Università degli Studi di Parma
Facoltà di Scienze MM. FF. NN.
Corso di Laurea in Informatica

Giulio Destri
Oscar Figus
Alberto Picca

AreaMVC: Linee guida per l'applicazione pratica di MVC in .NET e Java

Collana "I quaderni di Area Professional"

URL originale: http://www.areaprofessional.net/documenti/AreaMVC.pdf

AreaMVC: linee guida per l'applicazione pratica di MVC in .NET e Java

Introduzione

L'ormai storico pattern MVC, nato nei laboratori Xerox negli anni'70 all'alba dell'era delle interfacce grafiche (si veda [1]), ha dato origine a molti pattern derivati e ad un enorme numero di implementazioni, venendo integrato in diversi tra i frame work più usati come JEE, .NET, Ruby.
In particolare il pattern Model View Presenter [2], nato negli anni'90, pur essendo oggi classificato come "retired" [3], ha dato vita a molte implementazioni e traendo ispirazione da esso i progettisti di Microsoft hanno realizzato nel framework .NET alcune delle caratteristiche di base presentate nel corso di questo articolo.
In questo articolo vogliamo concentrarci su un idioma, ossia un pattern di implementazione [4], ispirato dalla applicazione delle linee guida di MVC e applicabile con ottima efficacia in .NET e, sia pure con meno immediatezza, in Java.
Il contesto di applicazione è soggetto ai seguenti vincoli:

- Separazione netta di logica business ed interfaccia (come in tutti i derivati di MVC) e disaccoppiamento totale dell'interfaccia dai sistemi di persistenza sottostanti
- Incapsulamento di tutta la logica business entro macro-oggetti contenitori, per poterli riusare in contesti molto diversi
- Definizione dei blocchi di logica business a partire dall'analisi funzionale (tipicamente un blocco di logica business corrispondente ad uno use case), ovvero incapsulamento totale di moduli preesistenti
- Accesso il più diretto possibile al database, per quanto la logica business può nascondere al proprio interno anche sistemi multistrato, come ad esempio un modulo basato su un ORM come Hibernate
- Comunicazione dei dati fra gli strati software come oggetti entità o simili (ad esempio, le datatable di .NET).

Questo idioma può essere derivato dal pattern MVC-Model 2 di Sun [5] o da un esempio di Martin Fowler [6] ed è ampiamente usato, con diverse varianti, in progetti piccoli e medi nelle software house italiane. Una implementazione open source che segue queste linee guida è l'esempio di NSK su codePlex [7]. In particolare questo idioma, denominato di qui in seguito **AreaMVC**, è stato usato dagli autori per realizzare i progetti con tecnologia Java e .NET in AreaSP e su di esso è basata la libreria .NET AreaFramework [8].
Nel corso del presente articolo, partendo dall'architettura teorica di MVC verrà mostrato come nell'idioma AreaMVC vengono mappati gli elementi architetturali .NET e Java su quelli del modello teorico e come.

Lo schema di MVC

In figura 1 è rappresentato lo schema logico del pattern MVC teorico. I tre componenti rivestono ognuno il proprio ruolo.
Nei pattern derivanti dal modello teorico sono state introdotte alcune differenze: in alcune varianti il model è formato soltanto dalle entità ed il controller fornisce l'infrastruttura per gestirle, mentre in altre il model rappresenta e "contiene" la logica business completa. In quanto segue ci rifaremo a quest'ultima categoria di varianti, nella quale l'input dell'utente, il programma che modella i processi del dominio business e la risposta visuale (testo, immagini ecc...) del sistema all'utente sono separati fra loro e gestiti, rispettivamente:

- dal modulo controller

- dal model
- dal view.

