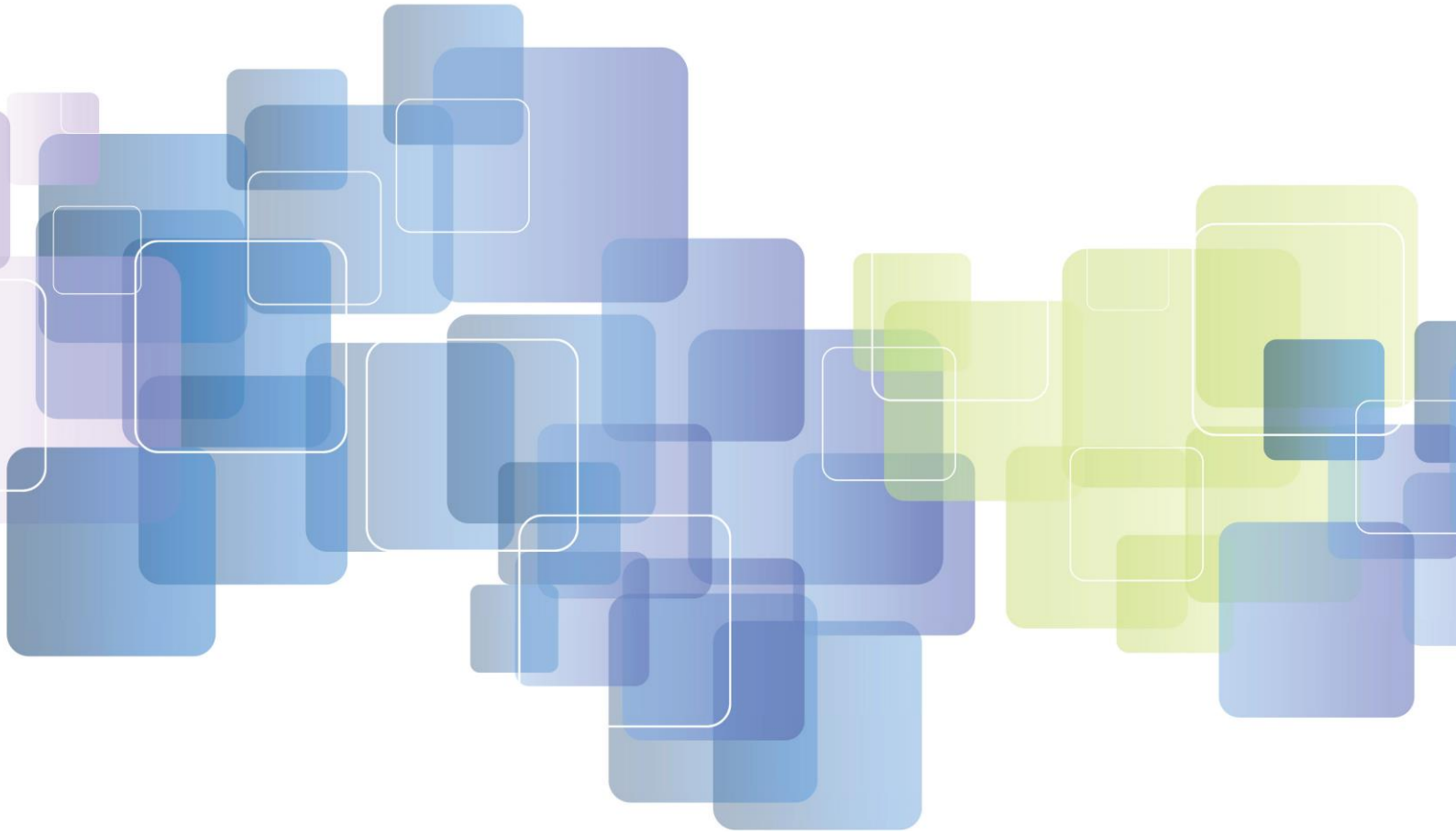
TecnetDati

Sviluppo di applicazioni web con il pattern Model-View-Controller

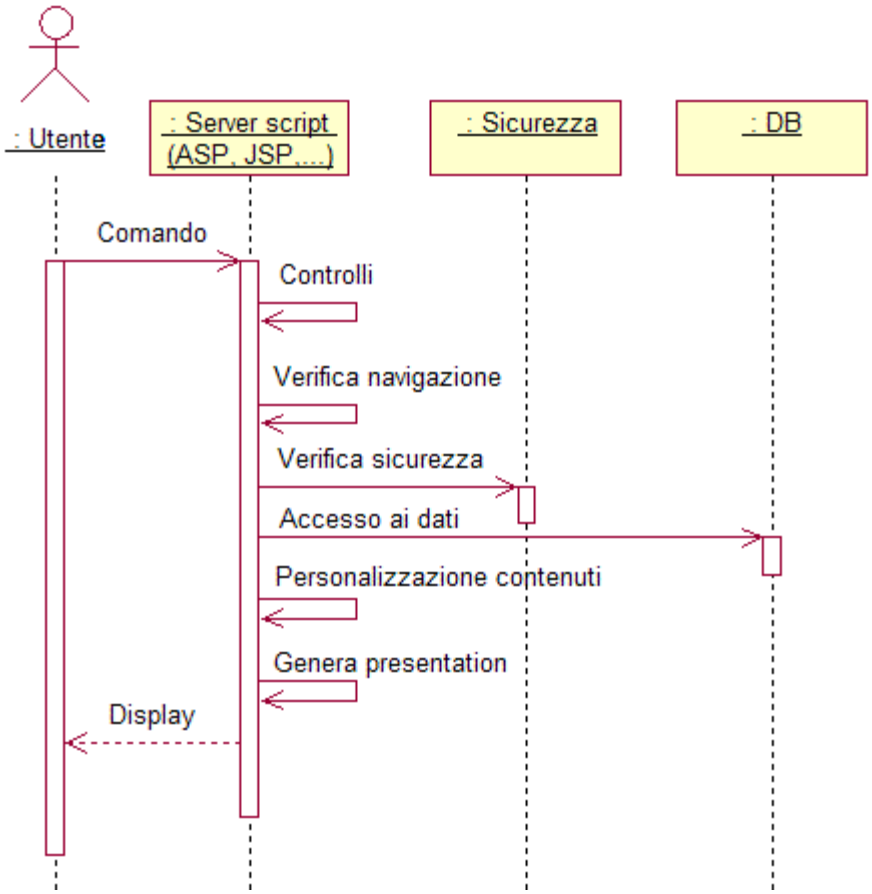
Gabriele Pellegrinetti



MVC: come funziona e quali sono vantaggi che derivano dal suo utilizzo?

La grande diffusione della tecnologia Internet ha spinto le aziende a realizzare un numero sempre maggiore di applicazioni basate su tecnologia web o su tecnologie correlate, per esempio wireless. Molte delle applicazioni web esistenti, però, sono state realizzata senza fare uso di particolari pattern di sviluppo ma lasciando libero lo sviluppatore di scegliere la metodologia ad esso più congegnale. Questo ha portato a realizzare applicazioni costituite da componenti monolitiche che si occupano della gestione della user interface, dell'elaborazione e del reperimento delle informazioni da database o da altri sottosistemi (es. mailnframe). Supponiamo ad esempio che sia stata realizzata una generica applicazione per la gestione di un catalogo di prodotti che mette a disposizione dell'utente tre funzioni principali: la ricerca di un prodotto nel catalogo, la visualizzazione dell'elenco dei prodotti selezionati e la visualizzazione delle informazioni di dettaglio di un prodotto. La prassi che viene seguita normalmente nella creazione di un'applicazione consiste nel creare tre componenti software ognuna delle quali gestisce una delle funzionalità dell'applicazione.

Il funzionamento di una generica applicazione web, così costruita, è descritto nel seguente diagramma di sequenza:



dove si puo osservare che la maggior parte delle responsabilità sulla gestione delle applicazioni è detenuta dall'oggetto ":Server script".

