Métodos y Herramientas para el Desarrollo de Aplicaciones

Actas del Taller de Trabajo ZOCO

R. Corchuelo, D. Ruiz y J.L. Arjona
(Editores)

Zoco es un taller de trabajo organizado periódicamente por el Grupo de Sistemas Distribuidos de la Universidad de Sevilla con el propósito de dar cabida en él a investigadores y profesionales que trabajan en métodos y herramientas para el desarrollo de aplicaciones, con un marcado énfasis en la Web.

En este libro se recogen las actas de dicho taller, que fue celebrado en el contexto de las Jornadas de Ingeniería del Software y Bases de Datos en Octubre de 2006 en Sitges. Podrá encontrar información adicional sobre estas y otras ediciones en la dirección http://www.tdg-seville.info/cfp/Zoco.

JORNADAS DE
Ingeniería del Software y Bases de Datos
Sitges, 3 al 6 de Octubre de 2006

mec.es

Métodos y Herramientas para el Desarrollo de Aplicaciones – R. Corchuelo, D. Ruiz, J.L. Arjona (Editores)

Métodos y Herramientas para el Desarrollo de Aplicaciones

Actas del Taller de Trabajo ZOCO

Rafael Corchuelo, David Ruiz y Jos é Luis Arjona
(Editores)

Métodos y Herramientas para el Desarrollo de Aplicaciones

Actas del Taller de Trabajo ZOCO

Rafael Corchuelo, David Ruiz, José L. Arjona

Publicado por The Distributed Group
ETSI Informática

ISBN: 978-84-690-5792-6

Imprime Artes Gráficas Domínguez, S.L.L.
Impreso en España

Copyright © 2006 Los autores

Se permite la reproducción total o parcial de este libro siempre y cuando sea con propósitos de aprendizaje o investigación y tan sólo se cobre por la misma los costes derivados de la reproducción de este material. Cualquier otro uso debe ser autorizado por los autores de los trabajos en cuestión.

Financiación: Proyecto AgilWeb (TIC2003-02737-C02-01) y Acción Complementaria Zoco (TIN2005-24794-E)

Índice general

Prólogo VII

1 An abstract architecture for service trading 1
- 1.1 Introduction 2
- 1.2 Process 2
- 1.3 Architecture 6
  - 1.3.1 Trading 6
  - 1.3.2 Discovery 7
  - 1.3.3 Information 8
  - 1.3.4 Selection 9
  - 1.3.5 Agreement making 10
  - 1.3.6 Binding 11
- 1.4 Related work 12
- 1.5 Conclusions 12

2 Consistencia y conformidad en un contexto temporal 15
- 2.1 Introducción 16
- 2.2 Demandas y ofertas con consciencia temporal 16
- 2.3 Aspectos de implementación 20
- 2.4 Trabajo relacionado 22
- 2.5 Conclusiones y trabajo futuro 24

3 Procesos de negocio con intervención humana 25
- 3.1 Introducción 26
- 3.2 Casos de estudio 27
  - 3.2.1 Atento 27
  - 3.2.2 GESiMED 28

3.3 Análisis de posibilidades de implementación 30
3.3.1 BPEL y BPEL4People 31
3.3.2 WS-Coordination/WS-Transaction 32
3.3.3 Codificación ad-hoc del proceso 32
3.3.4 Síntesis del análisis de alternativas 33
3.4 Conclusiones 34

4 Migración de aplicaciones locales a cliente/servidor 37
4.1 Introducción 38
4.2 Estado del arte 38
4.3 DARTOOL 39
4.3.1 Visión general de la herramienta 39
4.3.2 Entradas, salidas y limitaciones de DARTOOL 39
4.4 Arquitectura de la aplicación distribuida 41
4.5 Una vista detallada del proceso de migración 43
4.6 Adaptación a la arquitectura cliente/servidor 46
4.7 Conclusiones y trabajo futuro 49

5 Obtención de servicios en BBDDRR 51
5.1 Introducción 52
5.2 Estado del arte 53
5.3 Model-Driven Pattern Matching 53
5.4 Patrones de descubrimiento básicos 56
5.4.1 Patrón Clave Ajena (CA) 56
5.4.2 Patrón Doble Clave Ajena (DCA) 57
5.4.3 Patrón Clave Ajena n-aria (CANA) 59
5.5 Conclusiones y trabajo futuro 61