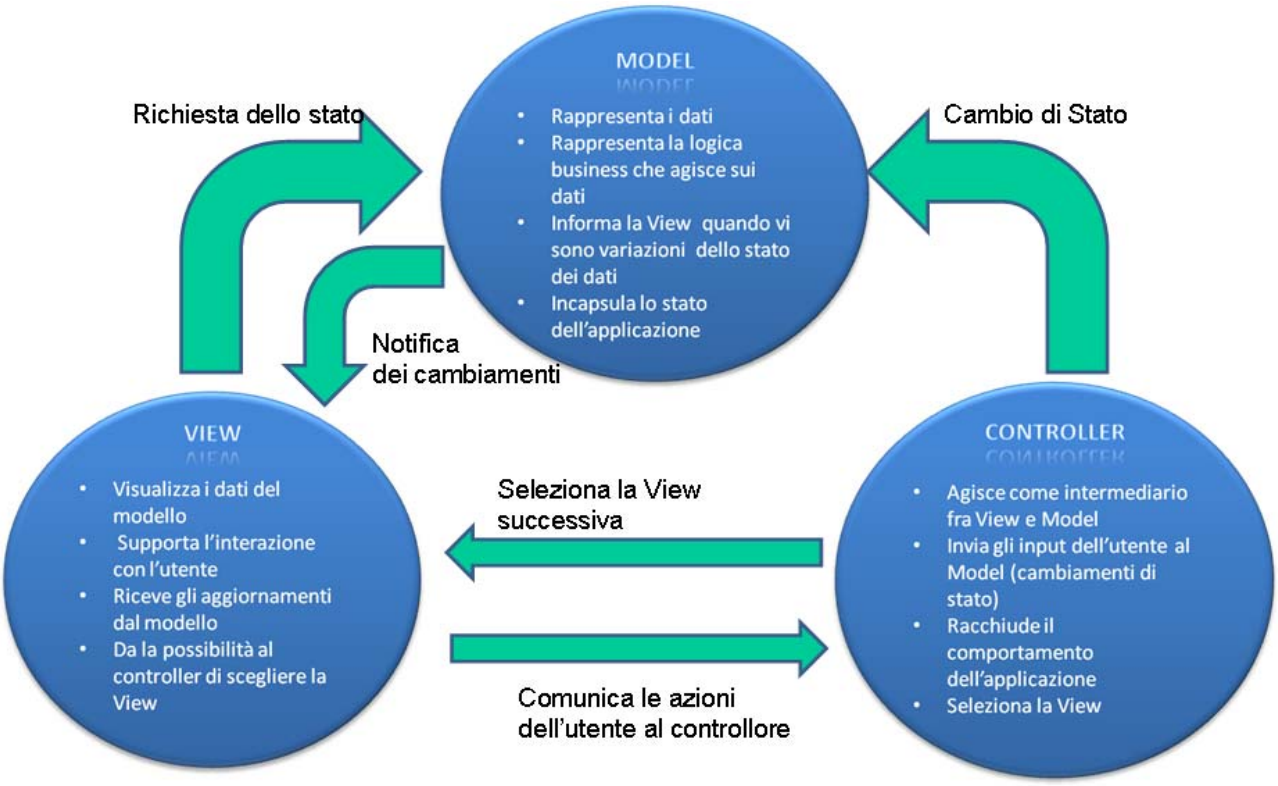

Figura 1: il pattern MVC teorico

Il **controller** interpreta gli eventi di input via tastiera, mouse o il dispositivo di input del sistema fisico su cui l'interfaccia utente opera dell'utente (si pensi, ad esempio, al tastierino di un telefono cellulare). Il controller quindi mappa questi eventi, traducendoli in comandi/segnali (normalmente invocazioni di metodi), che invia al model per realizzare il cambiamento di stato o l'azione richiesti, oppure al view per fare variare la visualizzazione (ad esempio attivare una voce di menu prima non attiva).
Pertanto si può affermare che il controller è il modulo del programma (o della sezione di programma in esame) attraverso cui l'utente interagisce con l'applicazione. Compito del controller è ricevere l'input dall'utente ed fare compiere al model e alla view le azioni associate a tale input. Quindi il controller è responsabile di mappare le azioni dell'utente sulle risposte dell'applicazione.
Il **model** gestisce uno o più elementi dei dati, risponde alle interrogazioni relative al suo stato interno e reagisce alle istruzioni relative al cambio di stato. Il model viene usato per trattare le informazioni, ma può anche esser usato per notificare gli osservatori (ossia i moduli view che mostrano i dati all'utente) quando le informazioni cambiano, (es. il news ticker di borsa).
Possiamo considerare il model come il risultato di una analisi, modellazione ed adattamento al mondo informatico del mondo reale, ovvero come una "approssimazione" computazionale di un sistema o processo del mondo reale (o meglio, del dominio di business).
Il model contiene quindi non solo lo stato del sistema reale ma anche i suoi principali processi di funzionamento. In pratica quindi in un'applicazione ad oggetti il model è l'insieme dei componenti interni dell'applicazione, risultanti dalla proiezione sotto forma di

