分層策略

Peter Eeles

Rational Software 白皮書

TP 199, 08/01

Rational®
the software development company

目錄

摘要

有一些分解軟體系統的技術。分層是範例之一，本白皮書會加以說明。這種技術處理兩項主要考量：大部分系統太過複雜而無法完全理解，不同讀者需要系統的不同層面。

許多軟體系統都採用過「分層」，許多文字和 Rational Unified Process (RUP) 也都支持分層。然而，分層常常被誤解及套用錯誤。本白皮書澄清分層的意義，並討論套用不同分層策略的影響。

什麼是「分層」?

首先，我們先定義什麼是「分層」。*分層*這個詞彙是指架構型樣的應用，一般稱爲*「分層」型樣*，它在許多文字中 ([Buschmann]、[Herzum]、[PloP2]) 以及在 RUP 中都有描述。*型樣*代表存在於特定環境定義中的一般問題的解決方案。「表 1」有提供分層型樣的概觀。

表 1：分層型樣的概觀

	分層型樣
環境定義	需要分解的系統
問題	太過複雜而無法全面瞭解的系統 難以維護的系統 未隔離其最不穩定元素的系統 難以識別其大部分可重複使用的元素的系統 要由不同團隊（可能以不同技巧）建置的系統
解決方案	建立系統分層結構

其中一個最熟悉的分層範例是 OSI 7 層模型，它是由國際標準組織 (ISO) 所定義。「圖 1」所顯示的這個模型定義一組網路通訊協定— 每一層聚焦在通訊的特定層面，並建立在其下一層的機能上。OSI 7 層模型使用責任式分層策略：每一層有特定的責任。本白皮書稍後會詳細描述此策略。

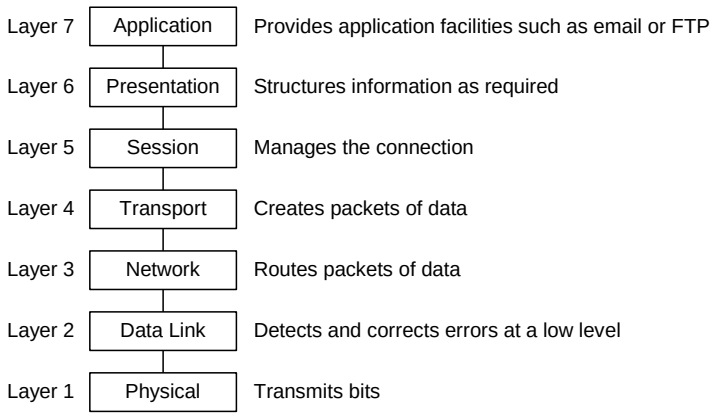

圖 1：OSI 7 層模型（責任式分層）

「圖 2」顯示另一個責任式分層範例。

- *呈現邏輯層*包含負責提供某種衍生形式給使用者的元素，例如使用者介面中的元素。
- *商業邏輯層*包含負責執行某種商業流程及套用商業規則的元素。
- *資料存取邏輯層*包含負責提供資訊來源存取權的元素，例如關聯式資料庫。

請注意，分層有多種建模方式，本白皮書稍後會說明。目前，我們使用造型爲 <layer> 的 UML 套件來明確代表層。

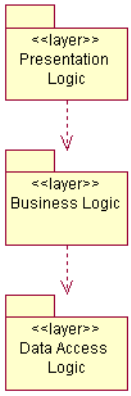

圖 2：責任式分層

此責任式分層的特定範例所顯示的層通常稱爲 Tier，在分散式系統開發中，這是一個大家都熟悉的概念，可以在裡面找到 *2-層*、*3-層*和 *n-層*系統。

「圖 2」的一個重要層面是所顯示的 *相依關係方向* ，因爲它暗示一特定規則，是分層系統的一個性質 — 特定層的元素只能存取同一層的元素，或其下層的元素[1]。在這裡所提供的範例中，*商業邏輯層* 不能存取*呈現邏輯層*的元素。而且，*資料存取邏輯層*的元素也不能存取*商業邏輯層*的元素。此結構通常稱爲指引式非循環圖 (directed acyclic graph，DAG)。它*有指引方向*是因爲相依關係是單向的，*非循環*是因爲相依關係的路徑從不循環。