6 Auditoría a través de servicios web 63
6.1 Introducción 64
6.2 Arquitectura del servicio web 66
6.2.1 Lógica de negocio 68
6.2.2 Modelo de datos 70
6.3 Caso de aplicación 72
6.3.1 Ventajas e inconvenientes del uso de servicio web 72
6.4 Conclusiones 73

A Bibliografía 75

Índice de Figuras

1.1 Abstract architecture 5
1.2 Trading 6
1.3 Discovery 7
1.4 Information 8
1.5 Selection 9
1.6 Agreement making 10
1.7 Binding 11

2.1 Estructura de un documento con consciencia temporal de grado 1 17
2.2 Documentos con consciencia temporal de grado 2 17
2.3 Documentos con consciencia temporal de grado 3 19
2.4 Condiciones de calidad vigentes del 23 al 27 de diciembre 19
2.5 Resultado de la unión temporal 21

3.1 Arquitectura del sistema Atento 28
3.2 Composición para "Grabar interacción para un ciudadano" 29
3.3 Composición para "Obtener imágenes médicas procesadas" 30

4.1 Pantalla principal de DARTOOL 40
4.2 Arquitectura de la aplicación generada 42
4.3 Ejemplo de invocación de los métodos 43
4.4 Configuración de los objetos COM+ 45
4.5 Esquema de generación de las capas intermedias 47
4.6 Secuencia de invocaciones para instanciar una clase de dominio 48

5.1 Metamodelo de patrón para la búsqueda en un modelo SQL-92 54

5.2 Modelo, patrón y resultados del matching 56
5.3 Relación entre tablas A y B por una clave ajena Fk 58
5.4 Relación entre las tablas A, B y M con las claves ajenas Fk_1 y Fk_2 58
5.5 Interrelación n-aria 59
5.6 Relación n-aria de con n = 3 61

6.1 Arquitectura del Servicio Web 67
6.2 Métodos expuestos del Servicio Web 68
6.3 Modelo de datos 70
6.4 Ejemplo de uso del Servicio Web 72

Índice de Tablas

1.1 Comparison of abstract architectures for service trading 13

2.1 Grados de expresividad en las diferentes propuestas 23

3.1 Análisis de las posibilidades de implementación 35

4.1 Capas originales y capas incluidas 43
4.2 Comparación entre los modelos de capas 46

5.1 Equivalencias entre el patrón de entrada y los matchings 57

6.1 Descripción del modelo de datos 71

Capítulo 4

DARTOOL: Una herramienta para la migración de aplicaciones locales en C♯ a cliente/servidor

Ignacio García-Rodríguez, José M. Ruiz, Macario Polo, Mario Piattini
Escuela Superior de Informática. Universidad de Castilla–La Mancha
Paseo de la Universidad, 4. Ciudad Real 13071
(Ignacio.GRodriguez,Macario.Polo,Mario.Piattini)@uclm.es
JoseManuel.Ruiz@alu.uclm.es

Existe gran cantidad de software que a pesar de resultar muy útil, ha sido desarrollado para ser ejecutado de manera local. Cuando esto ocurre puede ser útil migrarla a una versión distribuida de la misma, de manera que pueda prestar su funcionalidad en un entorno distribuido. En este artículo, se presenta una primera aproximación para la migración de aplicaciones en modo local hacia arquitecturas cliente/servidor: DARTOOL. Esta herramienta facilita el proceso de migración en aplicaciones desarrolladas en C♯.NET. Partiendo del código fuente y el ejecutable de una aplicación genera, previa configuración del usuario, una nueva versión de la aplicación de entrada, pero orientada a una arquitectura cliente/servidor. DARTOOL aprovecha muchas características de la plataforma .NET para configurar y generar la aplicación distribuida.

4.1. Introducción

Tradicionalmente, la reingeniería ha sido aplicada a sistemas heredados con el objetivo de mejorar su calidad, añadir nuevas características, obtener una nueva versión del sistema, etc. [8]. La reingeniería suele aplicarse sobre sistemas con una edad y coste de mantenimiento considerable, desarrollados con técnicas y metodologías arcaicas, etc. [83]. Sin embargo, también aplicaciones desarrolladas bajo técnicas y paradigmas novedosos pueden ser susceptibles de sufrir un proceso de ingeniería inversa, migradas o analizadas mediante reingeniería, como ocurre en algunas propuestas [7, 31, 88].