classi ed oggetti entro il dominio software delle entità del dominio di business e delle loro relazioni.
Nell'ottica della progettazione, sarebbe necessario raggruppare entro uno stesso modulo model soltanto dati e funzionalità legate da uno scopo comune, mentre per mettere assieme due gruppi di dati e funzionalità non in relazione tra loro sarebbe bene creare due model separati, uno per ciascuno di essi.
In particolare si può pensare ad un model che crea un "ponte" fra le parti interne di un sistema informatico e la sua interfaccia utente. In questo scenario il modello "avvolge" ed astrae tutte le funzionalità interne del sistema(software o hardware) e agisce da collegamento con il mondo esterno, ovvero l'interfaccia utente.
La **view** (detta anche vista o viewport) gestisce un'area del display o anche l'intero display. E' responsabile della presentazione dei dati all'utente attraverso un'apposita combinazione di elementi grafici e testuali, adatti alle caratteristiche del dispositivo "fisico" di visualizzazione. Ad esempio una view costruita con una pagina Web ha caratteristiche diverse da una costruita con una Windows form e ancor più da una adattata al display grafico di un dispositivo mobile. Si pensi, ad esempio. ai siti Web in cui, accanto alla parte standard HTML viene sviluppata una corrispondente allo standard Wap. In pratica quindi la view ha lo scopo di mappare una visualizzazione (grafica o no) su un dispositivo di output; la view ha una corrispondenza con le caratteristiche del dispositivo e "sa" come rappresentare i dati su di esso. La view si "aggancia" ad un model e visualizza i suoi contenuti sul dispositivo di output.
Nel pattern teorico, come mostrato in figura 1, esiste anche un legame diretto fra view e model: in molte implementazioni, quando il model cambia, la view automaticamente aggiorna le parti corrispondenti della visualizzazione, in modo da rendere visibili tali cambiamenti per l'utente.
Per lo stesso model possono esistere diverse view (si pensi, ad esempio, alle possibili rappresentazioni dei dati di una tabella numerica, sotto forma di grafici di vario tipo). Le view in alcuni casi sono adattabili e quindi possono visualizzare gli stessi dati su dispositivi di output diversi in modo diverso. Una view inoltre può essere composita, ovvero formata da varie sub-view, eventualmente a loro volta composite.

Mappatura tra elementi MVC ed elementi reali

L'idioma AreaMVC si implementa ovviamente in modo diverso secondo la piattaforma utilizzata. Prima di vedere le singole implementazioni è necessario stabilire gli elementi comuni, derivanti dall'MVC teorico, che rispettano i vincoli sopra definiti.
Il model è il cuore dell'idioma, è formato da una classe che espone i metodi corrispondenti alle funzionalità definite dallo use case di analisi che le ha dato origine. I metodi accettano i dati opportuni e restituiscono i dati opportuni. Per capire meglio questo concetto è utile una metafora tratta dall'hardware del PC. Il dispositivo di input primario, la tastiera, si compone di una interfaccia esterna, elettromeccanica, composta dai tasti e dalla parte sottostante, ossia i contatti elettrici che essi azionano meccanicamente. A parità di lavoro logico svolto dai contatti sono realizzabili tastiere molto diverse. Analogamente, per il mouse o per il trackball il dispositivo di input è la pallina, "sotto" la quale stanno i sensori che ne registrano il moto. E ogni tastiera e ogni mouse è collegata al PC con una determinata porta standard (USB, PS/2 o altro).
Quindi il model è la parte "sottostante alla tastiera", la cui interfaccia espone i metodi corrispondenti alle funzionalità e generici, ossia indipendenti dall'interfaccia utente, con il suo controller e la sua view adattati al tipo di dispositivo fisico utilizzato, che la formano.
Il model contiene le funzioni di trattamento dei dati ed eventuali strutture di memorizzazione dei dati stessi (ad esempio, cache di risultati), nonché le funzioni per

rendere persistenti i dati stessi, tipicamente entro un DBMS relazionale. Questa complessità implica che in realtà anche il model dovrà poi avere una sua struttura stratificata per migliorarne la manutenibilità e l'adattabilità, che sarà descritta in seguito.
Il controller contiene invece i metodi di reazione (anche chiamati metodi reattori) agli eventi raccolti dall'interfaccia utente, eventi che saranno fortemente determinati dal tipo di interfaccia (Windows form, pagina HTML o Web form, midlet su dispositivo mobile ecc…). Seguendo l'analisi definita dallo use case ciascun metodo reattore, corrispondente all'interazione dell'utente con un determinato controllo visuale (ad esempio, bottoni, menu a tendina, campi di immissione testo ecc…), dovrà richiamare al proprio interno il o i metodi opportuni esposti dalla interfaccia esterna del model che corrispondono alla realizzazione di quella funzionalità.
La view corrisponde alla interfaccia visuale vera e propria, ovvero alle istruzioni di codice necessarie per la sua renderizzazione e contiene gli elementi corrispondenti ai dati che dovranno essere visualizzati, se l'interazione con l'utente conduce ad un output, od inseriti nel sistema, se l'interazione è di input. Sono individuabili diversi modelli generali di view, in particolare la vista a griglia o tabella dei dati, di solito realizzata con componenti opportuni (ad esempio, la JTable in Java e la DataGridView in .NET) e la vista di dettaglio o maschera, per la quale non sempre esistono componenti già presenti nel framework (ad esempio, in ASP.NET è presente il componente DetailView che non ha un corrispondente nelle Windows form).
I casi d'uso, durante la loro stesura e raffinazione, dovranno tendere a definire anche il tipo di view, in funzione della interazione con i dati che si vuole ottenere.
Naturalmente occorre anche definire che tipo di strutture dati viaggiano fra gli strati software (ad esempio, oggetti entità, strutture tabellari, vettori di parametri ecc…) per permettere una trasmissione efficiente e manutenibile dei dati stessi all'evolvere dell'applicativo.
Questo insieme di linee guida generali deve essere poi "mappato" sulle strutture messe a disposizione dai framework. Nei paragrafi successivi verrà trattata la implementazione reale usando le strutture basilari di .NET e di Java.

