利用 UML 建立 Web 應用程式架構模型

Jim Conallen

Rational Software 白皮書

TP 157, 6/99

此資料的版本出現在 Communications of the ACM 的 1999 年 10 月期刊（第 42 冊，編號 10）。

Rational®
the software development company

目錄

摘要

Web 應用程式變得越來越複雜且任務重大。爲協助應付此複雜性，它們需要建模。「統一建模語言 (UML)」用於軟體加強系統建模的標準語言。嘗試以 UML 建立 Web 應用程式模型時，顯然它的一些元件不太適合標準 UML 建模元素。爲了承擔整個系統的一個建模表示法（Web 元件和傳統中層元件），必須延伸 UML。本白皮書使用正式延伸機制來呈現 UML 的延伸規格。此延伸規格是設計成讓 Web 特定元件能夠與其餘系統模型整合，並可顯現出適合 Web 應用程式的設計師、執行者和架構師的適當摘要和詳細資料層次。

概觀

在過去幾年，有一個新詞彙出現在 IT 詞彙中：Web 應用程式。看起來每一個與商業軟體系統有關的人都計劃建置 Web 應用程式，許多非商業相關軟體人員也感興趣。對於此架構的許多初期採用者而言，Web 應用程式這個詞彙就像系統本身，已從小型成功的網站附加元件發展成健全的 n 層應用程式。一個 Web 應用程式同時服務數萬個分散在世界各地的使用者並不奇怪。架構 Web 應用程式非同小可。

在接受考驗時，Web 應用程式一詞對於不同的人具有不太相同的意義。有些人認爲 Web 應用程式就是使用 Java 的應用程式；有些人認爲 Web 應用程式是使用 Web 伺服器的應用程式。而介於這兩者之間，才是一般人的共識。基於本白皮書的用途，我們粗略地將 Web 應用程式定義爲 Web 系統（Web 伺服器、網路、HTTP、瀏覽器），其中使用者輸入（瀏覽和資料輸入）將影響商業狀態。此定義嘗試建立這樣的定義：Web 應用程式是一個有商業狀態的軟體系統，它的「前端系統」大部分是透過 Web 系統交付的。

Web 應用程式的一般架構是用戶端伺服器系統的架構，但有幾項顯著的不同。 Web 應用程式最大優點之一是它的部署。部署 Web 應用程式通常就是在網路上設定伺服器端元件。在用戶端那邊並不需要特殊軟體或配置。另一個重大差異是用戶端和伺服器通訊的本質。Web 應用程式的主要通訊協定是 HTTP，它是無連線通訊協定，其設計目的在於健全及容錯，而非最大通訊產量。在 Web 應用程式中，用戶端與伺服器之間的通訊通常是以網頁的導覽爲中心，而不是伺服器端與用戶端物件之間直接通訊。在某種抽象層次上，Web 應用程式中的所有傳訊都可以描述爲網頁實體的要求和接受。一般而言，Web 應用程式的架構與動態網站的架構沒什麼差別。

Web 應用程式和動態網站之間的差異與其用法有關。Web 應用程式實作商業邏輯，而其使用則變更商業的狀態（如系統所擷取）。這點很重要，因爲它定義建模工作的焦點。Web 應用程式執行商業邏輯，因此，系統最重要模型的焦點是在於商業邏輯和商業狀態，而非呈現方式的細節。然而，呈現方式也很重要（否則系統對人一無是處），您應該致力於在商業和呈現問題之間的劃分。如果呈現問題很重要，甚至更複雜，則也應該塑造它們，但不像商業邏輯模型的整體部分那麼必要。另外，處理呈現方式的資源傾向於更具藝術性，與商業規則的實作較無關係。

關係管理方法（Relationship Management Methodology，RMM）是與 Web 系統的開發相關聯的方法/表示法之一。RMM 是企業內部網路和網際網路 Web 系統的設計、建構和維護的方法。它的主要目標是要減少動態資料庫導向網站的維護成本。它提倡以系統的視覺化表示法來促進設計上的討論。它是一種反覆式流程，包括將網頁視覺化元素及這些元素與資料庫實體的關聯加以分解。RMM 在動態網站的建立和維護上一種「一應俱全」的方法。

RMM 在建置 Web 應用程式上較不符合要求。Web 應用程式是以商業邏輯爲中心，包括一些實作商業邏輯的技術性機制，RMM 表示法對這些機制並未面面俱到。像用戶端 Scripting、Applet 和 ActiveX 控制項之類的技術，對系統商業規則的執行通常有很大的貢獻。另外，Web 應用程式也可以作爲分散式物件系統的交付機制。 Applet 和 ActiveX 可包含一些元件，它們獨立於 Web 伺服器之外，透過 RMI 或 DCOM 與伺服器端元件非同步互動。更準確的應用程式也在用戶端上利用多個瀏覽器實例和頁框，來建立及維護自己的通訊機制。