En muchas ocasiones, también el cambio de paradigma o arquitectura es razón suficiente para acometer este cambio [15], incluso cuando el sistema funciona correctamente. Un buen ejemplo de esto es la migración de aplicaciones que se ejecutan en modo local hacia arquitecturas cliente/servidor. Donde se aplica reingeniería para obtener una aplicación cliente/servidor partiendo de otra que se ejecuta de manera local.

En este artículo se presenta DARTOOL, la primera versión de una herramienta desarrollada con este propósito. Basada en la plataforma .NET, DARTOOL implementa un proceso de migración para obtener, a partir de aplicaciones desarrolladas en C♯ [90], una nueva versión de las mismas, pero orientadas a la arquitectura cliente/servidor. Esta herramienta implementa un análisis puramente estático de la aplicación original. Aunque sujeto a algunas restricciones, las pruebas realizadas sobre aplicaciones de tamaño medio han producido buenos resultados. Sin embargo las restricciones a las que DARTOOL se ve sometida son parte del trabajo en curso.

Este artículo se organiza de la siguiente manera: la Sección §4.2 introduce algunos conceptos y terminología relacionada con .NET.; la Sección §4.3 resume el proceso de migración; la sección §4.4 detalla la arquitectura de la aplicación generada; la Sección §4.5 trata con más profundidad el proceso de migración; la Sección §4.6 explica cómo el código fuente original es modificado para poder distribuir la aplicación a la arquitectura cliente/servidor; la Sección §4.7 proporciona algunas conclusiones y líneas trabajo futuro.

4.2. Estado del arte

La migración puede ser vista como una de esas actividades del mantenimiento soportadas por la reingeniería, y puede ser aplicada, por ejemplo, a sistemas heredados basados en lenguajes desfasados [10, 15, 58, 82], bases de datos [2, 20], etc.

Parte del trabajo de migración se ha centrado en adaptar sistemas heredados a las nuevas tendencias tecnológicas, como la integración de sistemas heredados en la Web [10, 15, 16, 20, 51]. Sin embargo, la migración no es siempre posible, ya que depende en gran medida del diseño y arquitectura de la aplicación, existiendo situaciones en las que debido al acoplamiento interno de la aplicación no es posible distribuirla [24] o resulta muy complicado.

El Framework de .NET incluye características interesantes para la reingeniería y la migración hacia arquitecturas distribuidas. Por ejemplo, todos los metadatos que se incluyen en los ensamblados de .NET (accesibles mediante instrospección) permiten la extracción de mucha información útil acerca de la aplicación sin necesidad de conocimiento previo.

La tecnología Remoting [33], empleada en este trabajo, permite la interacción entre aplicaciones a través de la red. Emplea protocolos estándares como SOAP (para los mensajes), HTTP y TCP (ambos para la comunicación). Aparte de estos estándares, Remoting permite usar otros protocolos propietarios.

4.3. DARTOOL

4.3.1. Visión general de la herramienta

DARTOOL toma como entrada una aplicación desarrollada en el lenguaje C♯, y produce una versión de la misma orientada a la arquitectura cliente/servidor. La funcionalidad de la aplicación producida no varía, solamente la distribución. Tras la migración, habrá uno o varios clientes, y una aplicación servidora, donde reside la lógica de la aplicación.

Además, DARTOOL permite a los usuarios que tengan un gran conocimiento de .NET aprovechar las características avanzadas del Framework de .NET. DARTOOL permite configurar detalles de bajo nivel, como el protocolo de comunicación o los permisos sobre los objetos publicados en el servidor.

4.3.2. Entradas, salidas y limitaciones de DARTOOL

Debido al hecho de que DARTOOL implementa una primera aproximación, hay algunas limitaciones que deben ser tenidas en cuenta a la hora de migrar una aplicación. Debe tenerse en cuenta que no todas las aplicaciones pueden ser migradas hacia una arquitectura cliente/servidor, en lo que influye mucho el acoplamiento interno. En [24] se distinguen diferentes niveles que