Implementazione di AreaMVC in .NET

Il framework .NET è stato disegnato con anni di esperienza alle spalle, sfruttando i pattern e l'ingegneria del software per porre rimedio ai più gravi difetti dello sviluppo software manifestatisi nell'era del Visual Basic e del VisualC/C++. Sono poi stati aggiunte negli ultimi tempi anche estensioni al frame work come MVC.NET che implementano al meglio il pattern teorico.
Ma l'impronta del pattern MVC è presente anche nella dotazione base di .NET, per il quale l'implementazione di AreaMVC diventa come descritto in seguito.
Per quanto riguarda le Windows form, a partire dal .NET 2.0 con Visual Studio 2005, è stato introdotto il concetto di partial class [9] che ha consentito di suddividere la definizione di una classe in più file. Pertanto, quando creiamo una Windows form dentro l'ambiente di Visual Studio, si ottengono normalmente due o più file (se si considera anche il file delle risorse, come per esempio le icone, avente estensione .resx):

- uno avente il nome della form con estensione .cs o .vb che rappresenta il controller, in cui vengono creati con gli appositi comandi i metodi reattori;
- uno avente il nome della form con estensione .Designer.cs o .Designer.vb, che rappresenta la view e all'interno del quale stanno le istruzioni di costruzione, posizionamento e visualizzazione dei controlli visuali aggiunti graficamente col il visual composer, oltre che le istruzioni di registrazione degli eventi che associano ai controlli i metodi reattori che stanno nel controller.

Un progetto di class library, con dentro le strutture dati e le funzioni appropriate, va a costituire il model, che corrisponde fisicamente alla libreria così ottenuta. In realtà, grazie al disaccoppiamento fra form e model, si rende possibile avere più librerie richiamate entro una stessa form, oppure una stessa libreria richiamata da form diverse che ne usano funzionalità diverse, garantendo una buona modularità e riusabilità del codice così ottenuto, come riportato in figura 2.
La classe o le classi corrispondenti al model e incapsulanti tutta la logica business vengono istanziate come oggetti di servizio entro la classe della form. Estremizzando la necessità di minimizzare il codice in alcuni casi sarebbe possibile anche creare delle classi di servizio composte unicamente di metodi statici, che quindi non richiedono l'istanziazione delle classi, ma questo, oltre a irrigidire la struttura, non è comunque sempre possibile.
La situazione è molto simile anche per le Web form di ASP.NET. Infatti anche qui, sfruttando un meccanismo analogo a quello della partial class, ciascuna form viene suddivisa in:

- un file avente nomeform con estensione .aspx, che corrisponde alla view, entro la quale, scritto in codice HTML o XHTML è la descrizione visuale della form con il posizionamento dei vari controlli Web;
- un file avente nomeform con estensione .aspx.cs o aspx.vb, che corrisponde al controller, entro il quale sta il codice (chiamato anche tecnicamente code-behind) dei metodi reattori agli eventi raccolti dalla pagina Web contenuta nella view;
- un file avente estensione aspx.designer.cs o .vb che serve a collegare i due elementi suddetti, facendo sì che gli elementi visuali della view, descritti da strutture XML, corrispondano a variabili .NET (attributi della classe) entro il code-behind del controller, esattamente come avviene nelle Windows form.

La connessione fra elementi visuali e reattori avviene molto semplicemente attraverso appositi attributi dei descrittori XML (ad esempio, onclick="nome del metodo reattore").
Il code-behind viene compilato in precedenza, le parti delle view sono compilate alla prima chiamata della form attraverso il Web, e poi gli elementi .NET così ottenuti vengono gestiti dal motore .NET associato al Web server (IIS o Apache), realizzando l'interazione con l'utente.
Una caratteristica estremamente importante è che la libreria che corrisponde al model, a meno di vincoli particolari imposti dal carico atteso per il sistema, si può riusare senza problemi entro l'applicazione Web. Questo significa che la trasformazione di un applicativo Windows form in applicativo Web si realizza semplicemente sostituendo le form con le Web form. I controlli visuali Web di .NET sono stati progettati per essere molto simili alle loro controparti nelle Windows form, per cui anche i nomi degli eventi sono spesso uguali. Quindi la riscrittura della form come Web form è una operazione estremamente veloce.
La stessa struttura inoltre esiste nelle applicazioni Web service:

- un file .asmx costituisce la view;
- un file .asmx.cs o .asmx.vb costituisce il file con il code-behind del controller.