由於所有這些機制促成系統的商業邏輯，所以它們本身需要建模。另外，由於它們只代表商業邏輯的一部分，因此，需要與剩餘系統模型整合。在許多狀況中，商業邏輯的主體是在其中一個伺服器端層級的 Web 伺服器背後執行。建模語言和表示法的選擇通常由這一端的應用程式需求來決定。在 OMG 接受以 UML 作爲正式物件建模語言之後，有越來越多的系統是以 UML 表示法來表示。對於許多人而言，UML 是軟體密集系統建模的理想語言。因此，Web 應用程式建模的主要問題變成：我要如何表示在 Web 特定元件中執行的商業邏輯，來與我的應用程式剩餘部分並排靠攏？答案在於我們表達以 UML 在那些 Web 特定元素和技術中執行系統商業邏輯的能力。

本白皮書的用意就是要介紹 Web 應用程式建模的問題和可能的解決方案。它的焦點放在 Web 應用程式特定的架構重要元件以及如何以 UML 建模。它已假設讀者熟悉 UML、物件導向主體和 Web 應用程式開發。本白皮書所描述的工作是以一些比較無害的假設爲基礎：

- Web 應用程式是日益複雜的軟體密集系統，它們讓自己扮演更多重要的角色。
- 管理軟體系統複雜性的方法之一是將它們做成摘要及建模。
- 軟體系統通常含有多個模型，每一個代表不同觀點、摘要層次和詳細資料。
- 適當的摘要和詳細層次，視開發流程中的構件和活動而定。
- 軟體密集系統的標準建模語言就是統一建模語言（Unified Modeling Language, UML）。

本白皮書所表達的一些概念和想法，在即將發行的 Building Web Applications with UML 這本新書中有更完整的論述，該書預計在今年後半年由 Addison Wesley Longman 以 Object Technology Series 發行。

建模

模型簡化一些細節，幫助我們瞭解系統。建模內容的選擇對於問題的瞭解和解決方案的製作具有重大影響。 Web 應用程式和其他軟體密集系統一樣，通常是由一組模型來代表：使用案例模型、實作模型、部署模型、安全模型等等。 Web 系統專用的另一個模型是網站對照表，它是網頁的摘要和整個系統的導覽路徑。

目前實作的大部分建模技術很適合 Web 應用程式各種模型的開發，不需要進一步討論。然而，有一個非常重要的模型，即分析/設計模型（Analysis/Design Model，ADM），當它嘗試包含網頁及其相關聯的執行碼，要與模型中的其他元素並排靠攏時，的確遇到一些瓶頸。

在決定如何建模時，決定正確摘要和詳細層次是爲模型使用者帶來好處的重要關鍵。一般而言，建模的對象最好是系統構件 — 這些是要建構及操作來產生最終產品的真實實體。建模對象若是 Web 伺服器的內容或 Web 瀏覽器的詳細資料，對 Web 應用程式的設計師和架構師並無幫助。網頁、彼此的鏈結、用來建立網頁的所有動態內容和一度出現在用戶端上的所有動態內容，爲它們建模很重要—而且非常重要。設計師所設計、執行者所實作的就是這些構件。用戶端和伺服器上的網頁、超鏈結和動態內容就是需要建模的對象。

下一步是要將這些構件對映到建模元素。例如，超鏈結自然對映到模型中的關聯元素。超鏈結代表網頁之間的導覽路徑。由此看來，網頁可對映到模型邏輯視圖中的類別。如果網頁是模型中的一個類別，則網頁的 Script 將自然對映到該類別的作業。Script 中的任何網頁範圍變數將對映到類別屬性。若網頁包含一組可在伺服器執行的 Script（準備網頁的動態內容），以及一組完全不同且只能在用戶端執行的 Script（即 JavaScript），則會發生問題。在此情境中，當我們查看模型中的網頁類別時，伺服器上有哪些作用中的作業、屬性、甚至關係（正在準備網頁時），以及使用者與用戶端的網頁互動時有哪些又在作用中，我們對此感到十分困惑。此外，在 Web 應用程式中呈現的網頁，實際上最好建模成系統的元件。只將網頁對映到 UML 類別不能幫助我們更瞭解系統。

UML 的建立者瞭解，使用立即可用的 UML 來擷取特定網域或架構的相關語意是不夠的。爲了因應這個用途，已定義正式延伸延伸規格，讓執行者可以延伸 UML 的語意。此機制可讓我們定義*造型*、*標示值*和*限制*，它們可套用到模型元素。

*造型*是一種裝飾，可讓我們定義建模元素的新語意。*標示值*是鍵值對，可與建模元素相關聯，讓我們能夠在建模元素上「標示」任何值。*限制*是定義模型完善格式的規則。可以用開放式文字或更正式的物件限制語言（Object Constraint Language，OCL）來表示它們。

