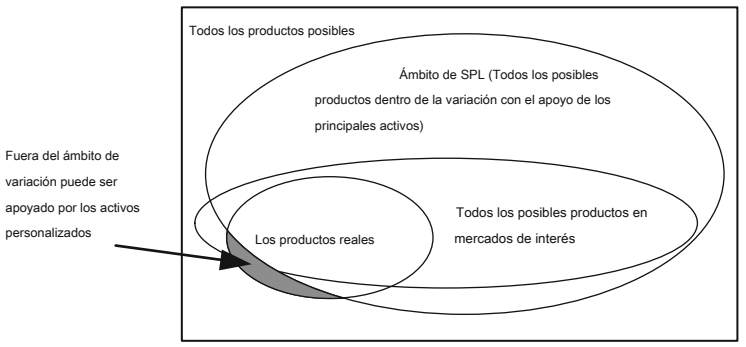


Fig. 15.3 El alcance de una SPL 222

una vez. Por otra parte, aunque el defecto podría encontrarse inicialmente en el uso de un solo producto, cada producto en el SPL se beneficiarse del defecto fi x. Estos factores permiten más rápidas mejoras a la calidad del producto en el desarrollo de SPL.

Hay de segundo orden beneficios adicionales para el desarrollo de SPL. Por ejemplo, el desarrollo SPL proporciona a las organizaciones un camino claro lo que les permite convertir el trabajo de proyectos a medida para clientes especí fi cas en la línea de productos cuenta reutilizado en todo el SPL. Cuando las organizaciones tienen procesos para reutilizar los activos administrados, el desarrollo de fi co trabajo del proyecto en el cliente especifica inicialmente puede ser administrado en un activo personalizado. Si las características demuestran tener mayor significación, el activo personalizado se puede mover en la base principal activo reutilizado.

Otro beneficio relacionado es que la gestión de los activos principales y personalizados ofrece una visión clara y sencilla de la gama de productos mantenidos por la organización. Este punto de vista permite a las organizaciones a más fácilmente:

- Actualizar los productos a utilizar una nueva versión de la base
- Ver qué activos son fundamentales para el negocio
- Ver cómo los productos se diferencian el uno del otro
- Considerar las opciones de funcionalidad futuro para el SPL

15.2.2 líneas de productos para ICDE

Los tres productos ICDE planificadas todos funcionan de una manera similar y las diferencias para cada uno de los productos están bastante bien comprendidos. El producto Gobierno tendrá una interfaz de usuario que soporta listas de control de política y de gobierno, el producto Finanzas apoyará muestra continuamente actualizadas de información de mercado en tiempo real, y el producto de Inteligencia integrará vistas de datos de diversas fuentes de datos Clasi fi cado.

La variación requerida en la línea de productos puede ser definido en gran medida en términos de los componentes de recogida de datos. Las opciones de la GUI y el acceso a fuentes de datos especí fi cos de dominio tendrán que ser apoyado por la variación en los puntos de recogida de los componentes. Esto significa que el Recopilación de datos componente cliente necesitará puntos de variación con el fin de facilitar el acceso a fuentes de datos fi cas de dominio-específico de aplicación. Esto requerirá componentes personalizados para manejar los detalles especí fi cos de cada una de las nuevas gobierno / fi nanciera fuentes de datos / de inteligencia. los Almacén de datos componente no debería tener que soportar cualquier variación para los tres productos diferentes. Debe ser apto para ser reutilizado como un simple activo básico.

15.3 Línea de productos Arquitectura

desarrollo SPL se describe generalmente como haciendo uso de una línea de Arquitectura del producto (PLA). Un PLA es una arquitectura orientada a reutilizar-para los activos principales en el SPL. Los objetivos de reutilización y variación de un PLA son los siguientes:

- Sistemáticamente apoyar un ámbito de aplicación planificada de antemano de la funcionalidad variante
- Permitir que los productos dentro del SPL a elegir opciones de entre las que la variante funcionalidad

Un PLA logra estos objetivos utilizando una variedad de mecanismos técnicos para la reutilización y la variación que se describen en las siguientes secciones. jan Bosch [2] ha identi fi cado tres niveles de madurez PLA:

1. Bajo especí arquitectura fi ed (variación ad-hoc)
2. Arquitectura fi cado
3. Arquitectura forzada (todo variación requerida apoyo de puntos de variación arquitectónicos planificados)

El aumento de los niveles de madurez de arquitectura proporcionan más beneficios a partir de la variación sistemática al hacer el desarrollo de productos más rápido y más barato. Sin embargo, cada vez más maduros PLA ofrecen menos oportunidades para la variación ad-hoc, lo que puede reducir las posibilidades de reutilización. Sin embargo, los crecientes niveles de reutilización pueden lograrse si hay una mejor variación sistemática, es decir, una mejor adaptación de la PLA al dominio de alcance y la aplicación de la SPL.

APLA no siempre es necesaria para el desarrollo exitoso de SPL. Lo menos madura de los niveles de madurez de Bosch es “underspeci arquitectura fi cado”, y se han reportado experiencias de la adopción de desarrollo SPL con un extremadamente underspeci fi cada PLA. Aunque los productos en un SPL siempre tendrán algún tipo de arquitectura, que no necesariamente tiene que ser un PLA, es decir, uno diseñado para apoyar los objetivos de reutilización y la variación. En esencia, la reutilización de software, los desarrolladores deben:

1. Encontrar y entender el software
2. Hacer que el software disponible para su uso por incorporarlo en su contexto de desarrollo
3. Utilice el software mediante la invocación se

de Software Producto Línea (San Diego, CA, EE.UU., Agosto 19-22, 2002). Springer LNCS vol. 2379, 2002, pp. 257-271.