Osservando le figure precedenti si possono evidenziare i seguenti difetti di progettazione:

- ognuna delle tre componenti monolitiche implementa al suo interno una parte dei controlli di sicurezza, delle funzioni di personalizzazione, delle funzioni di visualizzazione e del controllo di navigazione. Questo può influire negativamente nella successiva evoluzione dell'applicazione e nella sua manutenzione. Ad esempio, la nascita di nuovi requisiti di sicurezza, obbliga lo sviluppatore a modificare le funzioni per la gestione della sicurezza all'interno di ognuna delle tre componenti costringendolo a trovare la soluzione migliore per mantenere allineate le funzionalità. Lo stesso dicasi per la gestione delle funzioni di navigazione e di personalizzazione
- le componenti applicative realizzate non sono facilmente portabili in altre applicazioni.

Supponiamo poi che la nostra applicazione richieda la presenza di front-end diversi, in quando si vuole realizzare un'applicazione multicanale con le seguenti caratteristiche:

- si vuole implementare un front-end in tecnologia web (pagine HTML) in modo da consentire agli utenti di accedere all'applicazione da una qualsiasi macchina su cui sia presente un browser.

- si vuole consentire agli utenti mobili di accedere all'applicazione tramite un terminale wap (pagine WML) presente all'interno di un telefonino dell'ultima generazione o all'interno di un computer palmare.
- le funzioni di amministrazione dell'applicazione devono essere fruite in modalità client/server tramite un'interfaccia utente realizzata in tecnologia windows.
- l'applicazione deve fornire un'interfaccia di comunicazione verso altre applicazioni e verso altri sottosistemi tramite l'utilizzo del linguaggio XML.