更明確的說，在定義分層策略時，每一層的意義一定要準確，這樣元素才能正確放置在適當的層。若沒有正確指派元素至適當的層，將會貶低首先套用策略的價值。當每一層策略的討論越來越詳細時，對每一層的意義會提供一般指引。

建模層

當我們探索不同的分層策略時，顯然，使用特定*模型*（及特定 UML 元素）來溝通每一個策略是適當的作法。從特定觀點來看，*模型*代表系統的完整說明。「圖 3」顯示四個模型的範例，代表所考慮的系統的不同視景：

- 使用案例模型：擷取系統需求
- 分析模型：擷取系統需求分析
- 設計模型：擷取系統設計
- 實作模型：擷取系統實作

[1] 雖然事件通知可能導致訊息從某一層的元素傳送到上一層的元素，但並無此方向的明確相依關係存在。

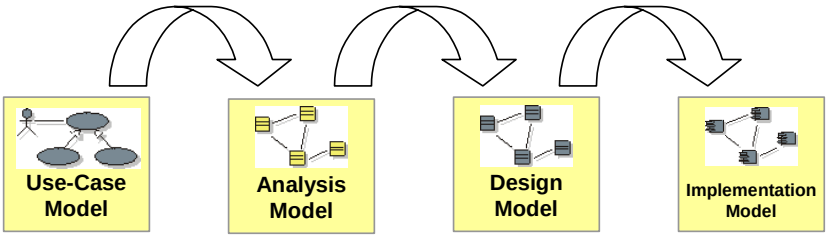

圖 3：代表逐漸修正的四個模型

其他的模型包括：

- *部署模型：*擷取系統的分送層面
- *資料模型：*擷取系統的持續性層面

分層策略

分層可以一些特性爲基礎。本節基於下列特性來討論分層：

- 責任
- 重複使用

詳細討論每一個策略時，會考慮每一個策略的表示法。

責任式分層

或許最常用的分層策略就是責任式分層。此特定策略可改進系統的開發和維護，因爲不同的系統責任彼此之間是隔離的。例如（請參閱「圖 2」），系統可以基於下列責任來分層：

- 呈現邏輯
- 商業邏輯
- 資料存取邏輯

這些責任每一個都可以用一層來代表，如「圖 4」所示，它顯示每一層的一些範例內容。這裡我們考量訂單處理系統中的三個概念 — Customer、Order 和 Product。例如，*Customer* 概念包含下列各項：

- *CustomerView 類別：*負責與客戶相關聯的*呈現邏輯*，例如客戶在使用者介面的衍生
- *Customer 類別*：負責與客戶相關聯的*商業邏輯*，例如客戶詳細資料的驗證
- *CustomerData 類別*：負責與客戶相關聯的*資料存取邏輯*，例如建立客戶持續性狀態