Veamos cada uno de estos pasos en turno.

15.3.1 encontrar y entender software

Los ingenieros de software usan documentación de la API y los manuales de referencia para apoyar la sencilla reutilización de bibliotecas de software. Para el desarrollo de SPL, las directrices de práctica la línea de productos de la SEI (ver lecturas adicionales) describen el Producto del patrón de piezas que aborda el descubrimiento y la comprensión de software activo fundamental para el desarrollo de SPL. Este patrón se basa en la documentación de los procedimientos para utilizar y crear instancias activos principales en la construcción de productos.

---

[2] J. Bosch, La madurez y la Evolución en Líneas de Producto Software. En Actas de la Segunda Conferencia Internacional

15.3.2 Llevar software en el contexto de desarrollo

Después hallazgo del software, un desarrollador tiene que hacer que esté disponible para ser utilizado. Hay muchas formas de llevar el software en un contexto de desarrollo, que pueden clasificarse en función de su “tiempo de unión.” Este es el momento en el que los nombres de los activos de software reutilizados se unen a una aplicación específica. Los principales veces vinculantes y algunos mecanismos de ejemplo son:

- el tiempo de programación - por el control de versiones de código fuente
- El tiempo de construcción - por el control de versiones de las bibliotecas estáticas
- tiempo de enlace - por el sistema operativo o el soporte de la máquina virtual para bibliotecas dinámicas
- Tiempo de ejecución - por mecanismos fi middleware o aplicaciones especí para con fi guración

o plug-ins, y por la programación mecanismos del lenguaje dinámico para la reflexión

A principios de los tiempos de enlace (tales como la programación o el tiempo de construcción) que sea más fácil de usar variación ad-hoc. veces (como enlace o el tiempo de ejecución) después ligarlos retrasar compromiso con variantes específicas, y así hacer más fácil para beneficiarse de las opciones proporcionadas por la variación sistemática. PLA cada vez más maduro para el desarrollo SPL tienden a utilizar más adelante mecanismos de tiempo de unión. Esto les permite maximizar los beneficios de un ámbito de SPL que se entiende bien y tiene un buen fi cio con los mercados de interés de la compañía.

15.3.3 Software de invocación

Para invocar el software, lenguajes de programación proporcionan mecanismos de procedimiento / función / llamada al método. Para sistemas distribuidos, los estándares de interoperabilidad como CORBA y SOAP proporcionan mecanismos de invocación remotos que están vinculados a los mecanismos del lenguaje de programación, para permitir a los desarrolladores de software en ejecución invocan los sistemas en otras máquinas. Estos mecanismos de invocación son los mismos para el desarrollo SPL como para el desarrollo de software tradicional.

Gestión fi guración 15.3.4 Software Con para reutilización

Para las organizaciones que están adoptando el desarrollo SPL, los tiempos de unión más comunes para la reutilización están programando tiempo y construyen tiempo. Esto hace que el software de gestión con fi guración (SMC) un área crítica proceso de apoyo para el desarrollo de SPL. SMC incluye control de versiones y control de cambios de los activos de software.

SCM para el desarrollo SPL es más complicado que en el desarrollo de producto normal en parte porque con fi guración identi fi cación (CI) es más complicado. CI es la actividad SCM de especificar los nombres, atributos y relaciones entre configuraciones fi (una colección versionado de objetos versionados). En producto normal

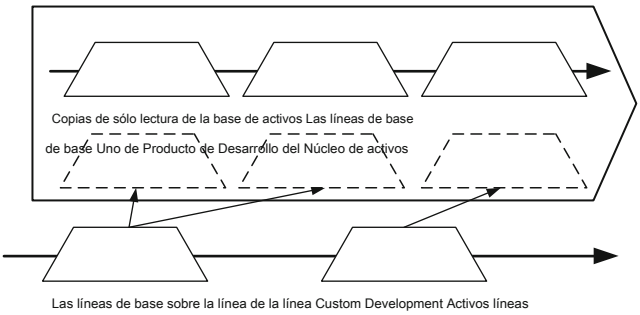


Fig. 15.4 Un patrón de ramificación SMC para el desarrollo SPL 226

desarrollo, con fi guración de un producto por lo general tiene una estructura simple (por ejemplo, un único binario versionado o versionado fi le sistema de jerarquía de directorios). Sin embargo, en el desarrollo de SPL, cada núcleo activo, activo personalizado, y el producto es una configuración con fi que deben ser identificados y las relaciones entre estas configuraciones Fi debe estar especi fi y gestionado. Básicamente, SMC se hace mucho más arquitectónica para el desarrollo de SPL.

Un enfoque para SCM para el desarrollo SPL se representa en la Fig. 15.4 . En esto enfoque, los activos y los productos de la base tienen cada uno su propia línea de desarrollo (LOD). Cada versión del producto incluye sus propios activos personalizados, así como versiones de los principales activos. El sistema de control de versiones asegura que los activos centrales reutilizados son de sólo lectura para un producto, y que no se modi fi cado exclusivamente en el contexto de un LOD fi co del producto específico. Sin embargo, LOD de un producto puede tener una versión posterior de un activo núcleo que se ha producido en su propio límite de detección.