Pertanto, per trasformare una libreria "model", ovvero un componente di logica business in un Web service, è sufficiente "incapsularlo" includendo la classe entro un progetto Web service, e creando un wrapper che esponga, cambiandone la firma ovvero l'interfaccia se necessario, i metodi che si vuole esporre come servizi Web. E la parte client può essere ottenuta facilmente con il generatore automatico di proxy che si ha con la funzione di aggiunta riferimento Web di Visual Studio, includendolo entro il progetto Windows o Web form che costituiva l'interfaccia utente dell'applicativo basato su AreaMVC descritto in precedenza.
Anche la struttura del Windows Presentation Foundation [10] è ispirata ad MVC. Infatti anche qui troviamo per ogni form:

- un file con estensione .xaml, che corrisponde alla view e che contiene i descrittori XAML dei controlli visuali inseriti, corredati degli attributi XML che indicano i metodi reattori;
- un file con estensione .xaml.cs o .vb, che corrisponde al controller e che contiene il code-behind, in modo del tutto analogo alle Web form di ASP.NET.

E anche qui si può riusare un insieme di classi model definite nelle form tradizionali.
Usando la Windows Workflow Foundation, si possono includere entro le classi di tipo Activity, ovvero i componenti elementari del workflow, le classi del model, realizzando un completo riuso della logica business in esse contenuta.
E da ultimo, scrivendo applicativi di tipo console, è possibile riusare tali classi anche entro contesti di applicazioni batch o di servizi Windows, come riportato in figura 3.
In pratica quindi seguire le linee guida di AreaMVC è abbastanza facile e, a prezzo di una complicazione talvolta anche notevole del design, si ottiene una reale riusabilità di tutto il codice della logica business.

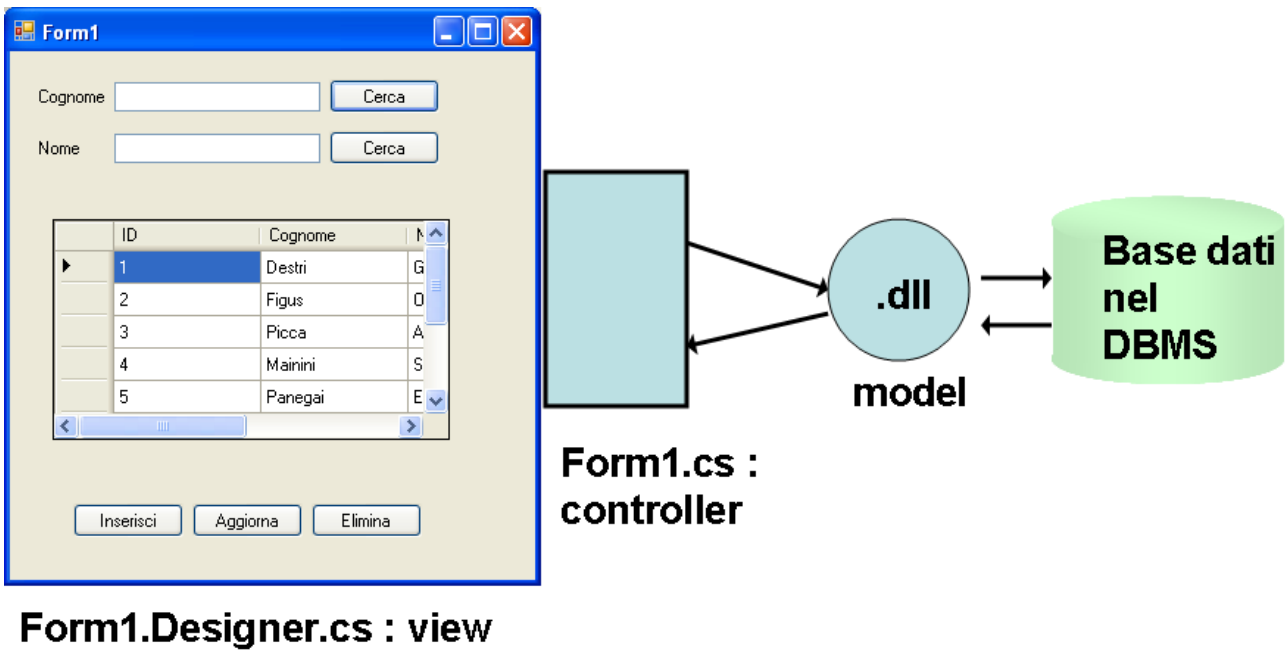

Figura 2: stratificazione di un applicativo .NET secondo AreaMVC. Il controller ha i metodi reattori corrispondenti a ciascuno dei controlli visuali presenti entro la view. Ciascuno di tali metodi richiama al suo interno il metodo del model che implementa effettivamente l'azione corrispondente, attraverso una azione sul DBMS o un filtraggio dei dati già presenti in memoria.