Dalla lettura dei punti precedenti si comprende subito che la realizzazione di componenti monolite non è auspicabile in quanto:

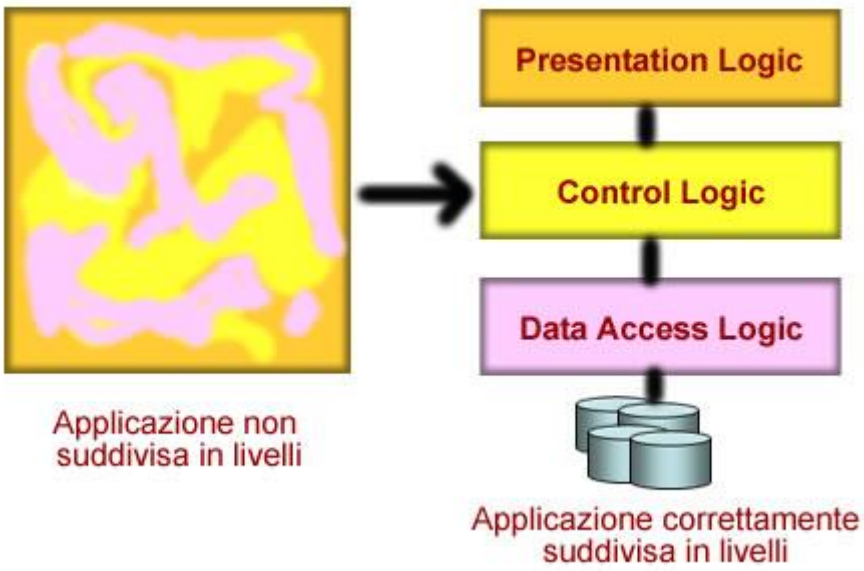
- In un'applicazione web l'interfaccia utente tende a cambiare più velocemente rispetto alle funzioni di business o di accesso ai dati in quanto i requisiti di un sito o di una applicazione web sono estremamente dinamici e richiedono di adattare la user interface alle esigenze del mercato, sia variandone l'aspetto estetico, sia modificando le funzionalità esistenti, sia aggiungendone di nuove. Le stesse interfacce possono poi richiedere visualizzazioni diverse a seconda della tipologia di utente connessa al sistema.
- Le applicazioni web di nuova generazione vengono progettate molto spesso in un' ottica multicanale, dove la stessa applicazione deve essere fruita tramite dispositivi diversi come il browser web, un computer palmare, un sistema client server o un telefonino di nuova generazione. Per tale motivo si rende necessario implementare una separazione delle funzioni di visualizzazione dalle funzioni di calcolo, trasformando queste ultime in un insieme di componenti che forniscono servizi al livello superiore. In tal caso l'aggiunta di una nuova user interface, o la modifica di una user interface esistente, non impatta sulla logica applicativa (e viceversa, a patto di non modificare le modalità di chiamata delle componenti applicative).
- Il disegno delle pagine web richiede figure professionali diverse da quelle necessarie per progettare e implementare le componenti applicative o di accesso ai dati. Una separazione delle componenti consente di ottimizzare l'uso delle risorse e la distribuzione dei compiti in fase di progetto e realizzazione di un'applicazione.

- Il codice relativo alla user interface tende a essere più "device-dependent" del codice della business logic. Una corretta separazione delle componenti semplifica eventuali migrazioni su front-end diversi e minimizza il rischio di introdurre errori di conversione.
- Per motivi di sicurezza è bene non inserire, all'interno di componenti per la generazione di user interface, logica di business o di accesso ai dati in quanto una compromissione delle prime può provocare gravi falle di sicurezza nelle seconde.

Suddivisione dell'applicazione in livelli e pattern MVC

La metodologia corretta da applicare per lo sviluppo di applicazioni web, richiede la suddivisione delle componenti applicative in tre distinti layer comunicanti fra loro tramite un'interfaccia applicativa:

- il layer di **presentation logic**, che si occupa dell'interazione con l'utente e della generazione della user interface;
- il layer di **control logic**, che si occupa di controllare il flusso delle operazioni e di eseguire l'elaborazione delle informazioni;
- Il layer di **data access logic**, che si occupa di fornite le funzioni per accedere al database e agli altri sottosistemi applicativi.