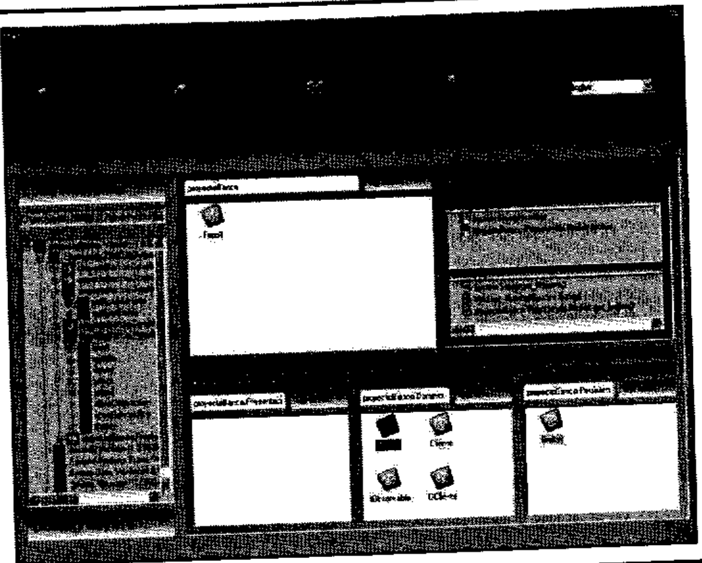

Figura 4.1: *Pantalla principal de* DARTOOL.

determinan la posibilidad de llevar a cabo la migración de un sistema heredado hacia una arquitectura cliente/servidor.

Las restricciones a tener en cuenta son las siguientes:

- La aplicación a migrar debe estar desarrollada siguiendo una arquitectura multicapa. De modo que se facilite la separación de los espacios de nombres correspondientes a la interfaz de usuario, la lógica de la aplicación y la comunicación con las bases de datos u otros sistemas.
- De la interfaz de usuario, que quedará en el cliente, no puede depender el resto del sistema, que irá al servidor. Si se necesita una comunicación bidireccional entre la interfaz de usuario y el resto de la aplicación, será necesario que la aplicación implemente el patrón observador [35].

Si por ejemplo la aplicación local no estuviera dividida en capas, o tuviera un alto nivel de acoplamiento entre los paquetes, o incluso si hubiera subrutinas de acceso a base de datos en la interfaz de usuario, sería necesario utilizar técnicas de slicing e incluso algoritmos de clustering para agrupar esas funcionalidades dispersas en el código [32].

DARTOOL toma como entrada: (1) la aplicación ejecutable y (2) el código fuente. Del archivo ejecutable (el ensamblado) se extrae la estructura de la aplicación utilizando los metadatos incluidos en el mismo. En la figura §4.1, la información extraída se muestra intuitivamente. A la izquierda se muestra la aplicación local como un árbol que muestra los espacios de nombres, librerías, clases, métodos o atributos de la aplicación, y permite ver una composición detallada de la aplicación. En el centro hay un explorador para navegar por la aplicación, ver sus capas e indicar a DARTOOL (ver las 3 cajas de la zona inferior) qué espacios de nombres pertenecen a cada una de ellas. No hay forma sencilla de determinar qué paquetes de la aplicación corresponden con la interfaz de usuario, y cuáles se corresponden con la lógica de la aplicación. Por ello el usuario tiene la posibilidad de dividir la aplicación por si mismo (atendiendo a la arquitectura multicapa). Esta información sobre la división en capas es necesaria en los siguientes pasos. El código fuente se usa para regenerar la nueva versión de la aplicación. Una vez que DARTOOL dispone de la información sobre las capas de la aplicación a distribuir, y la configuración de la aplicación final (más adelante), genera automáticamente una aplicación distribuida cliente/servidor.

4.4. Arquitectura de la aplicación distribuida

En esta sección se describe brevemente la aplicación distribuida. En la figura §4.2 se muestran de manera conjunta la arquitectura de la aplicación local (lado inferior) y su versión distribuida de la misma (lado superior). Según la figura §4.2, los componentes de la aplicación ya distribuida (generada por DARTOOL) son los siguientes:

- Componentes de la Interfaz de Usuario: Representan los espacios de nombres de la aplicación original que implementan la interfaz de usuario. El proceso de migración implementado por DARTOOL no modifica la interfaz de usuario, permaneciendo esta con el mismo aspecto que en la aplicación inicial.
- Capa de conexión: La capa de conexión esta compuesta por un conjunto de clases capaz de representar a la capa de dominio de la aplicación original. Esta capa es la encargada de redireccionar las invocaciones que se realizan desde las clases de la capa de presentación hacia las clases de la capa de dominio.