Esta visión del desarrollo SPL proporciona una base cuantitativa para ver por qué el desarrollo SPL puede resultar tan eficaz. El LOD para cada producto contiene el código fuente para los activos fi c-cliente específico y también (de sólo lectura) el código fuente para los activos principales. Por lo que cada LOD contiene esencialmente el mismo código fuente ya que eran la línea de productos no enfoques utilizados. Sin embargo, el volumen total de código ramificada se ha reducido, debido a que el tamaño de los activos básicos no se multiplica a través de cada producto. activos centrales no están ramificados para cada producto, y el diseño tan bajo nivel, los gastos de codificación y los exámenes de unidad dentro de los principales activos pueden ser compartidos a través de muchos productos.

En el ejemplo ICDE hay tres productos, y vamos a suponer que los componentes de la base tiene 140.000 LOC (líneas de código) y la parte personalizada de cada producto tiene 10.000 LOC. En el desarrollo normal del producto, cada producto se mantendría en un nivel de detalle independiente, dando un total de:

re 140; 000 þ 10; 000 Th? 3 ¼ 450; 000 ramificados LOC:

En el desarrollo de SPL, el núcleo está en su propio nivel de detalle, y cada producto tiene un nivel de detalle sólo para cambiar sus activos personalizados, dando un total de:

140; 000 þ ð 10; 000 3 Þ ¼ 170; 000 ramificados LOC:

Eso es sólo el 38% del total original. La mejora se pone mejor cuando se desarrollan más productos, o cuando el tamaño de los activos personalizados en comparación con los principales activos es proporcionalmente menor.

15.4 Mecanismos de variación

En un SPL, los activos núcleo de soporte funcionalidad variable de al proporcionar puntos de variación. Un PLA típicamente usa específico c mecanismos de variación de arquitectura para implementar la funcionalidad variable. Sin embargo, un SPL también puede utilizar los mecanismos de variación nonarchitectural para variar la funcionalidad del software.

Además de los mecanismos de variación de nivel de arquitectura, hay diseño de nivel y los mecanismos de variación de nivel de fuente. Estos diferentes tipos de variación no son incompatibles. Por ejemplo, es posible utilizar la variación fi le-nivel, al mismo tiempo como la variación de la arquitectura. En esta sección se describen algunos de los mecanismos de variación en estos diferentes niveles de abstracción. Esta clasificacion es similar a la taxonomía de las técnicas de la variabilidad de realización en términos de entidades de software que ha sido propuesto por Svahnberg et al. [3]

15.4.1 Puntos Arquitectura-nivel de variación

mecanismos de variación arquitectónicos son estrategias de diseño de alto nivel destinadas a permitir que los sistemas de apoyo a una amplia gama de funcionalidad. Estas estrategias son de sólo muy débilmente relacionados con las instalaciones de cualquier lenguaje de programación fi co. Ejemplos de estos incluyen los marcos y arquitecturas enchufables. Incluso el reconocimiento formal de un espacio de opciones de con fi guración o parámetros para la selección entre la funcionalidad variante puede ser considerada como un mecanismo de variación de la arquitectura.

15.4.2 Diseño Nivel Variación

El límite entre la arquitectura y el diseño no es siempre una clara. Aquí vamos a decir que los mecanismos a nivel de diseño son los soportados directamente por las instalaciones de lenguaje de programación y que los mecanismos a nivel de la arquitectura debe ser creado por la programación. mecanismos del lenguaje de programación se pueden utilizar para representar la variación. Estos mecanismos incluyen interfaces de componentes que pueden permitir diferentes implementaciones funcionalmente diferentes, y la herencia y de la anulación que de igual forma permitir que los objetos tienen funcionalidad variante que satisface las clases base fi ca.

---

[3] M. Svahnberg, J. van Gurp, J. Bosch, Una taxonomía de las técnicas de realización de variabilidad, documento técnico, Blekinge Institute of Technology, Suecia, 2002.

15.4.3 Variación de nivel de archivo

entornos de desarrollo y lenguajes de programación proporcionan formas de implementar la variación en el nivel de código fuente archivos. Algunos lenguajes de programación proporcionan la compilación condicional o mecanismos de macros que pueden poner en práctica la variación funcional. En cualquier caso, construir secuencias de comandos pueden realizar variaciones le lógica o fi físico que puede ser usado para representar la variación funcional.

marco, será mucho más fácil al 228

15.4.4 Modificación mediante Gestión de configuración del software Con fi

El papel principal de SMC para el desarrollo de la línea de productos es apoyar la reutilización de activos mediante la identificación y la gestión de las versiones de (y) los cambios en los productos y sus componentes activos constituyentes. Las nuevas versiones de productos no tienen que usar la versión más reciente de un activo núcleo, sistemas SCM pueden permitir que un producto de

marco GUI para hacerlas añadir nuevos tipos de paneles "plug-in" conectados a fuentes de datos. Cuando se extiende esto

usar cualquier versión principal activo que cumpla con las necesidades de los interesados del producto. El historial de versiones y la versión de ramificación espacio dentro de una herramienta de SMC se pueden utilizar para representar la variación.