Un'applicazione suddivisa in livelli funziona nel seguente modo:

1. l'utente interagisce con la presentation logic agendo sulle mappe delle user interface

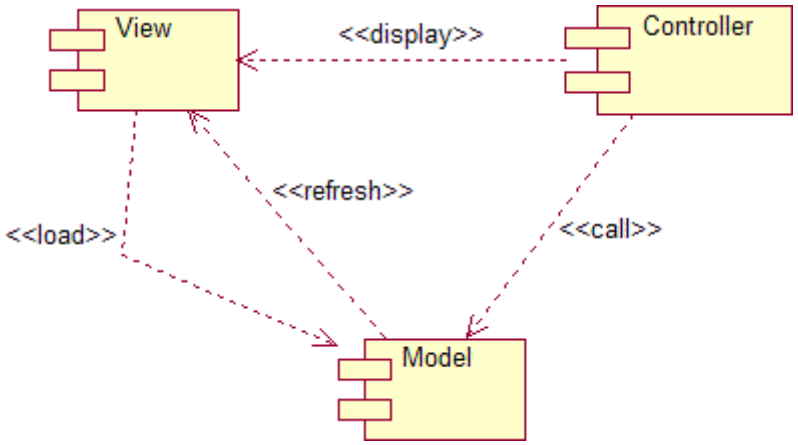
2. la presentation logic fornisce alla control logic gli input forniti dall'utente
3. la control logic elabora gli input forniti dall'utente.
4. nel caso di interazione con i database, la control logic richiede alla data access logic di eseguire delle query o di salvare i dati su DB.
5. la data access logic fornisce alla control logic le informazioni richieste
6. la control logic esegue, se necessario, l'elaborazione delle informazioni e fornisce il risultato alla presentation logic
7. la presentation logic formatta l'output e presenta all'utente le mappe con il risultato dell'elaborazione.

Fra i molti pattern che possono essere utilizzati per implementare questa suddivisione, nello sviluppo di applicazioni web il più utilizzato è il **Model-View-Controller (MVC)**. Si tratta di un pattern nato negli anni 80 per consentire agli sviluppatori in linguaggio Smalltalk di realizzare applicazioni Object Oriented con la possibilità di disaccoppiare le componenti di presentation e di controllo dalle componenti di accesso ai dati. Questo pattern è sempre stato "eluso" dagli sviluppatori per cui non ha mai avuto la diffusione che si meritava. Solo con l'avvento delle applicazioni web scritte in Java, grazie ad una politica di diffusione dei pattern proposta dalla SUN, MVC ha iniziato ad essere utilizzato in modo massivo.

La versione originale di MVC, discussa in modo dettagliato nel testo "*Application programming in Smalltalk 80: how to use Model-View-Controller [Burbeck92]",* descrive l'interazione di tre diverse tipologie di componenti che risultano essere fra loro disaccoppiate:

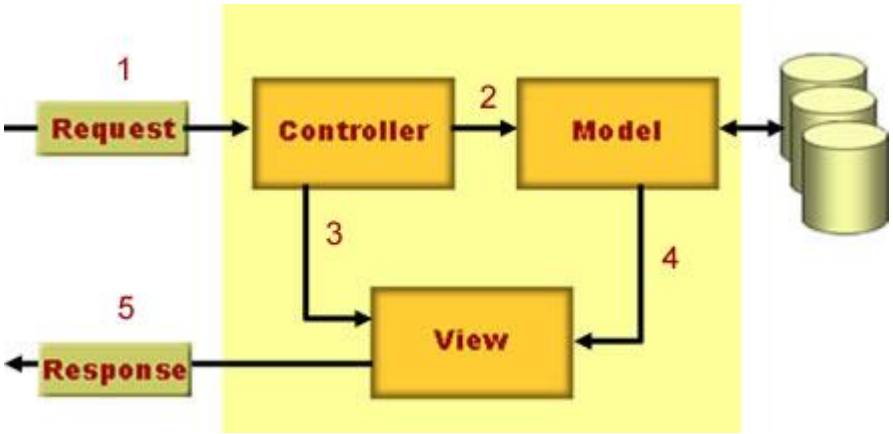
- il **model** rappresenta i dati e le regole di business che gestiscono l'interazione con i dati stessi ed espone al controller e alla view le funzioni per accedervi. L'unico responsabile dell'accesso ai dati è il model per cui ogni richiesta di accesso o di manipolazione dei dati stessi deve essere eseguita attraverso di esso. A seconda della variante del pattern utilizzata, il model può avere la responsabilità di notificare alla view l'eventuale variazione dei dati presenti all'interno del model stesso. Corrisponde al livello di data access logic descritto in precedenza.
- la **view** implementa la logica di presentazione e propone all'utente (o ad un altro sottosistema), sotto una certa forma (mappe, report, ...), i dati forniti dal modello. La view si occupa pure di filtrare gli input dell'utente e di inviarli all'oggetto di controllo. Nella versione originaria del pattern è responsabile del mantenimento della consistenza della presentation in base ai cambiamenti del modello.
- il **controller** traduce gli input dell'utente, eseguiti attraverso la view, in azioni che possono essere eseguite dal model. Successivamente, in base agli input dell'utente e al contenuto dei dati, il controller seleziona la view più appropriata per la visualizzazione dei dati stessi. Tramite le operazioni da esso eseguite il controller implementa la logica di controllo dell'applicazione.