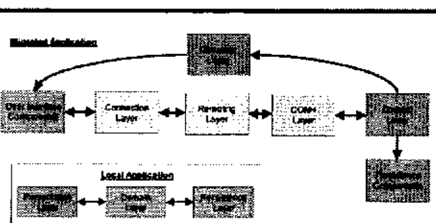

Figura 4.2: *Arquitectura de la aplicación generada.*

- Capa remoting: Esta capa representa al motor Remoting, que a su vez tiene la responsabilidad de gestionar las invocaciones realizadas desde la interfaz de usuario hacia los objetos remotos.
- Capa COM+: Representa el conjunto de componentes COM+ instalados en el servidor. Éstos componentes accederán a los componentes de la capa de dominio original, ubicados ahora en el servidor. Aunque esta capa no es necesaria, su uso provee al usuario de ventajas avanzadas tales como cierto control de la seguridad, control transaccional, etc.
- Capa de dominio: Esta capa contiene todas las clases de la capa de dominio de la aplicación original. Estas clases son accedidas desde la capa de dominio gracias a las capas intermedias (ver figura §4.2).
- Capa observador: En ocasiones, la capa de dominio debe enviar algún mensaje hacia alguna clase de la capa de presentación. En estos casos, y para preservar el bajo acoplamiento entre las capas del sistema, es necesario aplicar el patrón observador..
- Capa de persistencia: Muchas aplicaciones utilizan bases de datos para almacenar la información que manejan. Si la aplicación local incluye alguna capa para la gestión de la base de datos, esta es incluida en la aplicación migrada, y además sin ningún tipo de cambio, ya que su comportamiento permanece inalterado.

Para clarificar las capas existentes (junto con la figura §4.2), se incluye la Tabla §4.1, donde se incluyen las capas originales y las que DARTOOL incluye en la aplicación migrada.

La figura §4.3 muestra un sencillo ejemplo (y comparación) de cómo se lleva a cabo la invocación de un método. Las cajas representan clases, cada

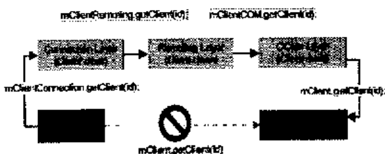

Figura 4.3: *Ejemplo de invocación de los métodos.*

Capas originales	Capas intermedias
Presentación	Conexión
Dominio	Remoting
Persistencia (si existe)	COM+
Observador (si existe)	COM+

Tabla 4.1: *Capas originales y capas incluidas.*

una perteneciente a una de las capas enumeradas anteriormente (presentación, conexión, remoting, COM+). La capa de persistencia se ha eliminado del ejemplo por no ser necesaria. Las clases inferiores representan respectivamente una clase de la capa de presentación y otra de la capa de dominio. Las tres clases intermedias (arriba en la figura) representan las clases correspondientes a la capa de dominio en las capas intermedias. Es decir, para la clase Client de la capa de dominio, las correspondientes clases Client de conexión, remoting, y COM+. Tal y como se observa abajo en la figura §4.3, existe una flecha que va directamente de la clase de la capa de presentación hacia la clase de la capa de dominio. Esta invocación corresponde a la que se sucede en la aplicación original.Sin embargo, ese mismo mensaje se propagaría en la aplicación migrada tal y como se observa entre las clases superiores de la figura §4.3.

4.5. Una vista detallada del proceso de migración

El fichero ejecutable (o ensamblado) contiene gran cantidad de metadatos accesibles mediante introspección. Estos metadatos son recuperados mediante reflexión. Esta característica, presente en otros lenguajes de programación como Java, permite extraer información sobre el ejecutable tal y como puede ser su distribución en espacios de nombres y clases, los métodos y atributos de

cada clase, tipos definidos en los ensamblados, etc. [34]. Además de esta información, DARTOOL recupera también información sobre librerías que puedan ser necesarias (dlls). Toda esta información se presenta en la pantalla principal de DARTOOL (ver figura §4.1).