本白皮書討論的工作是針對 Web 應用程式引進 UML 的延伸規格。從整體上來看，此延伸規格超出本白皮書的範圍，然而，這裡仍會討論大部分概念和解釋。

建模的最後一個重點—商業邏輯和呈現邏輯之間需要清楚劃分。對於典型商業應用系統，只有商業邏輯才是 ADM 的一部分。像動畫按鈕、滑鼠越過時的說明和其他使用者介面增強功能之類的呈現方式細節，通常不屬於 ADM 的範疇。如果爲應用程式建構了個別的使用者介面模型，則這是放置上述項目的適當場所。ADM 需要將焦點持續放在商業問題和解決方案空間的表達上。在網頁設計師的這個時代，網頁的外觀與操作方式最好是由專業人員（技術圖形設計師）而不是傳統開發人員來設計和實作。

Web 應用程式架構

Web 應用程式的基本架構包括瀏覽器、網路和 Web 伺服器。瀏覽器向伺服器要求「網頁」。每一張網頁是內容和格式化指示的混合物，以 HTML 表示。有些網頁包含由瀏覽器解譯的用戶端 Script。這些 Script 定義顯示網頁的其他動態行爲，它們通常與瀏覽器、網頁內容和網頁包含的其他控制項（Applet、ActiveX 控制項和外掛程式）互動。使用者檢視網頁內容並與之互動。有時候使用者在網頁的欄位元素中輸入資訊，並將它們送出至伺服器處理。使用者也可以透過超鏈結瀏覽到系統的不同頁面，與系統互動。不論是哪一種情況，使用者都會在系統上輸入資料，這可能會改變系統的「商業狀態」。

從用戶端的觀點來看，網頁固定爲 HTML 格式的文件。然而，在伺服器上，「網頁」可以用許多不同的方式來證明自己。在最早的 Web 應用程式中，動態網頁是以一般閘道介面（Common Gateway Interface, CGI）來建置的。CGI 爲要使用的 Script 和已編譯模組定義一個介面，來存取隨著網頁要求傳遞的資訊。在以 CGI 爲基礎的系統中，有一個特殊目錄會配置在 Web 伺服器上，該伺服器能夠執行 Script 來回應網頁要求。在要求 CGI Script 時，伺服器不只傳回檔案的內容（像對任何 HTML 格式檔案一樣），還會以適當的直譯器（通常是 PERL Shell）處理或執行檔案，並以串流方式將輸出傳回到發出要求的用戶端。此處理的最後結果是一個 HTML 格式串流，它傳回到發出要求的用戶端。在處理檔案時，商業邏輯會在系統中執行。在這段時間，它有可能與伺服器端資源互動，例如資料庫和中層元件。

現在的 Web 伺服器在此基本設計上已經有所改善。現在，它們更注重安全，包括了像在伺服器上管理用戶端狀態、交易處理整合、遠端管理和資源儲存區作業之類的特性。總而言之，最新一代的 Web 伺服器能夠處理那些對於重要、可調式、健全應用程式的架構師很重要的問題。

在檢視 CGI Script 的角色時，今天的 Web 伺服器可以分成三大類：Script 化網頁、已編譯的網頁、兩者的混合。第一類，用戶端瀏覽器可要求的每一張網頁在 Web 伺服器的檔案系統上是以 Script 化檔案來表示。此檔案通常是 HTML 與其他 Scripting 語言的混合。要求此網頁時，Web 伺服器會將此網頁的處理委派給能夠辨識它的引擎，最後結果是 HTML 格式串流傳回到發出要求的用戶端。Microsoft 的 Active Server Pages、JavaServer Pages 和 Cold Fusion 就是其中的一些範例。

第二類是已編譯的網頁，Web 伺服器將載入和執行二進位元件。就像 Script 化網頁一樣，此元件也可以存取隨著網頁要求而來的所有資訊（格式欄位和參數的值）。已編譯的程式碼使用要求詳細資料，通常存取伺服器端資源來產生要傳回至用戶端的 HTML 串流。雖然沒有規定，但已編譯的網頁傾向於封裝比 Script 化網頁更大的功能。藉由將參數傳遞到已編譯的網頁要求，即可獲得不同的功能。任何一個已編譯的元件實際上可包含整個目錄的 Script 化網頁的所有功能。可代表此架構類型的技術爲 Microsoft 的 ISAPI 和 Netscape 的 NSAPI。

第三類代表 Script 化網頁，一旦有此要求，就會編譯 Script 化網頁，然後所有後續要求就會使用這個已編譯的版本。只有在原始網頁的內容變更時，此網頁才會歷經另一次編譯。這個種類是介於 Script 化網頁的彈性和已編譯網頁的有效性之間的折衷方案。