Le dipendenze fra le tre componenti sono descritte nel seguente diagramma UML:



mentre le loro interazioni possono essere schematizzate nel seguente modo:

1. l'utente invia un comando al sistema premendo, ad esempio, un pulsante su una mappa. Tramite questa azione l'utente interagisce con la view che intercetta gli input forniti dall'utente. La view genera quindi una **request** che viene inviata all'oggetto di controllo (controller).
2. il controller, in base agli input dell'utente, richiede al model di eseguire una determinata operazione (in pratica traduce le richieste dell'utente in azioni eseguite dal model) che può provocare un cambiamento di stato del model stesso (ad esempio viene eseguita un'operazione di update sui dati). Nel diagramma UML questa operazione è rappresentata dalla dipendenza << call >>.
3. il controller, in base alle richieste dell'utente e allo stato del model, seleziona view più opportuna per presentare i dati. Nel diagramma UML questa operazione è rappresentata dalla dipendenza << display >>.
4. la view recupera i dati dal model. Questa operazione può essere eseguita o richiamando un metodo del model stesso (ma in tal caso si riduce il disaccopiamento fra le componenti) o utilizzando delle strutture dati intermedie nelle quali il modello scrive le informazioni dopo averle elaborate in modo che la view possa successivamente recuperarle. Nel diagramma UML questa operazione è rappresentata dalla dipendenza << load >>.
5. la view genera la presentazione dei dati e la invia all'utente (**response**).



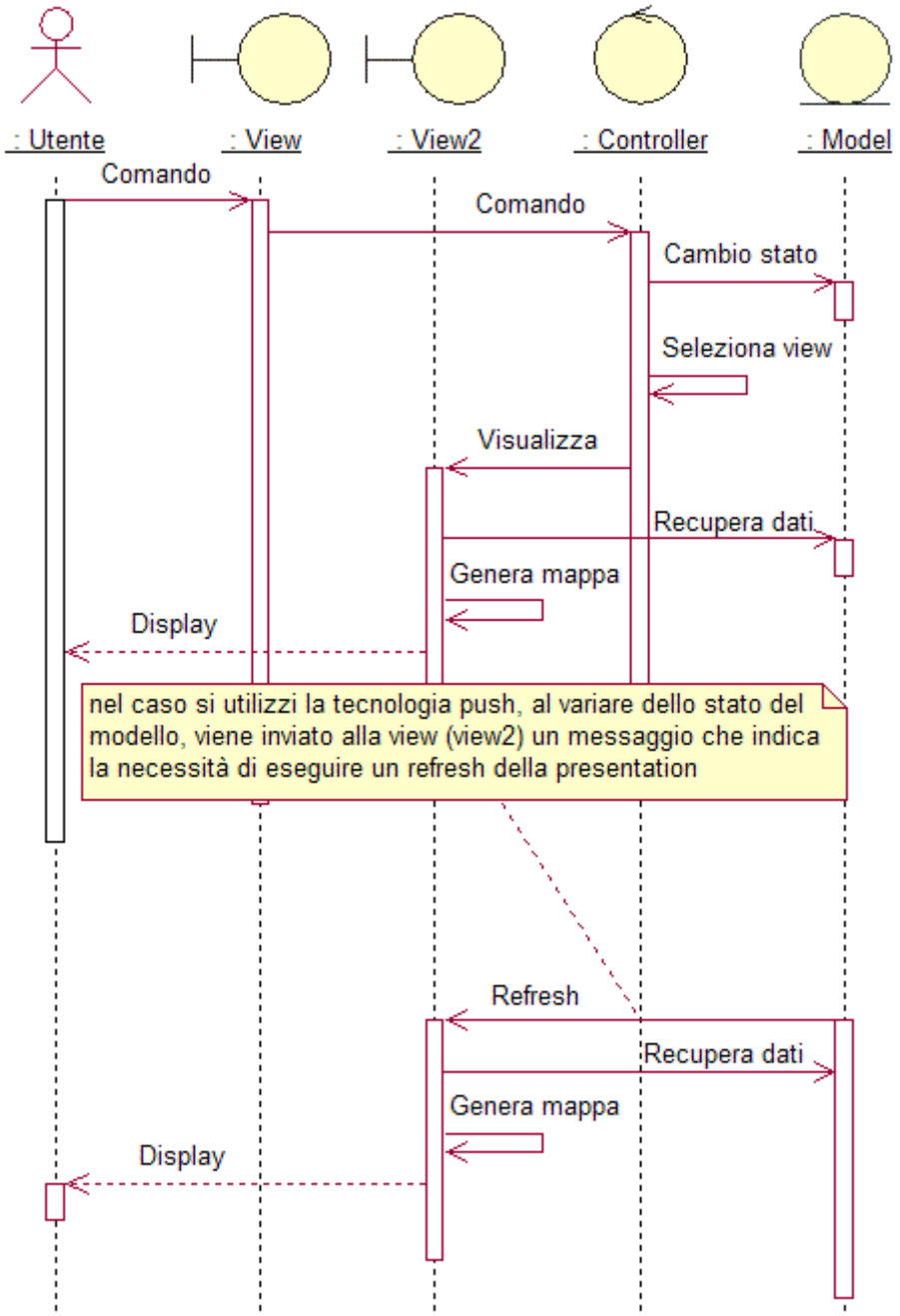
Nella descrizione delle componenti che costituiscono il pattern MVC abbiamo detto che la view è responsabile di mantenere la coerenza della presentazione dei dati all'utente rispetto alla variazione degli stessi all'interno del model.