Implementazione di AreaMVC in Java

Java, con ormai 14 anni di storia alle spalle, è sicuramente una tecnologia matura. La sua collocazione nel mercato, ove è leader per le applicazioni Web enterprise e molto diffuso nell'ambito mobile, ha provocato la nascita di molti frame work aggiuntivi, come, ad esempio, Struts, che spesso fanno sì che lo sviluppo in Java avvenga esclusivamente entro tali frame work.

In questo paragrafo ci occuperemo della implementazione di AreaMVC in Java form base, senza altri componenti aggiuntivi e nella versione più semplice di Java Web, quella basata su Servlet e JSP.
Per quanto riguarda le Java form esistono tre librerie di base di controlli grafici, AWT, Swing e SWT, tra loro per la maggior parte incompatibili.
Inoltre una Windows form Java non è suddivisa nettamente tra view e controller, ma il codice corrispondente all'uno e all'altra sono all'interno della stessa classe.
Però è possibile in modo analogo a prima creare un progetto di libreria .Jar che corrisponde al model e poi includere la o le classi di servizio qui definite come attributi della Windows form.
Nel Java Web la situazione invece è molto analoga ad ASP.NET. Se si segue MVC-Model 2 di Sun i componenti sono:

- un file Java Server Page, con estensione .jsp, che corrisponde alla view e contiene gli elementi visuali della form;
- un file Servlet, con estensione .java, che corrisponde al controller e al cui interno sono presenti i reattori agli eventi Web raccolti dalla form HTML presente nella JSP avente il servlet come target; in versioni più evolute, come ad esempio Java Server Faces, la situazione è ancora più vicina a quella delle pagine ASP.NET: non è necessario agire sui messaggi e distinguere il controllo di provenienza, sono presenti dei veri e propri metodi reattori come quelli delle form;
- una o più classi di servizio istanziate in oggetti interni al servlet (la raccomandazione di Sun è di averne una sola, detta classe di front-end [12]), corrispondenti al model e provenienti, ad esempio, da una libreria esterna già usata in una applicazione form.

Un discorso analogo vale per i Web service, che possono essere ottenuti incapsulando la classe del model entro la classe che implementa il Web service, in modo analogo all'equivalente .NET.