網頁建模

不論是 Script 化或已編譯的網頁，都會以一對一的方式對映到 UML 中的元件。元件是系統的「實體」及可更換組件。模型的實作視圖（元件視圖）說明系統的元件及其關係。在 Web 應用程式中，此視圖說明系統的所有網頁，及彼此之間的關係（亦即，超鏈結）。在某個層次上，Web 系統的元件圖類似網站對照表。

由於元件只代表介面的實體套裝，因此，它們並不適合網頁內協同作業的建模。此摘要層次雖對於設計師和執行者非常重要，但仍然必須是模型的一部分。一開始，我們認爲每一張網頁是模型的設計視圖（邏輯視圖）中的一個 UML 類別，它與其他網頁的關係（關聯）代表超鏈結。然而，想想看，若一個給定的網頁有可能代表一組只存在於伺服器上的功能和協同作業，而在用戶端上是完全不同的另外一組，那麼這種摘要就失敗。運用動態 HTML（用戶端 Scripting）作爲其輸出之一部分的任何伺服器 Script 化網頁，就是這類網頁的一個範例。此問題的直覺反應可能是爲類別中的每一個屬性或作業建立造型，來指出它在伺服器或用戶端上是否有效。這時候，原本打算幫助簡化內容的模型就變得很複雜。

解決此問題的較好方法是考量「重點分隔」的主體。在邏輯上，伺服器上的網頁行爲與用戶端完全不同。在伺服器執行時，它可以存取（亦即，產生關係）伺服器端資源（中層元件、資料庫、檔案系統等等）。在用戶端上，同一頁或該頁的串流式 HTML 輸出，卻有完全不同的行爲和關係設定。在用戶端上，Script 化網頁是透過文件物件模型

（Document Object Model，DOM）而與瀏覽器本身、以及該網頁指定的任何 Java Applet、ActiveX 控制項或外掛程式產生關係。對於認真的設計師而言，可能與用戶端的其他「作用中」網頁還有其他關係，這些網頁出現在另一個 HTML 頁框或瀏覽器實例中。

透過重點分隔，我們可以用一個類別爲網頁的伺服器端層面建模，用另一個類別爲用戶端層面建模。我們區分兩者，作法是使用 UML 的延伸機制，爲每一個伺服器頁面和用戶端頁面定義造型和圖示。 UML 的造型可讓我們定義建模元素的新語意。造型類別可在 UML 圖表中以自訂圖示呈現，或直接以<< >>括住造型名稱來裝飾。圖示對總覽圖很有幫助，而在顯示類別屬性和作業時，最好使用簡單標示。

對於網頁而言，造型可指出類別是用戶端或伺服器上的網頁邏輯行爲的摘要。兩個摘要彼此相關，兩者之間存在著有方向性的關係。此關聯的造型爲 <<build>>，因爲可以說伺服器網頁建置用戶端網頁（圖 1）。每一個動態網頁（亦即，其內容是在執行時期決定）是以伺服器網頁建構。每一個用戶端網頁至多是由單一伺服器網頁建置；然而，伺服器網頁有可能建置多個用戶端網頁。

網頁之間的一般關係是超鏈結。 Web 應用程式中的超鏈結代表系統的導覽路徑。此關係在模型中是以鏈結造型關聯來表示。此關聯一律源自用戶端網頁，並指向用戶端或伺服器網頁。

超鏈結在系統中是實作爲對網頁的要求，而網頁在實作視圖中是以元件建模。大部分用戶端網頁的鏈結關聯相等於建置用戶端網頁的伺服器網頁的鏈結關聯。這是因爲鏈結實際上是對網頁的要求，而不是任何類別摘要。由於網頁元件實現這兩個網頁摘要，因此網頁元件所實現的任何類別的鏈結是相等的。

標示值是用來定義隨著鏈結要求而傳遞的參數。連結關聯標示值 arameters 是參數名稱（及選用值）的清單，處理該要求的伺服器網頁將預期及使用它們。在「圖 2」，SearchResults 網頁包含對 GetProduct 伺服器網頁的一些不定數量的超鏈結 (0..*)，其中每一個鏈結對 productId 參數有不同的值。GetProduct 網頁建置 productId 參數所指定之產品的 ProductDetail 網頁。