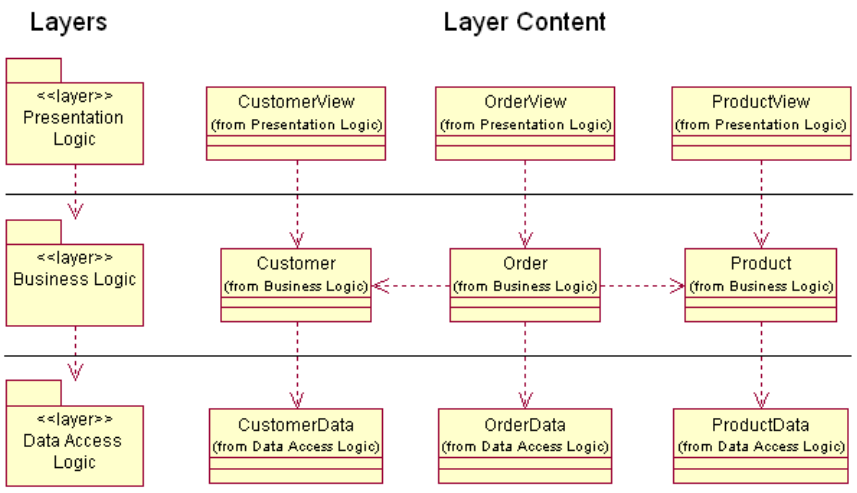

圖 4：責任式分層的層和內容

現在我們考量有關此特定分層策略的一些「迷思」。

迷思 1：Layer 和 Tier 不同

此特定迷思是常見的困惑來源。事實上，tier *就是* layer，只不過 layer 是以特定策略爲基礎—責任之一。如「表 2」所示，Tier 的概念有許多套用方式，這樣的事實令人更加困惑。

表 2：Tier 定義

應用程式	Layer (Tier)
2 層	結合呈現邏輯與商業邏輯 資料存取邏輯
3-層	呈現邏輯 商業邏輯 資料存取邏輯
N-層	呈現邏輯 商業邏輯（*分散式*） 資料存取邏輯

迷思 2：Layer (tier) 默示實體分送

另一個常見的誤解是邏輯層默示實體分送。想想一個 3-層的分層。即使不同元素會位於其中一層，每一層本身還是可以使用像「表 3」所顯示的一些方式來套用，「表 3」使用常用來描繪特定實體分送的名稱（例如「小型用戶端」）。

表 3：3-層分層的應用程式

應用程式	層	
	用戶端	伺服器端
單一系統	呈現邏輯 商業邏輯 資料存取邏輯	
小型用戶端	呈現邏輯	商業邏輯 資料存取邏輯
大型用戶端	呈現邏輯 商業邏輯	資料存取邏輯

單一系統可運用多個實體分送策略，這種說法也對，其中特定元素可分類爲象徵「小型用戶端」分送及「大型用戶端」分送。通常，是依據非功能需求來選擇，例如效能。

責任式分層建模

接下來我們會看到，此策略的應用程式可能影響*設計模型*、*實作模型和部署模型*。*設計模型*通常是使用兩個方法的其中之一建構。

第一個方法顯示元素「內含」於該層內。結果隱含於「圖 5」中，這是一個 Rational Rose 瀏覽器擷取畫面，它顯示：

- *呈現類別*（CustomerView、OrderView 和 ProductView），位於呈現邏輯套件內
- *商業邏輯類別*（Customer、Order 和 Product），位於商業邏輯套件內
- *資料存取邏輯類別*（CustomerData、OrderData 和 ProductData），位於資料存取邏輯套件內

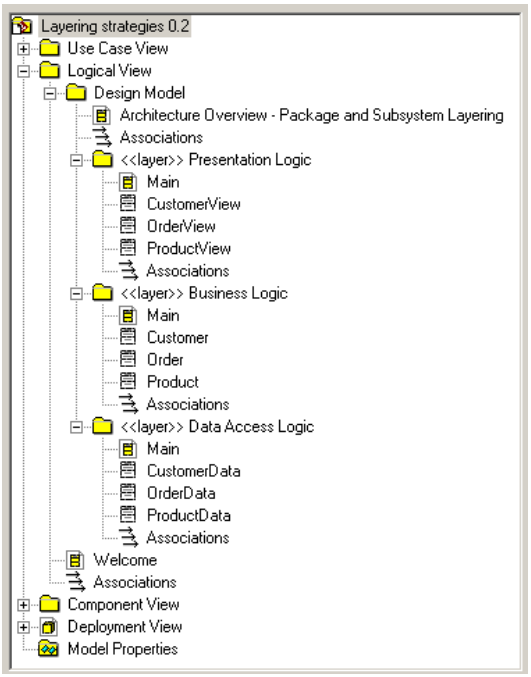

圖 5：層內包含的元素

第二個方法結合*商業元件*一等公民的概念（在此案例中，是指 Customer、Order 和 Product），重要的主要元素藉此成爲系統支援的網域相關概念。例如，*Customer* 概念可能有呈現邏輯、商業邏輯和資料存取邏輯的相關聯元素。此商業元件概念在 [Eeles] 和 [Herzum] 有進一步的討論。此思考方式產生「圖 6」所顯示的模型結構。在此範例中，分層是以元素名稱來*默示*。例如，所有 *View* 類別（例如 *CustomerView*）默示呈現邏輯層，所有 *Data* 類別（例如 *CustomerData*）默示資料存取邏輯層。未限定的類別名稱（例如 *Customer*）默示商業邏輯層。