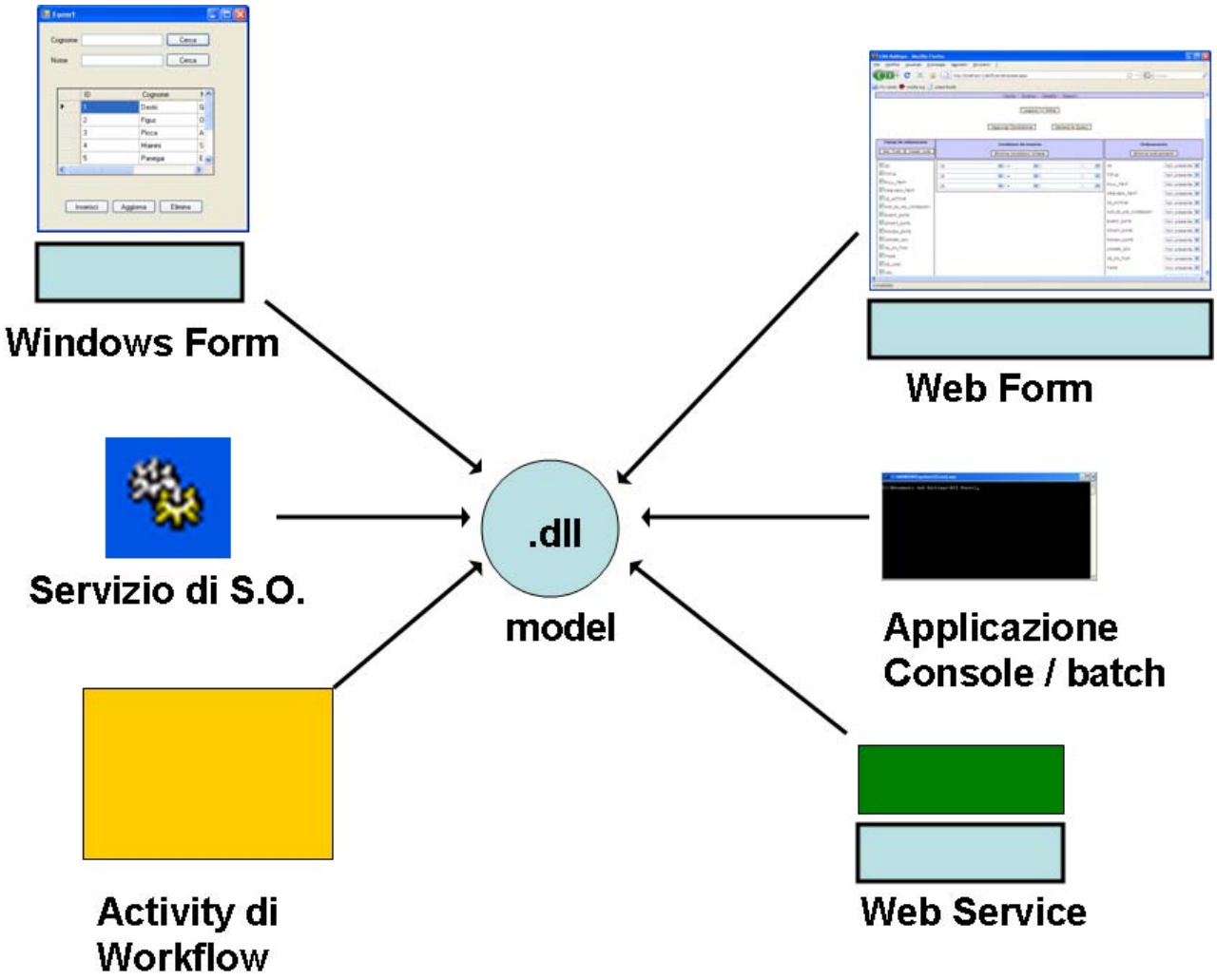

Figura 3: riuso dello stesso model in contesti diversi: Windows Form, Web Form, Web Service, servizio di sistema operativo, applicazione console o batch e attività di un Workflow.

La struttura interna del model

Fino a questo momento non è stata ancora definita la struttura interna del model. Nel contesto esaminato, il model svolge essenzialmente questi compiti distinti:

- memorizza entro strutture apposite i dati, presenti sotto forma di classi entità o strutture come le DataTable di .NET;
- effettua le operazioni di trasformazione/aggiornamento dei dati in memoria sulla base dei comandi ricevuti dal controller attraverso la sua interfaccia esterna;
- accetta nuovi dati a partire da quanto inserito dall'interfaccia utente;
- realizza le operazioni di lettura/scrittura verso il sistema di persistenza (tipicamente il DBMS relazionale o una struttura XML).

I dati, ovvero le classi entità o le DataTable, vengono trasmessi dalla interfaccia utente verso il model (inserimento) oppure restituiti dal model verso l'interfaccia utente (visualizzazione). In un'ottica di portabilità è conveniente usare strutture indipendenti dal DBMS (ad esempio, OleDB in ADO.NET o JDBC) ed incapsulare le parti specifiche entro elementi cambiabili a run-time (ad esempio, la stringa di connessione, sì da avere un model applicabile, a parità di schema sottostante, a diversi DBMS.

In ogni caso, a parità di interfaccia esterna del model, questo può contenere anche strutture molto complesse o molto semplici. Per esempio, il model può essere semplicemente un generatore di dati fissi, avente lo scopo di controllare il funzionamento corretto di view e controller in un contesto completo di ciclo di funzionamento. Oppure il model, come già accennato in precedenza, può essere un elemento complesso e contenere una cache vera e propria di dati, comprensiva di controlli di coerenza ed accesso concorrente, senza cambiare in modo rilevante la sua struttura esterna.

Per quanto riguarda le condizioni di errore che generano eccezioni deve essere definito in sede di design come e dove trattare tali eccezioni. Lo standard seguito in AreaMVC è quello di definire una classe eccezione (chiamata, ad esempio, AreaException), derivata da Exception e corredata di attributi accessibili attraverso proprietà o metodi getter e setter che descrivono completamente la situazione di errore. Tutte le eccezioni prodotte internamente al model sono catturate, le informazioni utili sono estratte da esse e aggiunte ad una istanza di AreaException, che viene poi lanciata esternamente ai metodi che il model espone. Nei metodi del controller è quindi necessario solo tenere presente il try-catch per AreaException.