Per eseguire questa operazione sono possibili due modalità:

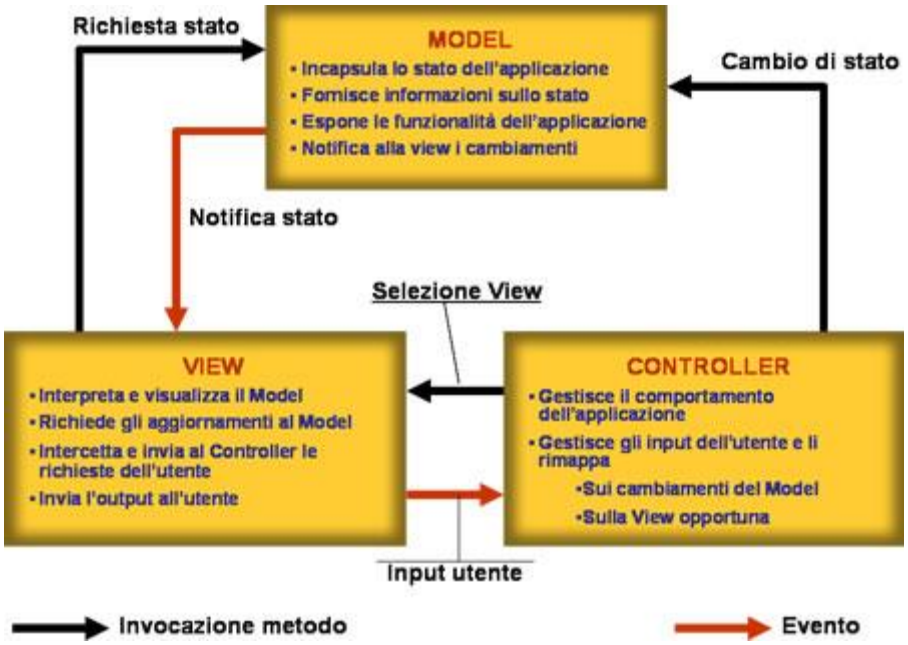
- la modalità **push** (detta anche a **model attivo**) dove, al verificarsi di un cambiamento di stato nel model, viene generato un evento che indica alla view che è necessario eseguire un refresh della presentation. Nel diagramma UML la generazione dell'evento è rappresentata dalla dipendenza << refresh >>. La view richiede quindi al model i dati aggiornati (<< load >>) e li presenta all'utente.
- la modalità **pull** (detta anche a **model passivo**) dove la view stessa, quando lo ritiene più opportuno, provvede a richiedere al model i dati aggiornati (<< load >>). Questo secondo meccanismo non garantisce l'aggiornamento in tempo reale della presentation con il variare dei dati presenti nel model.

Attualmente, a causa del meccanismo di funzionamento del protocollo HTTP, non è possibile utilizzare la modalità push nella realizzazione di applicazioni web.

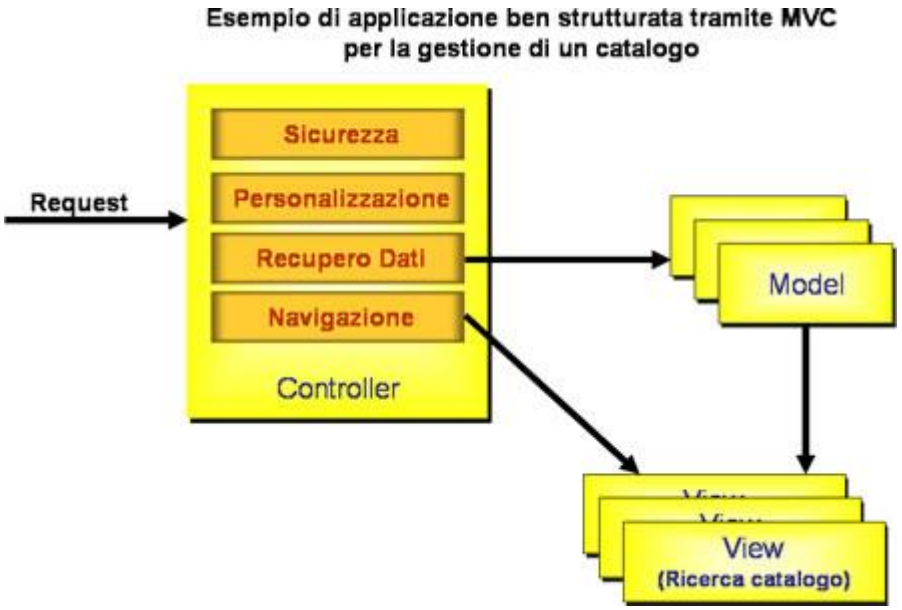
Il diagramma di sequenza sotto riportato mostra le interazioni che occorrono fra le componenti per l'esecuzione di un comando generico. Si suppone che l'utente interagisca su una mappa iniziale (generata dall'oggetto view) e che l'output del comando venga visualizzato su una seconda mappa (view2) selezionata dal controller in base al comando impartito.