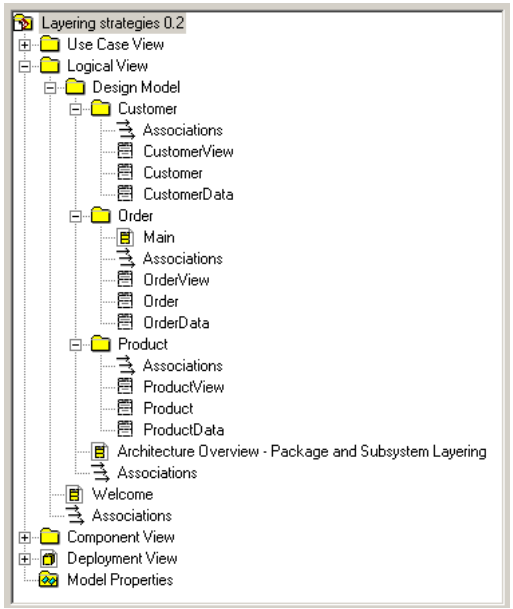

圖 6：在每一個商業元件套件內的*層*

分層也可以在代表商業元件的每一個套件內明確地表示（如「圖 7」所顯示）。*當給定的商業元件的每一層包含許多元素時，此結構更加合用*。雖然這個範例只展開 Customer 商業元件套件，Order 和 Product 套件也會有類似的結構。

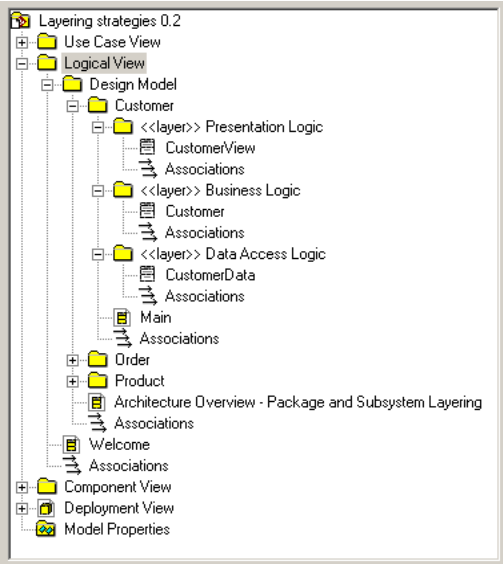

圖 7：商業元件套件內的*明確*分層

除了設計模型之外，如果有需要實體分割實作每一個責任的元素，則責任式分層策略通常會影響*實作模型*。例如，有一個系統展示「小型用戶端」實體分送：識別支援用戶端執行所需的實作單元以及支援伺服器執行所需的那些單元，會很有幫助。在此範例中，*呈現邏輯層*的元素位於已部署在用戶端的應用程式，而*商業邏輯層*和*資料邏輯層*的所有元素是位於已部署在伺服器上的另一個應用程式中。

此種情況暗示如「圖 8」所顯示的*實作模型*，它呈現一個 Rational Rose 瀏覽器影像和一個元件圖，該圖顯示已部署在用戶端的應用程式的元素。在此範例中，設計模型的一個類別與實作模型的一個 UML 元件之間剛好是一對一對映。然而，請注意，此對映通常視使用的實作技術而定。