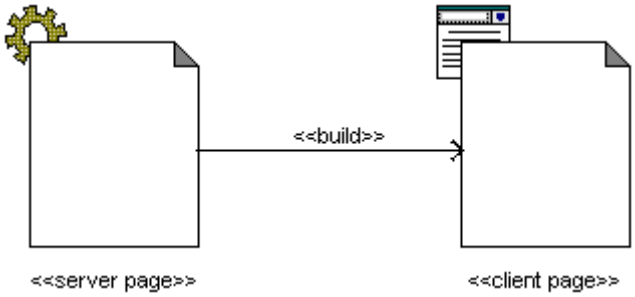

圖 1：伺服器網頁建置用戶端網頁

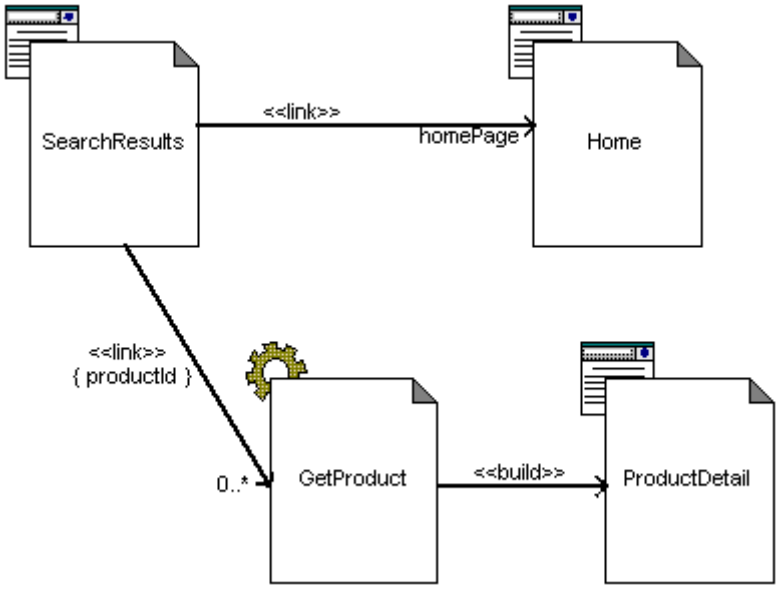

圖 2：使用超鏈結參數

使用這些造型，可以更容易地爲網頁 Script 和關係建模。 <<server page>> 類別的作業變成頁面伺服器端 Script 中的函數，其屬性則變成網頁範圍變數（可由網頁函數全域存取）。 <<client page>> 類別的作業和屬性同樣變成用戶端上可見的函數和變數。將網頁的伺服器和用戶端層面分成不同類別，其主要優點是在於網頁與系統其他類別之間的關係。用戶端網頁是以與用戶端資源之間的關係建模：DOM、Java Applet、ActiveX 控制項和外掛程式（圖 3）。伺服器網頁是以與伺服器端資源之間的關係建模：—中層元件、資料庫存取元件、伺服器作業系統等等，請參閱「圖 4」。

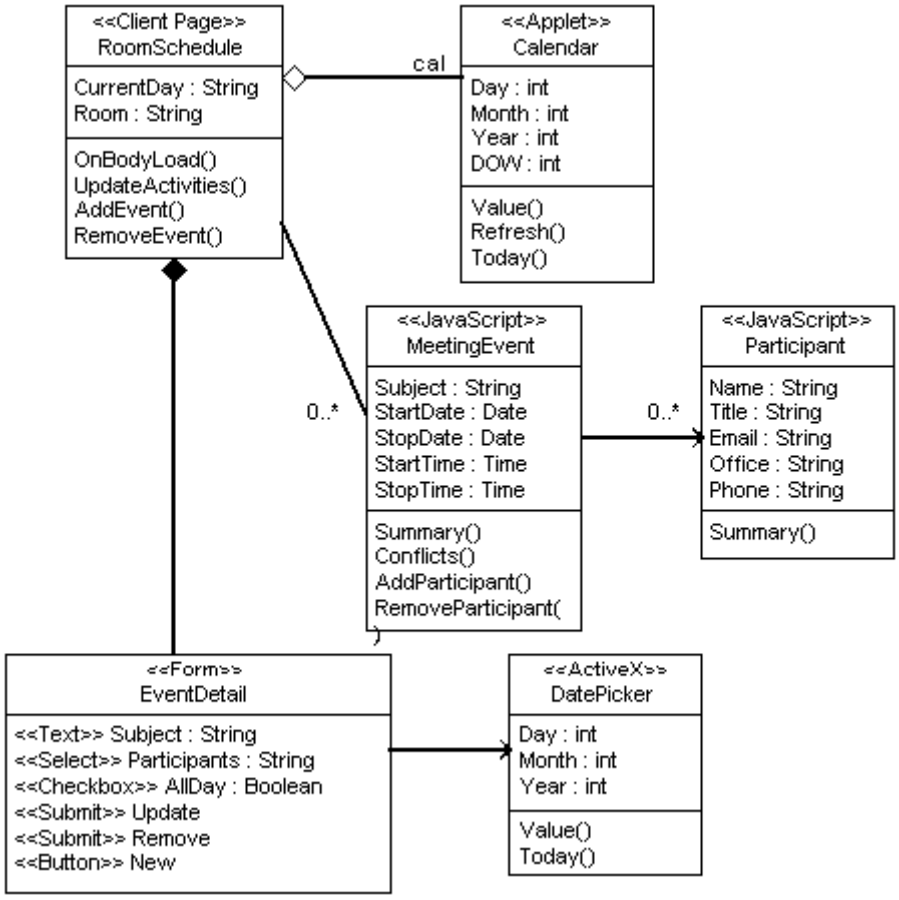
<<Client Page>>
RoomSchedule
CurrentDay : String
Room : String
OnBodyLoad()
UpdateActivities()
AddEvent()
RemoveEvent()
cal
<<Applet>>
Calendar
Day : int
Month : int
Year : int
DOW : int
Value()
Refresh()
Today()
0..*
<<JavaScript>>
MeetingEvent
Subject : String
StartDate : Date
StopDate : Date
StartTime : Time
StopTime : Time
Summary()
Conflicts()
AddParticipant()
RemoveParticipant(
)
0..*
<<JavaScript>>
Participant
Name : String
Title : String
Email : String
Office : String
Phone : String
Summary()
<<Form>>
EventDetail
<<Text>> Subject : String
<<Select>> Participants : String
<<Checkbox>> AllDay : Boolean
<<Submit>> Update
<<Submit>> Remove
<<Button>> New
<<ActiveX>>
DatePicker
Day : int
Month : int
Year : int
Value()
Today()