En una herramienta de control de versiones, un LOD ramificada de un activo núcleo puede ser creado para contener funcionalidad variante. Ramificaciones principales activos reutilizados con el fin de introducir variaciones en curso es una especie de decadencia técnica que reduce los beneficios del desarrollo SPL. En el caso extremo en que cada producto tiene su propia rama de activos centrales, una organización va a haber anulado el desarrollo SPL por

funcionalidades. La interfaz gráfica de usuario es actualmente demasiado rígida, por lo que el equipo tiene previsto ampliar el completo y va a hacer volver el desarrollo de productos ordinarios. No obstante, en algunas circunstancias, una rama temporal es la forma más pragmática para introducir una variación en un componente en la cara de un plazo de entrega que se avecina.

15.4.5 Arquitectura del producto Línea de ICDE

tiempo real de información de mercado para el producto financiero, los componentes GUI existentes necesitan nuevas

Desde el principio en el desarrollo del producto ICDE el equipo de desarrollo ha hecho un esfuerzo considerable en la arquitectura del producto. Esto significa que están en la posición afortunada de tener ya muchos mecanismos de variación de arquitectura en lugar, por lo que la adopción de desarrollo de la línea de productos más fácil. Por ejemplo, el

Fuente de datos mecanismo adaptador proporciona toda la variabilidad requerida para los tres nuevos productos. Estos mecanismos de variación existentes forman el corazón de la arquitectura de línea de producto para la línea de productos ICDE.

El equipo necesita para definir algunos de los nuevos mecanismos de variación también. Para apoyar la visualización en

implementar el panel de visualización en tiempo real, conectarlo a la fuente de datos de mercado, e incluirlo en la interfaz gráfica de usuario para la compilación del producto financiero.

Sin embargo, aunque el equipo pensó que el ICDE Almacén de datos sería el mismo para los tres productos, resulta que separa los datos de Clasi fi cado por el producto de seguridad es un problema trivial, con requisitos muy diferentes de los otros dos productos. El equipo tiene que llegar a algún destino especial Almacén de datos código sólo para ese producto. La manera más fácil de hacer estos cambios especiales se encuentra en una copia separada del código, por lo que en su herramienta de control de versiones que crear una rama de la Almacén de datos componente sólo por el producto de seguridad. Tener que mantener dos implementaciones diferentes de la Almacén de datos puede doler un poco, pero es lo mejor que el equipo puede hacer bajo un plazo muy corto. Una vez que se envía el producto que tendrán tiempo para diseñar un mecanismo de variación arquitectónica mejor para la próxima versión, y se mueven todos los productos sobre los que el nuevo Almacén de datos componente.

15.5 La adopción de software de desarrollo de la línea de productos

Al igual que muchos cambios radicales de negocios, la adopción de desarrollo SPL en una organización es a menudo impulsada en respuesta a una crisis (lo Schmid y Verlage [4] diplomáticamente una situación llamada "reingeniería-driven"). Esto puede ser una demanda urgente de desarrollar rápidamente muchos productos nuevos, o para reducir los costes de desarrollo, o para escalar el desarrollo de funciones nuevas en la cara de una creciente carga de mantenimiento. Esta sección señala algunos caminos y procesos pertinentes a la adopción del desarrollo SPL.

Hay dos puntos de partida diferentes en la adopción de desarrollo SPL:

1. Campos verdes: donde inicialmente no existen productos
2. Los campos arados: donde una colección de antiguos productos relacionados ya tienen ha desarrollado sin reutilización en mente

Cada situación tiene consideraciones especiales, como se describe a continuación. Para la adopción campos verdes de líneas de productos, la SEI Lo que hay que construir patrón es particularmente relevante. Este patrón se describe cómo una serie de áreas de práctica en interacción puede dar lugar a la generación de un ámbito SPL (SPL saber lo que se construirá) y un caso de negocio (a saber por qué la construcción de la SPL es una buena inversión para la organización). Esto es La determinación del alcance y La construcción de un Business Case áreas de práctica que son directamente responsables de estas salidas son compatibles con el Descripción de dominios relevantes, Análisis de mercado, y Predicción tecnología Areas de práctica.

Una organización tiene que decidir sobre sus mercados de interés, su alcance término medio SPL-Tolong, y sus planes de producción de productos de corto y medio plazo. La organización debe planificar y evaluar las distintas opciones de inversión de tener el PLA de la base del núcleo activo apoyar un ámbito SPL suficientemente grande. Esto hace que sea

---

[4] K. Schmid, M. Verlage, El impacto económico de la adopción línea de productos y Evolución. En IEEE Software, julio / agosto de 2002, pp. 50-57.

posible que el comercio fuera de las posibilidades de regreso de los productos que se pueden generar dentro de ese ámbito de los mercados de interés para la organización.

Invertir en un PLA al comienzo de una SPL proporcionará un mejor rendimiento a largo plazo suponiendo que los productos de la SPL tienen éxito en el mercado. Sin embargo, el costo y dif fi cultad técnica de crear una PLA tales ex nihlio puede suponer un obstáculo para la adopción de desarrollo SPL, sobre todo si la organización no está ya expertos en el dominio de aplicación está dirigida por el SPL.