Come si può vedere dall'invocazione dei metodi, le responsabilità sono ben delineate e distribuite fra model, view e controller (a differenza della situazione descritta inizialmente per le componenti monolitiche). Queste sono riassunte dal seguente schema:



In base alle considerazioni fatte fino ad ora, l'ipotetica applicazione per la gestione di un catalogo, descritta nell'introduzione, può essere progettata per componenti realizzando un oggetto di controllo che si occupa di gestire le funzioni di sicurezza, di navigazione e di personalizzazione e di coordinare, in base agli input, le operazioni di accesso ai dati (in genere un model per ogni entità) e la visualizzazione di tali informazioni (view). Il risultato è riassunto nel seguente schema.

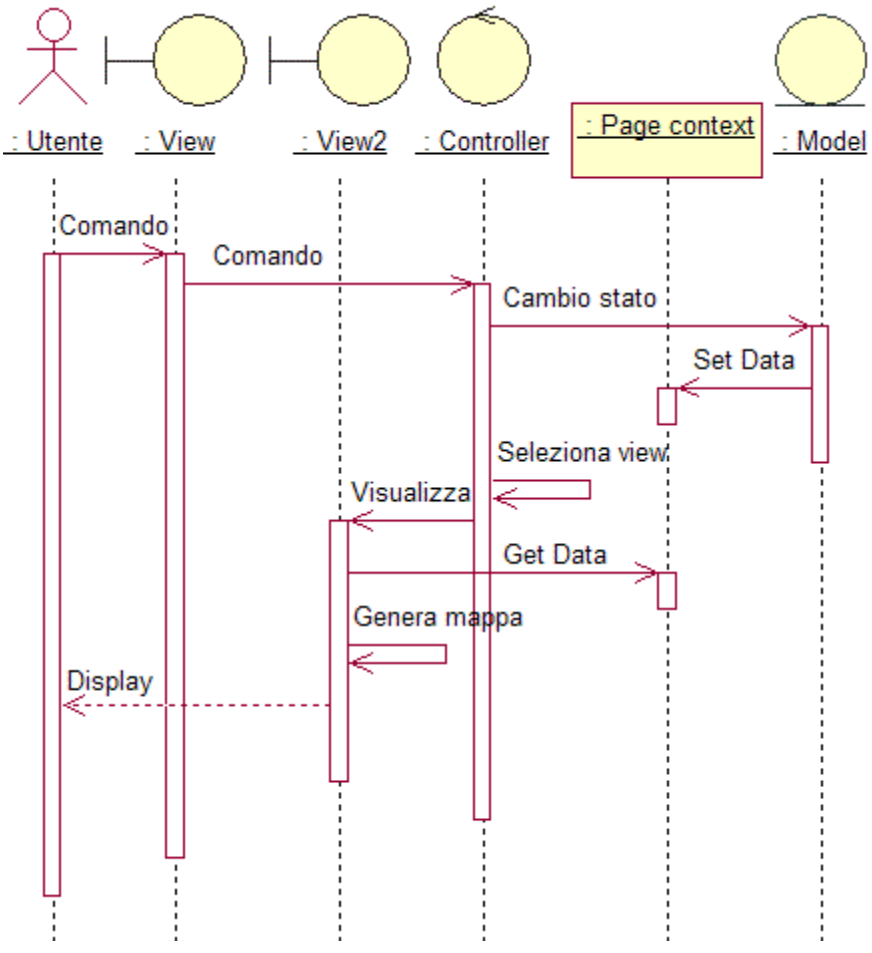


Il disaccoppiamento dei livelli di presentation, control e data access logic consente di realizzare applicazioni dove è possibile ottimizzare il riutilizzo del codice e agevolare la manutenzione del software. Ogni livello non deve conoscere la modalità di implentazione degli altri, ma deve solo conoscere le interfacce che consentono la comunicazione fra le componenti. In tal modo è possibile variare l'implementazione interna di una componente, ad esempio per ottimizzare le sue funzionalità, senza dover modificare il resto dell'applicazione. Oppure è possibile inserire una nuova modalità di presentation senza dover modificare la control e la data access logic.

Variazioni al pattern MVC standard

In letteratura è possibile trovare numerose versioni del pattern stesso che si differenziano dalla versione originale essenzialmente per le diverse responsabilità attribuite al controller o per il maggiore o minore disaccoppiamento esistente fra le componenti.