圖 3：用戶端合作

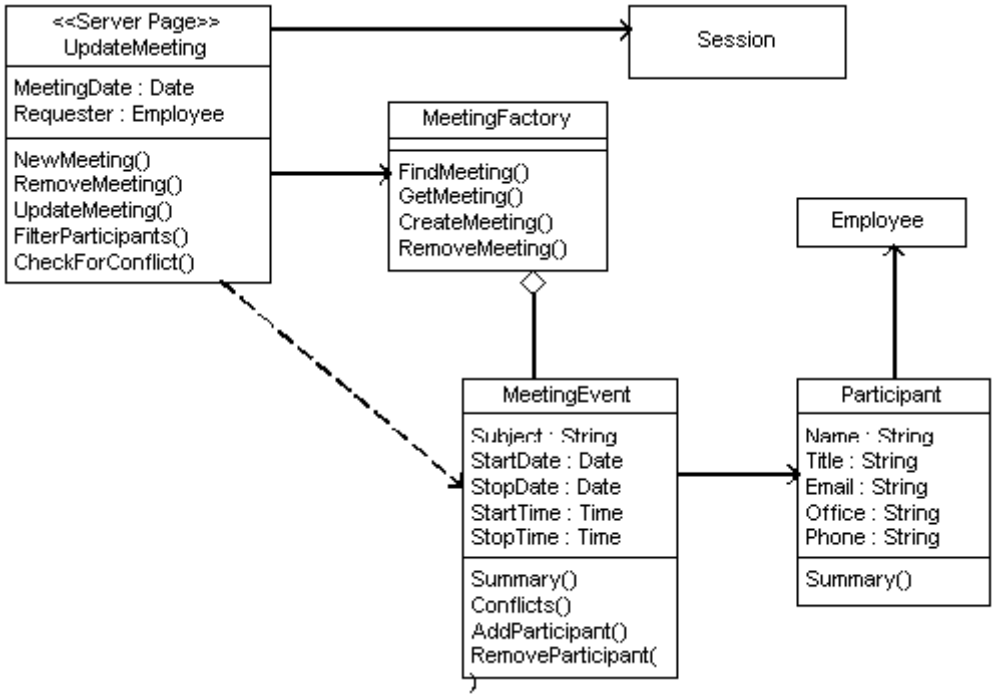

圖 4：伺服器合作

使用類別造型爲網頁的邏輯行爲建模的其中一個最大優點，就是它們與伺服器端元件的協同作業可以用和其他伺服器端協同作業幾乎相同的方式表示。<<server page>> 是參與系統商業邏輯的另一個類別。在比較概念性的層級上，伺服器網頁通常承擔控制器的角色，協調必要的商業物件活動來完成瀏覽器的網頁要求所起始的商業目標。

在用戶端，協同作業可能有點複雜。這有一部分是因爲可運用的技術的變化。最單純的用戶端網頁是同時包含內容和呈現資訊的 HTML 文件。瀏覽器使用網頁中的格式化指示來呈現 HTML 網頁，有時候會使用分開的樣式表。在邏輯模型中，此關係可以從用戶端網頁到 <<Style Sheet>> 造型類別的相依關係來表示。然而，樣式表主要是呈現問題，通常在 ADM 中會省略之

表單

網頁的主要資料輸入機制是表單。表單是以 <form> 標示定義在 HTML 文件中。每一個表單指定它要送出本身的那個網頁。表單包含一些輸入元素，全部以 HTML 標示表示。最常見的標示爲 <input>、<select> 和 <textarea> 標示。輸入標示有點超載，因爲它可以是文字欄位、勾選框、圓鈕、按鈕、影像、隱藏欄位以及一些其他較不常見的類型。建模表單代表另一個類別造型：<<Form>>。<<Form>> 沒有作業，因爲可以用 <form> 標示定義的作業實際上爲用戶端網頁所擁有。表單的輸入元素全部都是 <<Form>> 類別的造型屬性。<<Form>> 與扮演輸入控制項的 Applet 或 ActiveX 控制項之間可以有某種關係。每一個表單與伺服器網頁之間也有關係—伺服器網頁即處理表單送出的網頁。此關係的造型爲 <<submit>>。由於表單完全包含在 HTML 文件中，所以它們在 UML 圖表中是以固定聚集格式表示。「圖 5」顯示一個定義表單的簡單購物車網頁，並顯示與處理的伺服器網頁之間的送出關係。