Por el contrario, cuando existe un conjunto de productos y se realiza la migración a un SPL, una organización, como para la adopción Green Fields, tendrá que decidir sobre el alcance SPL y mercados de interés para el SPL. Sin embargo, las organizaciones en esta posición por lo general ya tienen un buen conocimiento acerca de estos. El alcance de la SPL en gran medida impulsado por la funcionalidad de los productos existentes y futuros planes de producto. Las otras consideraciones signi fi cativas para la adopción de los campos arados son posibles barreras relacionadas con el control de cambios y de fi nición de los principales activos y PLA.

problemas de control de cambios pueden suponer un obstáculo para la adopción de desarrollo SPL para los productos heredados de una organización. Los grupos de interés de los productos existentes que ya se han establecido las expectativas acerca de cómo cambian sus versiones de productos. Como se discutió en la sección de SMC, cada producto de la SPL tiene actores que influyen en los cambios realizados en los principales activos, y estos cambios en los activos principales en el SPL afectarán en última instancia, todos los productos de la SPL, incluyendo otras partes interesadas. Este cambio en la naturaleza de las versiones de los productos debe ser entendido y aceptado por las partes interesadas de los productos.

Cuando se proceda a la de fi nición de un SPL para un conjunto existente de productos independientes, la organización debe decidir lo que es central para cada producto, y lo que es habitual o una especí fi ca a cualquier producto individual. En lugar de tirar los activos existentes para los productos de la organización y partiendo de una pizarra en blanco, es posible utilizar un método de extracción de los principales activos de la mina de los productos existentes. La SEI describe una línea del área de práctica producto Minería activos existentes hacer frente a esta actividad. En muchos sentidos, la extracción de los principales activos es como un ejercicio de refactorización gigante, como se representa en la Fig. 15.5 . A partir de una colección inicial de productos, el objetivo de

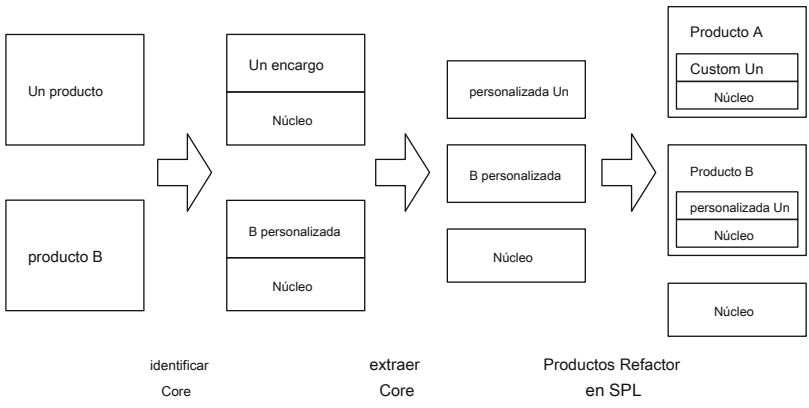


Fig. 15.5 Minería activos principales de una colección de productos existentes 230

el ejercicio es al fi nal con productos idénticos, excepto que ahora todos construidos usando un activo básico común.

Cuando de fi nición de los activos principales, la organización puede también definen una PLA para atender a la variación que se identi fi cada uno de los productos. Svahnberg et al. han presentado una serie de pasos mínimamente necesarias para introducir variabilidad en un SPL. Estos son:

- La identificación de la variabilidad
- La restricción de la variabilidad
- La implementación de la variabilidad
- La gestión de la variabilidad

Con el fin de reducir el control de cambio de conflictos, puede ser más fácil de introducir el desarrollo SPL temprano en el ciclo que conduce a la liberación de una nueva versión de un producto. actores de productos se preparan para los cambios importantes cuando se recibe una nueva versión principal. Aunque logro de un desarrollo SPL tiene por qué no en consecuencia principio de cualquier diferencia funcional a un producto, hay al menos será de control de cambios modi fi caciones de política, que los clientes pueden hallar más fácil de aceptar en el contexto de una importante nueva versión del producto.

Una organización adopción de líneas de productos también pueden reducir los riesgos comerciales y técnicos de forma incremental por el despliegue de la presión sonora dentro de la organización. La adopción puede ser incremental ya sea aumentando progresivamente el tamaño de los activos principales, añadiendo progresivamente más productos a utilizar los principales activos, o una combinación de ambos.

Áreas de Práctica Adopción 15.5.1 Línea de productos

La adopción de desarrollo SPL tiene un impacto fuera del contexto de desarrollo técnico. Independientemente del punto de partida para la adopción línea de productos (verde o los campos arados) y con independencia del producto fi co y cambios en los procesos técnicos que se van a realizar, muchas cuestiones de gestión de la organización deben ser tratados con éxito la transición al desarrollo SPL. Las guías de práctica línea de productos describen el SEI Patrón de arranque en frío que agrupa a la práctica de las áreas que pueden ayudar a una organización prepararse eficazmente para el lanzamiento de su primer SPL. La estructura del modelo se muestra en la Fig. 15.6 .

Aunque los detalles de estas áreas de práctica están más allá del alcance de este capítulo, el patrón en su conjunto destaca el hecho de que el desarrollo SPL debe tener el apoyo de grandes sectores de dentro de la organización y la adopción de sus clientes.