A la izquierda de la figura §4.1 puede verse una vista en forma de árbol del ensamblado, que resulta muy útil para explorar el contenido del ensamblado de manera jerárquica. La esquina superior derecha representa la misma información que el árbol, pero organizando ahora el ensamblado por espacios de nombres. Esta vista no está pensada para explorar a fondo el ensamblado, sino para que el usuario especifique a DARTOOL qué espacios de nombres de la aplicación original pertenecen a la interfaz gráfica, cuáles implementan la lógica de la aplicación, y cuáles implementan las subrutinas para la gestión de las bases de datos. Esta caja (arriba a la derecha) junto con las tres que hay debajo permiten desglosar la aplicación en capas.

A pesar de que en la interfaz solamente se considera una arquitectura de tres capas, lo cierto es que en cada una de estas capas pueden incluirse los espacios de nombres que sean necesarios. Así por ejemplo, una aplicación podría tener implementada su lógica en dos o más espacios de nombres, y lo único que habría que hacer es situar esos espacios de nombres en la caja de dominio de DARTOOL. Sin embargo, para futuras versiones de DARTOOL se tratará de que esta identificación se realice de manera automática, ya que no siempre el usuario tiene por que tener este conocimiento.

Una vez dividida la aplicación original, es necesario configurar cómo será la aplicación distribuida. En la primera parte de la configuración, el usuario debe decidir, cuáles de las capas intermedias deben ser generadas en la aplicación distribuida. Tal y como se explicó anteriormente, las capas intermedias eran conexión, remoting y COM+, sin embargo, no todas ellas tienen por que ser incluidas en la aplicación final. Ello dependerá de los requisitos que deba tener la aplicación distribuida, la experiencia y el conocimiento que tenga ingeniero en las tecnologías de .NET.

La configuración de las capas responde a tres tipos de modelos:

1. Conexión-Remoting (CR): En este modelo la capa remoting es la encargada de gestionar el acceso a las clases de dominio. Este modelo permite una comunicación rápida entre las capas de dominio y presentación. Por otro lado, este modelo no contempla ninguna característica de seguridad en el acceso. Este modelo resulta útil cuando la clase de dominio tiene una gran cantidad de clases (ya que es bastante ligero) y no hay restricciones de seguridad específicas.

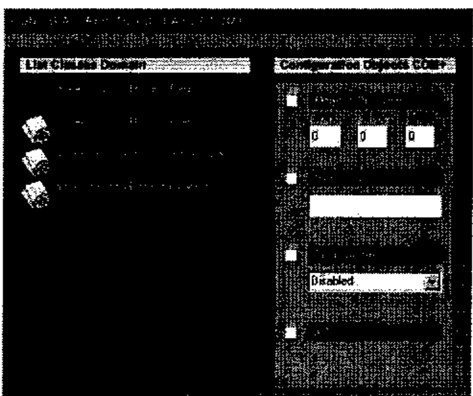

Figura 4.4: *Configuración de los objetos COM+.*

2. Conexión-COM+ (CC+): La capa de remoting no se incluye en este modelo. En esta configuración, los componentes COM+ permiten la configuración de características de seguridad y control de transacciones, entre otras características (ver figura §4.5). Este modelo resulta útil para aquellas aplicaciones donde la seguridad es un requisito importante.

3. Conexión-Remoting-COM+ (CRC+): Este modelo incluye todas las capas intermedias mencionadas. Aquí se une facilidad para el acceso de remoting y la versatilidad de COM+. Sin embargo, este modelo se ve penalizado en tiempo de acceso por el incremento del número de capas que deben atravesar las llamadas.

La Tabla §4.2 ofrece una pequeña comparativa entre los modelos de capas que ofrece DARTOOL.

Tal y como se mencionó, en la aplicación migrada encontramos un servidor (siempre necesario) y varias aplicaciones cliente. Todos los clientes comparten la misma aplicación servidora, sin embargo no es obligatorio que todos los clientes compartan las mismas instancias de las clases. Por ejemplo, si tres clientes dependen del estado de una clase, todos ellos deben trabajar con la misma instancia (ejemplo de patrón Singleton [48]), ya que de lo contrario, cada uno tendría una visión equivocada del estado actual. Sin embargo, también puede ocurrir que cada cliente trabaje con sus propias instancias. Una

Característica-Modelo	CR	CC+	CRC+
Tiempo de respuesta	Bajo	Medio	Alto
Optimización	Bajo	Alto	Alto
Comunicación	Bidireccional	Unidireccional	Unidireccional
Escalabilidad	Alto	Medio	Bajo

Tabla 4.2: *Comparación entre los modelos de capas.*

situación compleja que queda como trabajo futuro es aquella en la que unos clientes necesitan trabajar sobre la misma instancia clase, mientras que otros manejan la suya propia.