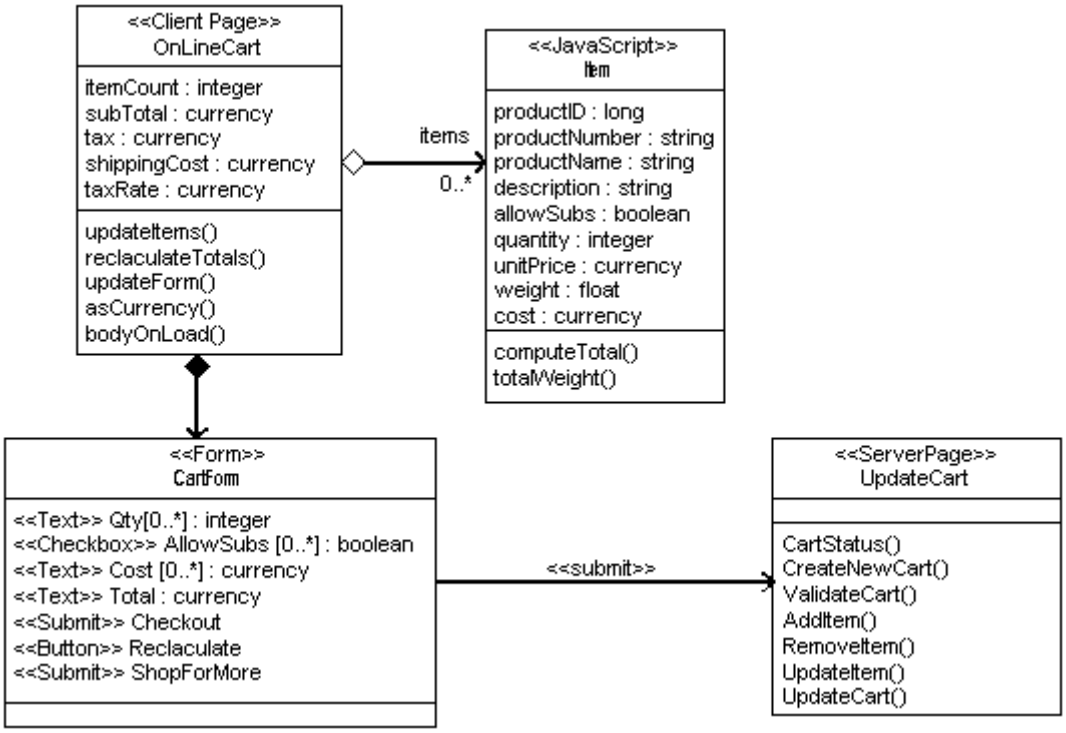

圖 5：表單送出至伺服器網頁

在「圖 5」，<<JavaScript>> 造型類別是代表購物車項目的一個物件。對於含有一些不定數量的實例的欄位，陣列語法將使用於表單內容的說明中。在此購物車的案例中，它表示購物車可以有零個或多個項目，每一個項目有 Qty、AllowSubs、Cost 和 Total <input> 元素。

由於用戶端網頁中的所有活動是以 JavaScript 執行，且 JavaScript 是無類型語言，因此，對任何這些屬性指定的資料類型只是爲了方便執行者澄清而使用。以 JavaScript 或 HTML 輸入標示實作時，類型會被忽略。這也適用於函數參數，雖然本圖未完全顯示出來，但函數參數也是模型的一部分。

頁框

自從引進之後，在網站或應用程式中使用 HTML 頁框一直是分化爭辯的主題。頁框容許有多頁在作用中並可讓使用者隨時看見。目前最普遍的瀏覽器的最新特性集也容許有多個瀏覽器實例作用在使用者機器上。利用動態 HTML Script 和元件，這些網頁可以彼此互動。用戶端上很可能有複雜的互動，因此建模的需求更大。