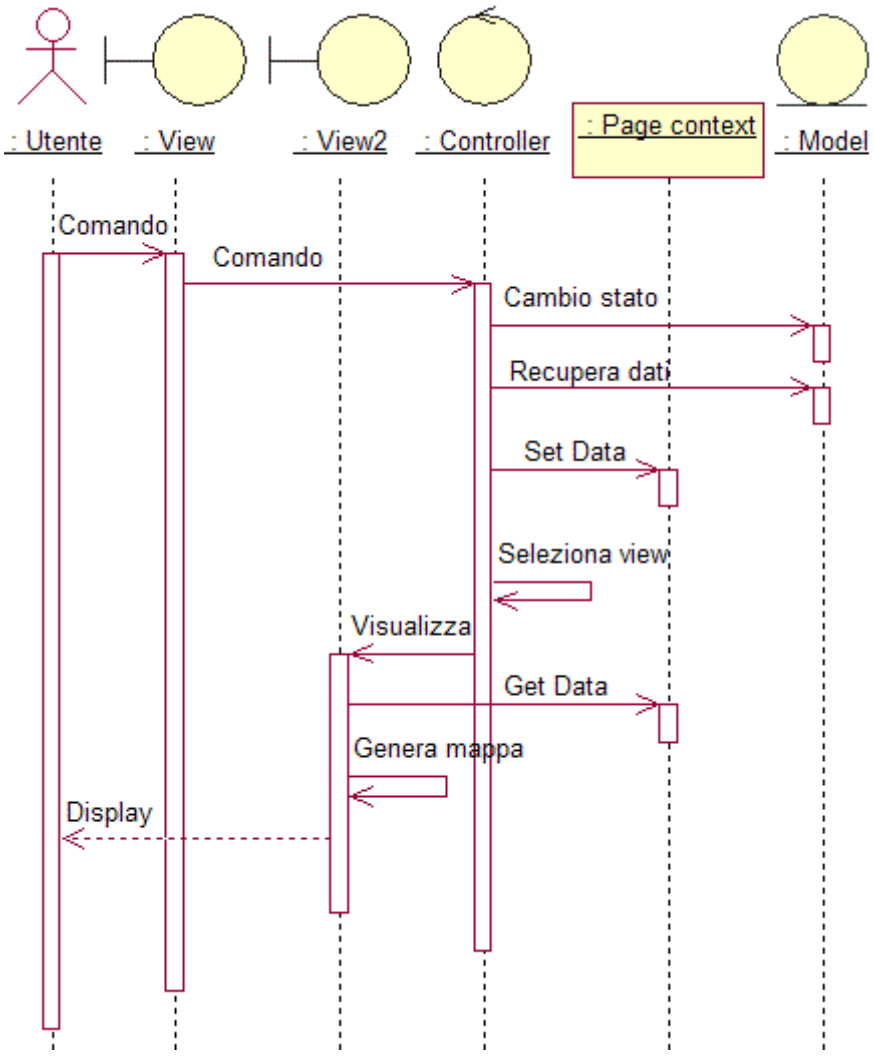
In un'applicazione web è possibile disaccoppiare ulteriormente le funzioni di presentation dalle funzioni di accesso ai dati evitando di richiamare dalla view i metodi del model. Questo avviene introducendo un oggetto intermedio per il passaggio dei dati. Per ottenere questo scopo è possibile utilizzare le variabili di page context (request, session, application) fornite dalle principali tecnologie di sviluppo (ASP, JSP, PHP). In questo modo, per la view, è sufficiente conoscere la struttura dei dati memorizzati nel page context senza dover conoscere l'interfaccia di comunicazione con il model. Lo svantaggio di questa soluzione è che, per un gran numero di request contemporanee o per dati di una certa dimensione, l'utilizzo di un page context può rivelarsi un collo di bottiglia in quanto viene introdotto nel sistema un over-head dovuto all'inserimento e alla lettura delle informazioni in questo oggetto aggiuntivo.



Un'altra variante, molto diffusa, dell'MVC si distingue per una diversa attribuzione della responsabilità del recupero delle informazioni dal modello. In questa variante si dà una

maggior importanza al controller e si evita che la view debba interagire direttamente con il model.

E' il controller, infatti, che dopo aver mandato in esecuzione i comandi sul model, recupera da quest'ultimo i dati per poi "passarli" (in qualche modo) alla view. Il passaggio dei dati dal model alla view può utilizzare, come nel caso precedente, le variabili di page-context. Con questa soluzione si ha il massimo disaccoppiamento fra presentation logic e data access logic, ma si ha come risultato una maggiore complessità dell'oggetto di controllo.



Tecnet Dati s.r.l.
C.so Svizzera 185 –
10149 - Torino (TO), Italia
Tel.: +39 011 7718090 Fax.: +39 011 7718092
P.I. 05793500017 C.F. 09205650154
www.tecnetdati.com