Para solucionar este problema, DARTOOL ofrece varios modos para el acceso a los objetos de la capa de dominio:

- Singleton instances model: Las clases de la capa de dominio que se vinculan con este modelo sólo se instancian una vez en cada ejecución, de forma que todos los clientes comparten siempre las mismas instancias. Siempre que sea posible, es mejor configurar las clases como Singleton, ya que de esta forma se obtiene un mejor rendimiento de los recursos existentes. Además, este modelo facilita en gran parte la interacción entre los diferentes clientes que pueda haber conectados, siempre que todos necesiten notificación de los cambios realizados sobre alguna clase.
- Not Singleton Instances Model: Cada clase de dominio vinculada a este modelo creará una instancia distinta por dada cliente que la invoque. En este modelo, cada aplicación cliente posee sus propias instancias, evitando así afectar a otros clientes en los cambios producidos. Sin embargo, este modelo puede reducir el rendimiento, al elevarse el número de objetos en el servidor con el número de clientes.

La mejor solución para evitar la carga excesiva en el servidor es adoptar un enfoque mixto. De esta forma, las clases necesarias con marcadas como Singleton de manera que se compartan entre todos los clientes, y las que deben ser propias a cada cliente son marcadas como no Singleton.

4.6. Adaptación a la arquitectura cliente/servidor

El proceso de migración no consiste sólo en la adición de nuevas capas a la aplicación inicial. El código fuente de la aplicación debe ser cuidadosamente

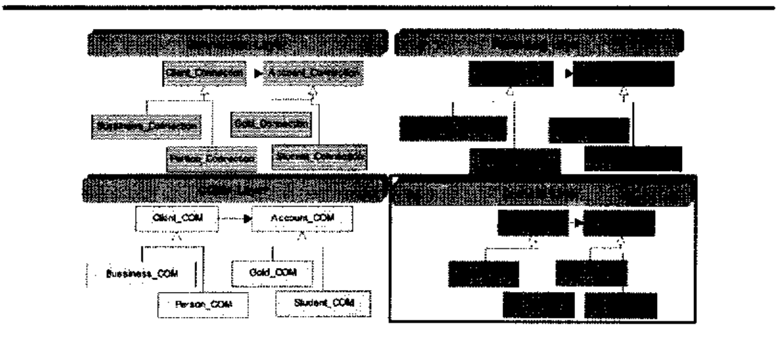

Figura 4.5: *Esquema de generación de las capas intermedias.*

analizado e incluso (en algunas ocasiones) modificado, según encontremos jerarquías de clases, propiedades, interfaces, etc. Así pues, en esta sección se tratará cómo gestiona DARTOOL la presencia de los elementos que puede encontrar en el código fuente cuando éste se está analizando.

En la aplicación inicial, cualquier clase de la interfaz de usuario puede invocar otra clase de la capa de dominio, y esta a su vez, devolver un valor a la primera (si fuera necesario). Sin embargo, cuando la aplicación migrada realiza esta misma acción, lo que realmente ocurre es que la invocación que realiza por parte de una clase de la interfaz de usuario es una invocación a un método remoto que está situado en una clase en el servidor. Sin embargo, este problema de invocación remota no es tan simple como puede parecer, ya que las clases de la capa de presentación pueden estar trabajando con las jerarquías de clases establecidas en la capa de dominio, o incluso instanciar clases de manera dinámica (el tipo de la clase a instanciar sólo se sabe en tiempo de ejecución). Por este motivo, el separar la aplicación original en una parte cliente y una parte servidor no resulta tan trivial como podría parecer a simple vista.

La estrategia adoptada es replicar de manera ligera el contenido de la capa de dominio en las capas intermedias (conexión, remoting y COM+). Es una réplica ligera porque estas capas no incluyen toda la capa de dominio, sino un esqueleto de la misma (ver figura §4.5). De esta forma se preserva la estructura y las relaciones son preservadas a lo largo de las clases intermedias.

Las interfaces (clases) incluidas en la capa de dominio también se tienen en cuenta en las capas intermedias, así como las relaciones entre éstas y las clases (por ejemplo, una clase de la interfaz de usuario podría conocer una instancia de una clase de dominio que implementa una interfaz).