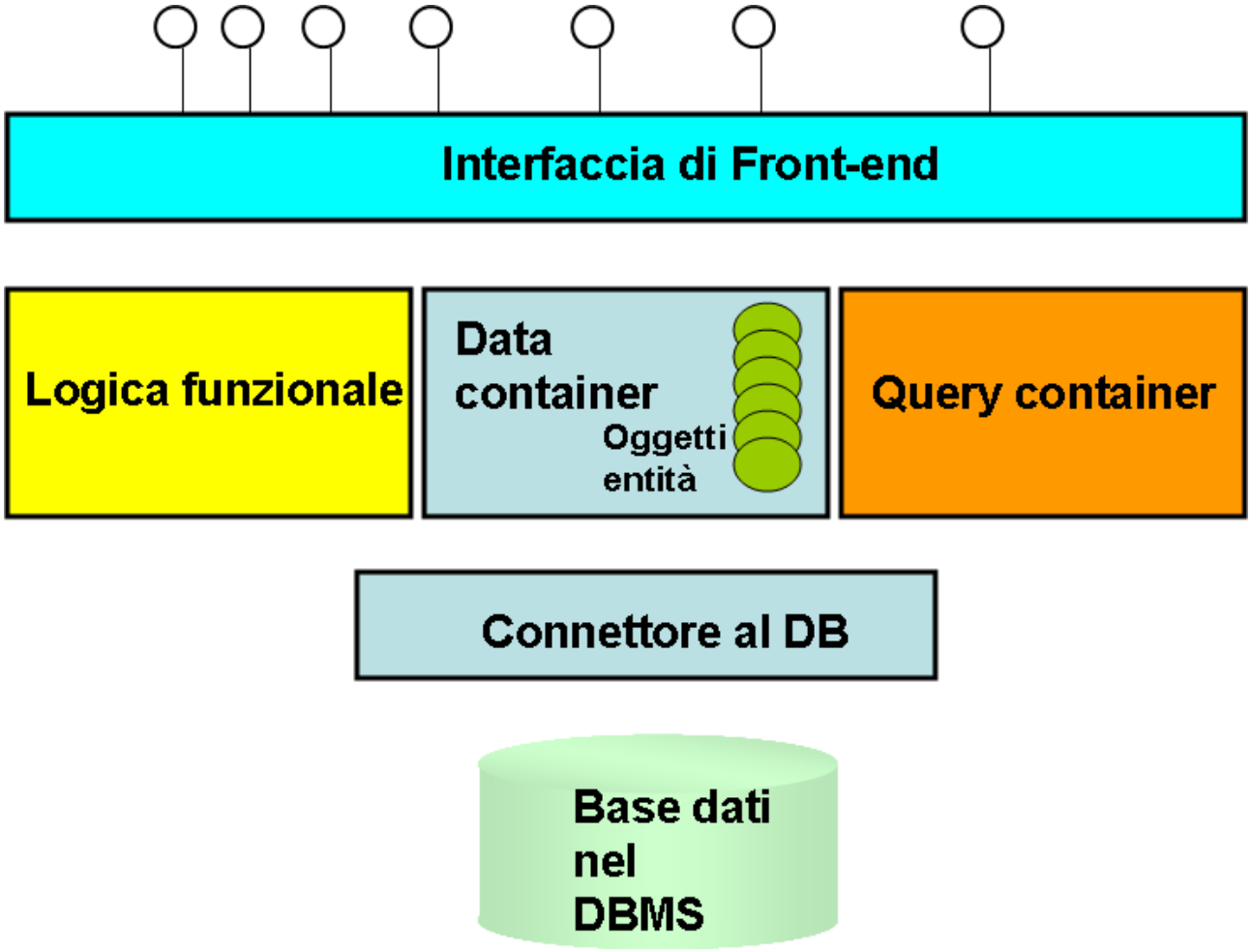

Figura 4: una possibile struttura interna per il model. La classe che forma l'interfaccia di front-end contiene gli altri elementi costitutivi del model, mascherandone completamente la struttura agli strati superiori.

Conclusioni

In questo articolo è stato dimostrato come le linee guida del pattern MVC siano facilmente applicabili nello sviluppo di applicativi con tecnologie Java e .NET. A prezzo di una complicazione della fase di design si ottengono però componenti di logica business realmente portabili e la trasformazione tra applicativi Windows form e Web è molto velocizzata.

Bibliografia

[1] Wikipedia e riferimenti esterni
Model–view–controller
http://en.wikipedia.org/wiki/Model-view-controller

[2] Mike Potel
MVP: Model-View-Presenter. The Taligent Programming Model for C++ and Java.
http://www.wildcrest.com/Potel/Portfolio/mvp.pdf

[3] Martin Fowler
Retirement note for Model View Presenter Pattern
http://www.martinfowler.com/eaaDev/ModelViewPresenter.html

[4] Riccardo Golia
Introduzione ai design pattern
http://msdn.microsoft.com/it-it/library/cc185081.aspx

[5] Beth Stearns et al.
Designing Enterprise Applications with the J2EE Platform
http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html

[6] Martin Fowler
Patterns of Enterprise Application Architecture
Ed. Addison-Wesley Professional, 2002

[7] Andrea Saltarello et al.
Northwind Starter Kit
http://www.codeplex.com/NSK

[8] Area Solutions Providers
La libreria AreaFramework
http://www.areasp.com/soluzioni_pacchettizzate.htm#AreaFramework

[9] Wikipedia
Partial Class
http://en.wikipedia.org/wiki/Partial_class

[10] Corrado Cavalli
Introduzione a Windows Presentation Foundation
http://msdn.microsoft.com/it-it/library/cc185038.aspx

[11] A.A.V.V.
Windows Workflow Foundation General Reference
http://msdn.microsoft.com/it-it/library/ms732093.aspx

[12] A.A.V.V.
Servlets and JSP Pages Best Practices
java.sun.com/developer/technicalArticles/javaserverpages/servlets_jsp/