Adopción 15.5.2 Línea de Producto de ICDE

El equipo ICDE fue conducido al desarrollo de SPL por la desalentadora perspectiva de desarrollar tres nuevos productos a la vez. Están creando tres nuevos productos para los tres mercados fi cas, pero están utilizando su producto existente como punto de partida.

Fig. 15.6 La estructura de áreas de práctica en línea de productos de SEI Inicio fresco patrón (después de Clements y Northrup 2002, P383)

Su adopción del desarrollo SPL es, pues, un escenario Campo arado. Ellos tienen que extraer los componentes reutilizables de su base de código existente.

Por suerte sus clientes existentes no van a estar demasiado preocupado inicialmente sobre el paso a un PLA, debido a que el movimiento es parte del desarrollo de una nueva versión del producto. Los clientes estarán encantados de actualizar debido a las nuevas características que también va a estar recibiendo.

15.6 Software curso Desarrollo Línea de Producto

SPL desarrollo debe ser eficaz no sólo para el desarrollo inicial de los nuevos productos, sino también por su continuo mantenimiento y mejora. Aunque el desarrollo de SPL puede tener muchos beneficios, es más complicado que el desarrollo normal del producto. procesos mejorados son necesarios para que el desarrollo continuo SPL eficaz. En esta sección se ofrece un resumen de algunos de estos procesos de desarrollo de SPL. Prestamos especial atención a “control de cambios” y “evolución arquitectónica” SPL para el desarrollo, sino también un resumen de otras áreas SEI Línea de productos de práctica para el desarrollo de SPL en curso.

15.6.1 Control de Cambios

control de cambios de software se relaciona con la gestión de con fi guración de software, y se ocupa de la planificación, coordinación, el seguimiento y la gestión de los efectos del cambio de artefactos de software (por ejemplo, el código fuente). El control de cambios es más difícil cuando se hace la reutilización del software, y esto afecta el desarrollo del SPL.

En cualquier tipo de desarrollo de productos, cada producto tiene una colección de grupos de interés que se ocupa de cómo sus cambios de productos para adaptarse a sus necesidades de nuevas funcionalidades. Además, las partes interesadas están preocupados por características no funcionales (tales como calendario de lanzamiento, la fiabilidad del producto) en relación con la liberación de sus productos. las partes interesadas con aversión al riesgo (como los que utilizan el software safetycritical o aquellos en el sector bancario) están motivados a menudo para asegurarse de que sus productos no cambian en absoluto! Dichas partes interesadas a veces prefieren estar con fi anza en su comprensión del producto (bugs y todo) en lugar de utilizar las nuevas versiones, quizás mejor.

El control de cambios es más difícil cuando se hace la reutilización de software, incluyendo la reutilización de software para el desarrollo de SPL. Para el desarrollo de productos ordinario, cada producto es desarrollado por separado, por lo que las partes interesadas de cada producto se mantienen separados también. Sin embargo, en el desarrollo SPL cada producto depende de los principales activos reutilizados, y así las partes interesadas de estos productos también dependen indirectamente de estos activos centrales reutilizados. Si el cliente de un producto tiene una solicitud de cambio que implica un cambio a un activo núcleo, a continuación, la aplicación que obligará a que el cambio en cualquier otro cliente que utiliza la nueva versión de ese activo núcleo. Los muchos, a menudo conflictivos, necesidades de los interesados de los productos tendrá que ser al mismo tiempo satisface fi cados por los principales activos reutilizados.

15.6.2 Evolución de Arquitectura para el Desarrollo SPL

En el desarrollo de SPL hay una evolución constante de ambos activos personalizados de productos individuales y los principales activos reutilizados. El PLA es la base arquitectónica para la variación con el apoyo de los principales activos. Un cambio a la interfaz de un núcleo activo es un cambio en el PLA, y puede forzar cambios en todos los productos que utilizan la nueva versión de estos activos principales. ¿Cómo entonces se deben agregar las nuevas o mejoradas características básicas de una línea de productos? Es decir, cómo se deben hacer cambios a la PLA?

Hay tres maneras de tiempo, la introducción de puntos de variación en los activos centrales:

- **Proactivo: Planificar el futuro para futuras características, y ponerlas en práctica en los principales activos** antes de que cualquier producto que necesita.
- **Reactivo: Espere hasta que una nueva característica que realmente se requiere de un producto, y luego** implementarlo en los principales activos en ese momento.
- **Retroactivo: Espere hasta que una nueva característica que realmente se requiere de un producto, y luego** implementarlo en un activo personalizado en ese momento. Cuando suficientes productos implementar la función en sus activos personalizados, agregarlo a los principales activos. Los nuevos productos se pueden utilizar característica, y los productos más antiguos pueden caer su aplicación activo personalizado a favor de los activos principales de los nuevos activos centrales aplicación.

Es posible utilizar una combinación de estos enfoques, para diferentes mejoras. Por ejemplo, las mejoras en un largo plazo Hoja de Ruta se podría añadir de una manera proactiva, mediante la planificación de cambios de arquitectura para apoyar la futura ampliación del alcance del SPL. mejoras limitadas pero generalmente útiles a los activos centrales podrían añadirse de manera reactiva, mediante la modificación del PLA como es requerido por esas mejoras.