Figura 4.6: *Secuencia de invocaciones para instanciar una clase de dominio.*

Los métodos constructores de las clases deben ser tratados de manera especial. Un método constructor tiene el objetivo de crear una nueva instancia de la clase cuando este es invocado. Tal y como se dijo previamente, las capas intermedias implementan un esqueleto que representa las clases de dominio. En este sentido, las clases de la interfaz de usuario deben ser capaces de instanciar cualquier clase de dominio tal y como lo hicieran en la aplicación original. Sin embargo, para tal instanciación hay que atender a ciertas restricciones impuestas por la propia tecnología (se verá más adelante).

Para la instanciación de clases de dominio, las clases de las capas intermedias incluyen también constructores. En el proceso de instanciación, la clase de la interfaz de usuario invocaría el método constructor, sobre la clase correspondiente en la capa de conexión. Esta a su vez invoca al constructor de la clase de la siguiente capa, la de Remoting, y así hasta llegar a la clase de dominio ubicada en el servidor. Sólo entonces se crea la instancia de la clase de dominio y es devuelta a la clase de la interfaz de usuario. Sin embargo, hay ocasiones en las que utilizando COM+ y Remoting no es posible un constructor con parámetros, pero sí un constructor por defecto que carece de parámetros [33]. Para resolver este problema, cuando una clase pertenece que a una de estas capas debe ser instanciada, se crea en primer lugar una instancia con este constructor por defecto, y se invoca en segundo lugar a un método que acepta los mismos argumentos que tiene el constructor original, y que es el encargado de seguir la cadena de invocaciones para realizar la instanciación.

La figura §4.6 esquematiza la secuencia de pasos para instanciar la clase Client de la capa de dominio. La primera caja representa ésta clase en la capa conexión, y su constructor se muestra como el original. Sin embargo, la clase Client de las dos siguientes capas sigue la estructura de un constructor sin argumentos (por defecto), que se invoca siempre en primer lugar, y un método que se invoca en segundo lugar (que sigue el patrón de nombrado new[class_name]), y que propaga la invocación. Este método hace las veces de constructor, pero adaptado a las capas intermedias de Remoting y COM+.

4.7. Conclusiones y trabajo futuro

En este artículo se ha presentado DARTOOL, una herramienta para la migración de aplicaciones locales, escritas en $C^\sharp$, hacia aplicaciones basadas en la arquitectura cliente/servidor. DARTOOL lleva a cabo un análisis estático tanto del código fuente como del ejecutable de la aplicación original. Utilizando la información recuperada tras este análisis, y la información suministrada por el usuario, DARTOOL genera una aplicación distribuida completamente funcional, basada en una parte servidora y tantos clientes como sea necesario.

La propuesta que se presenta en este artículo es una primera versión que debe ser mejorada mediante la adición de capacidades para el análisis dinámico y técnicas de clustering que puedan detectar funcionalidades que no se implementen solamente en un espacio de nombres (o lo que es lo mismo, que estén presentes en más de una capa de la arquitectura multicapa). Bajo ciertas circunstancias los componentes COM+, que se generan para acceder a la capa de dominio en el servidor, pueden fallar. Por esta razón, esta parte debe ser sometida a prueba más severamente para mejorar su fiabilidad.

La compatibilidad que existe entre los lenguajes de .NET es un hecho probado, que se debe en parte a la representación común que tienen cuando son compilados y a que utilizan el mismo sistema de tipos de datos. Por este motivo, resulta interesante ampliar la propuesta para poder migrar aplicaciones desarrolladas sobre otros lenguajes como Visual Basic.NET o $J^\sharp$.NET.

A pesar de que DARTOOL funciona bien con aplicaciones de tamaño medio, es necesario llevara cabo experimentos con aplicaciones más grandes.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto MÁS, Ministerio de Ciencia y Tecnología/FEDER, TIC2003-02737-C02-02; el proyecto ENIGMAS, Plan Regional de Investigación Científica, Desarrollo Tecnológico e Innovación, Junta de Comunidades de Castilla-La Mancha, PIB-05-058, el proyecto FAMOSO, parcialmente financiado por el Ministerio de Industria, Turismo y Comercio, FIT-340000-2006-67 Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica 2004-2007 y el Fondo Europeo de

Desarrollo Regional (FEDER), Unión Europea, y el proyecto MECENAS, Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, PBI06-0024.