應用程式中是否有運用頁框或多個瀏覽器實例，由軟體架構師決定。如果有，此用戶端行爲的模型需要以 ADM 表示，其理由同上。爲了將頁框用法建模，我們多定義兩個類別造型—<<frameset>> 和 <<target>>—和一個關聯造型 <<targeted link>>。頁框組類別代表儲存區物件，並直接對映到 HTML <frameset> 標示。它包含用戶端網頁和目標。目標類別是一個具名頁框或瀏覽器實例，由其他用戶端網頁參照。已設定目標的鏈結關聯是另一個網頁的超鏈結，也是在特定目標呈現的鏈結。在「圖 6」所顯示的範例中，一個常見的「概要」視圖使用兩個頁框呈現在瀏覽器中。一個頁框以目標命名 (Content)，另一個頁框只包含用戶端網頁。此用戶端網頁頁框是書籍的目錄 (TOC)。此網頁中的超鏈結已設定目標，因此它們會呈現在 Content 頁框中。結果左邊一個靜態目錄，右邊是逐章列出的書籍內容。

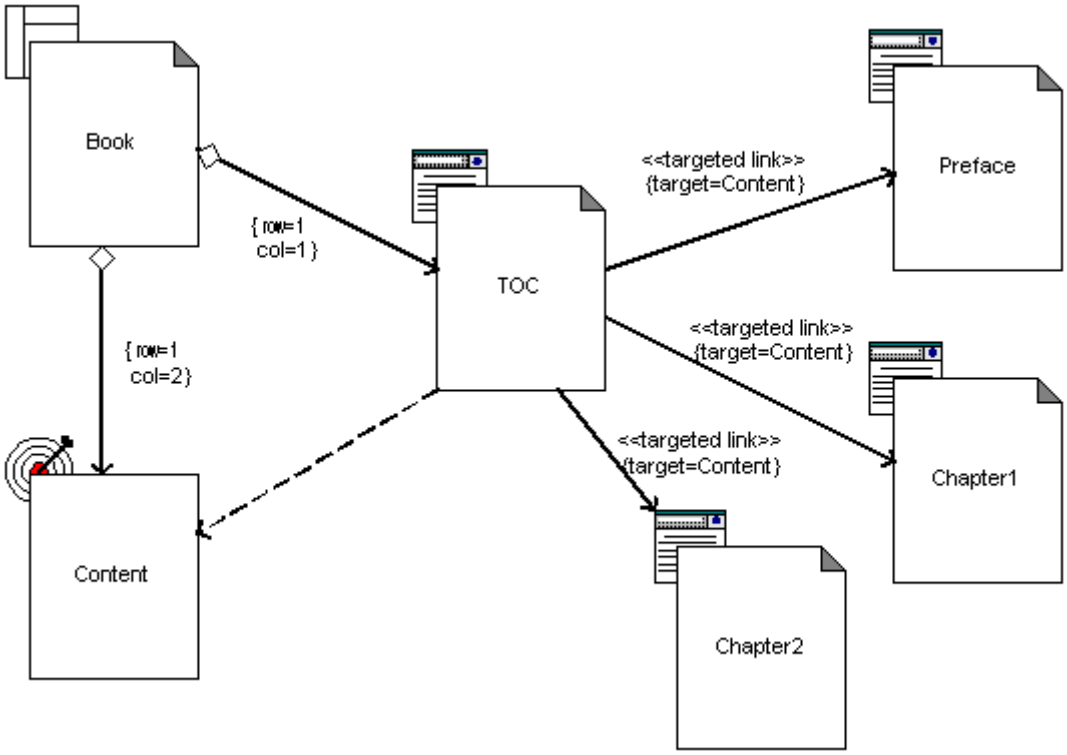

圖 6：頁框範例

很多實際呈現細節由頁框組及其關聯的標示值所擷取。頁框組與目標或用戶端網頁之間的聚集關係上的兩個標示值，指定目標或網頁所屬的頁框組列和欄。已設定目標的鏈結關聯上的標示值 Target 識別應呈現網頁的 <<target>>。

當目標不是以頁框組聚集時，這表示是使用個別瀏覽器實例來呈現網頁。請牢記，此表示法是表達用戶端機器的單一實例。多重獨立的目標全部假設爲在相同機器上執行，且該圖表表達一個用戶端實例的用戶端行爲。任何其他部署配置需要密集記載於模型中，以獲得更充分的瞭解。

結論

本白皮書所討論的一些想法和概念，是以 UML 爲 Web 應用程式特定元素建模時所遭遇的問題和解決方案的介紹。此工作的目標是要呈現一致而完整的方式，來整合建模 Web 特定元素與應用程式的其餘部分，使詳細和摘要層次適合 Web 應用程式的設計師、執行者和架構師。Web 應用程式的 UML 的正式延伸規格的第一個版本幾近完成。此延伸規格將提供一般方式，讓架構師和設計師以 UML 表達其 Web 應用程式設計的全部。

您可以在網際網路的 Rational Software 網站的 Rational Rose 和 UML 區段中找到此延伸規格的最新資訊。

Rational
the software development company

兩個總公司：

Rational Software
18880 Homestead Road
Cupertino, CA 95014
電話：(408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
電話：(781) 676-2400

免付費專線：(800) 728-1212
電子郵件：info@rational.com
網址：www.rational.com
國際辦事處：www.rational.com/worldwide