Tabla 15.1 La comparación de estrategias para la evolución de la arquitectura

	Proactivo	Reactivo	Retroactivo
Sin inversión a largo plazo	No	Sí	Sí
Reduce el riesgo de cambio de núcleo activo conflicto	Sí	No	Sí
Reduce el tiempo de espera para agregar función de primer producto fi Sí		No	No
Reduce el riesgo no se requiere de la función de núcleo en una serie de productos	No (0 productos)	No (1) de productos	Sí

Mejoras que necesita un producto que son más especulativo o menos bien de fi nido podrían añadirse en forma retroactiva.

Cada una de estas estrategias tiene diferentes costos, beneficios y riesgos. La elección de la estrategia para una característica particular será impulsado por la consideración de estas soluciones de compromiso en el contexto empresarial de la organización. Mesa 15.1 resume algunas de las diferencias entre los tres enfoques:

Áreas 15.6.3 Línea de Producto práctica del desarrollo

Las guías de práctica línea de productos proporcionan la SEI Fábrica patrón que conecta entre sí a otros patrones y sus áreas de práctica constituyentes relevantes para el desarrollo y el mantenimiento continuo de un SPL. los En movimiento grupos de patrones participen las áreas de práctica de gestión de la organización. Otros patrones SEI relevantes son la
Monitor, Proceso, y Plan de estudios patrones que describen aspectos del desarrollo en curso de SPL.

Por áreas de práctica técnicas, del SEI cada Activo patrón describe áreas de práctica que son relevantes para el desarrollo de los principales activos. los Piezas del producto patrón une los principales activos con el desarrollo de productos. los Constructor producto
patrón describe áreas de práctica relevantes para el desarrollo de cualquier producto específico. los Linea de ensamblaje patrón describe cómo los productos son emitidas desde el SPL.

15.6.4 líneas de productos con ICDE

Hacer el desarrollo SPL no era sólo una cuestión de arquitectura para el equipo ICDE. Cada uno de los productos que tenían un grupo de dirección del cliente que estuvo involucrado en los requisitos de fi nición de los nuevos productos y de fi nidas las solicitudes de mejora que querían realizar un seguimiento hasta la entrega de los productos. Pero el equipo ICDE no quería que el grupo de dirección del cliente de productos financieros para ver todos los detalles del grupo de dirección de productos de seguridad, y viceversa. Los problemwas que algunas solicitudes de mejora eran la misma (o similar), y el equipo no quieren confundirse acerca de peticiones duplicadas cuando empezaron codificación.

Por lo tanto, el equipo ICDE configurar diferentes sistemas de solicitud de cara al cliente para cada uno de los productos. Estos vinculados a un sistema de solicitud de cambio interno que podría realizar un seguimiento de los cambios en cada uno de los principales subsistemas reutilizados y también los componentes personalizados fi c-de productos específico.

Eventualmente, el primer producto fue puesto en libertad. En lugar de liberar los tres productos a la vez, el equipo envía el primer producto más simple ficción, es decir, el producto de Gobierno. Los clientes gobierno elevó rápidamente unos informes de defectos posteriores a la liberación, pero el desarrollo del ICDE teamwas capaz de responder **rápidamente. La buena noticia fue que uno de los defectos que era fijo fue en el núcleo Recopilación de datos componente,** por lo que cuando los otros dos productos fueron puestos en libertad más tarde, sus clientes no vería ese problema. El equipo ICDE estaba empezando a ver algunos bene fi cios de la calidad de desarrollo SPL.

La mala noticia llegó después de los demás productos fueron puestos en libertad. Los clientes de seguridad y financieros estaban felices de tener la nueva versión, aunque los clientes financieros hicieron levantar un informe sobre el **defecto Análisis de los datos componente. Hubiera sido fácil de fi x en el componente subyacente, pero en ese momento los clientes del gobierno habían entrado en producción. No se habían visto ese problema en el Análisis de los datos zona,** y de hecho, el error se relaciona con las extensiones marco necesario para soportar el panel de visualización en tiempo real de productos financieros.

Sin embargo, si el Análisis de los datos componente cambia de ninguna manera en absoluto, los clientes gubernamentales tendrían que seguir su política y vuelva a ejecutar todas las pruebas de aceptación relacionados, que les costaría tiempo y dinero. Así que realmente no quieren ver ningún cambio, y ejercer presión sobre el equipo de ventas ICDE para tratar de detener el cambio.

El equipo de desarrollo del ICDE realmente quería cambiar la versión de la base, pero como no podía satisfacer a todos ellos? Pensaron en falsificar los cambios fundamentales en activos a medida solo para el producto **financiero, pero al final decidieron mantener el producto Gobierno de la versión antigua de la Análisis de los datos componente,** e implementó el fi x en el núcleo. El equipo de desarrollo del ICDE también creó un BCC Core participación de miembros representativos de cada uno de los tres grupos de dirección del cliente. Esto significaba que en el futuro las negociaciones podrían gestionarse dentro del núcleo CCB, en lugar de a través del equipo de ventas de la ICDE.

Un punto brillante en el horizonte era que los clientes de seguridad estaban empezando a hablar de su necesidad de ver la visualización en tiempo real de las noticias. El equipo de desarrollo podría implementar ICDE que simplemente reutilizando el panel de visualización en tiempo real desarrollado por el producto financiero. La compañía ya había contabilizado los costes de desarrollo de esa característica, por lo que ser capaz de vender de nuevo para otros clientes que significaría todos los nuevos ingresos irían directamente a la línea de fondo.

15.7 Conclusiones

el desarrollo de la línea de productos ya ha dado muchas organizaciones órdenes de magnitud de las mejoras a la productividad y el tiempo de comercialización, y signi fi cativas mejoras en la calidad del producto. Si pensamos en el desarrollo de una SPL simplemente SMC

perspectiva, podemos ver que (proporcionalmente grandes) activos centrales no están ramificados para cada producto, por lo que el número total de líneas ramificadas de código se reduce enormemente para toda la SPL.

¿Qué depara el futuro para el desarrollo SPL? Debido a su enorme potencial, es probable que se convierta aún más ampliamente conocido, mejor entendida, y se utiliza cada vez más el desarrollo de SPL. Sin embargo, el desarrollo de SPL también tendrá un impacto en estudios de arquitectura de software, como mecanismos de arquitectura para su reutilización en la gran ser mejores y más ampliamente entendido.

prácticas arquitectónicas mejoradas combinadas con una comprensión más profunda de los dominios de aplicación específica fi también pueden apoyar los mecanismos de variación cada vez declarativas. Esto podría transformar la reutilización del software a ser más como la visión mítica de construcción de software utilizando bloques de construcción de software. reutilización sencilla se basa en gran medida de la variación de procedimiento, la escritura de código ad hoc para lograr la funcionalidad particular que se requiere. El aumento de la sofisticación arquitectónica y el conocimiento de dominio puede apoyar fi gurable con variación, realizado por la variación sistemática con el apoyo de interfaces principales activos.

La elección de una variante de este sistema requiere la elección de valores de una lista de opciones de con fi guración. Cuando un dominio de aplicación está muy bien entendida, a continuación, un lenguaje de dominio específico convierte en una forma viable de especificar declarativa variación del producto. Frases en este idioma pueden especificar variantes del sistema, y pueden ser interpretadas de forma dinámica por los activos principales.

Otros enfoques de diseño y arquitectura, como la programación orientada a aspectos y desarrollo dirigido por modelos también tienen promesa como mecanismos de variación o masa de personalización que pueden ser capaces de apoyar el desarrollo de SPL.

A medida que el tiempo de variación del sistema se extiende fuera del contexto del desarrollo, también lo hace la necesidad de ampliar el control y la gestión de variación. Para los sistemas que pueden variar en el tiempo de instalación, el tiempo de carga, o de tiempo de ejecución, la necesidad de controlar y gestionar variación del sistema no termina cuando el sistema se libera de desarrollo. con el software de gestión fi guración compatible con el control y la gestión de variación durante el desarrollo. Sin embargo, para la instalación, carga o tiempo de ejecución, los marcos de gestión de paquetes y gestión de aplicaciones existentes tienen instalaciones muy débiles para la versión y el control de la variación. En el futuro, los límites entre la gestión de con fi guración, la gestión de paquetes, y la gestión de aplicaciones se verá borrosa. por lo tanto, se requiere un marco unificado para controlar y gestionar variación a través de todo el ciclo de vida del producto.

15.8 Lectura adicional

El Instituto de Ingeniería de Software ha sido un líder en la de fi nición y reportar el uso de las líneas de productos de software. Una excelente fuente de información es el siguiente libro por dos de los pioneros del campo:

P. Clements, L. Northrop. Las líneas de producto de software: Prácticas y patrones.
Addison Wesley, 2001. 236

El sitio web del SEI también contiene mucha información valiosa y enlaces a otras fuentes relacionadas con la línea de productos:

http://www.sei.cmu.edu/productlines/

Otras referencias son excelentes:

Klaus Pohl, G € unter B € ockle, Frank J. van der Linden, la línea de productos de software Ingeniería: Fundamentos, Principios y Técnicas, Springer-Verlag 2010

Frank J. van der Linden, Klaus Schmid, Eelco Rommes, Líneas de Producto Software en Acción: La Práctica Mejor industrial en la línea de productos de ingeniería, Springer-2007.

con el software de gestión fi guración es una parte clave de las líneas de productos de software. Un buen libro sobre este tema es:

SP Berczuk, B. Appleton. Software con fi guración de las pautas de gestión: el trabajo en equipo, la integración práctica. Addison-Wesley, 2002.

Un estudio de casos que describen cómo explotar la variación basada en le fi para crear una línea de productos de software es:

M. Staples, D. Hill. La adopción de las experiencias de desarrollo de software Línea de Producto sin línea de productos de Arquitectura. Actas de la 11ª fi Asia y el Pacífico Conferencia de Ingeniería de Software (APSEC 2004), Busan, Corea del Sur 30 Nov - 3 dic 2004, por IEEE, pp 176-183..

Una perspectiva ligeramente diferente en las líneas de productos son las fábricas de software trabajan por Jack verde campo et al. Este libro es de fi nitivamente la pena leer.

J. Green campo, K. corto, S. Cook, S. Kent, J. Crupi, Software Factories: Ensamblaje de aplicaciones con los patrones, modelos, marcos y herramientas, Wiley 2004.

Índice

I. Gorton, Arquitectura de Software esencial,
DOI 10.1007 / 978-3-642-19176-3, # Springer-Verlag Berlin Heidelberg 2011