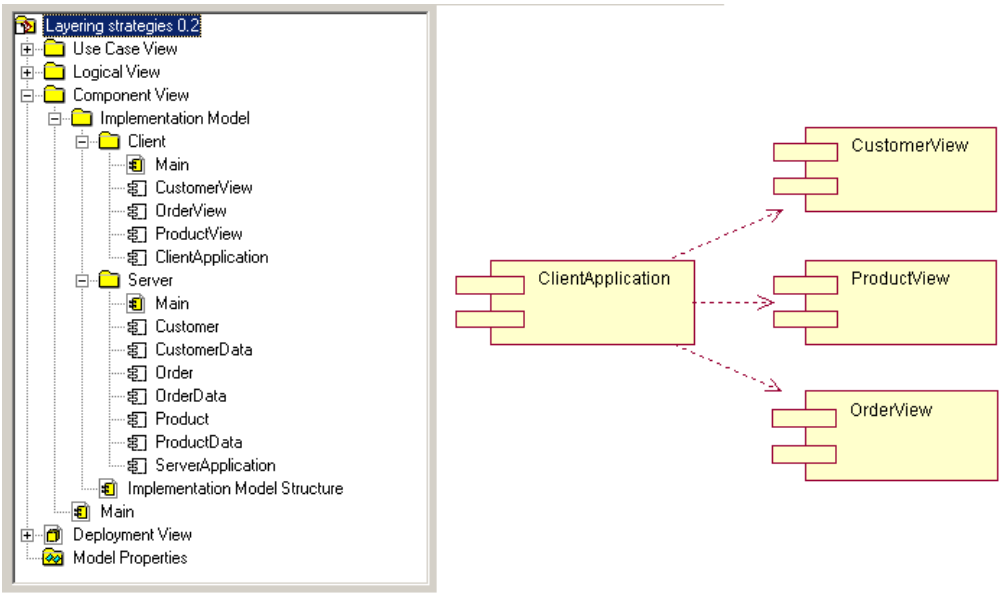

圖 8：實作模型內*隱含的*分層

同樣地，如果有需要說明責任的實體分送，則責任式分層策略也會影響*部署模型*。在「圖 9」中，及使用上述範例時，我們看到已定義六個節點。三個*用戶端節點*的每一個都存放了一個 ClientApplication 流程。FrontEndServer 節點存放 LoadBalancer 流程，它負責將用戶端要求分送到兩個伺服器節點的其中之一。每一個*伺服器節點*各存放一個 ServerApplication 流程。

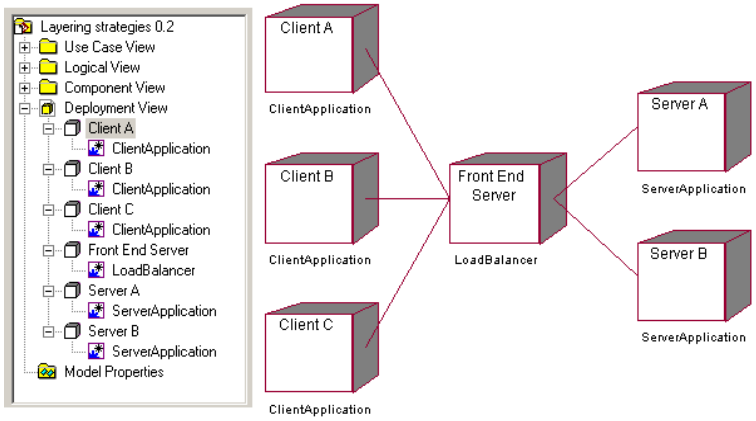

圖 9：說明責任實體分送的部署模型

重複使用式建模

另一個常用的分層是以重複使用爲基礎。此策略與有明確目標的組織特別有關，此明確目標就是在整個組織內重複使用元件。使用此分層策略的影響，就是元件的重複使用性非常明顯，因爲元件已根據其重複使用層次而明確分組。「圖 10」顯示從 [Jacobson] 所描述的策略中衍生的一個範例分層。這裡我們會看到三層：基本、特定商業專用和特定應用程式專用。

- *基本層*包含可套用到整個組織的元素（例如 Math）。這類元素將廣泛地重複使用。
- *特定商業專用層*包含套用至特定組織的那些元素，但它們與應用程式無關（例如 Address Book）。這類元素將在相同組織的應用程式內重複使用。
- *特定應用程式專用層*包含套用至特定應用程式或專案（例如 Personal Organizer）的元素。這些元素是最少重複使用的。

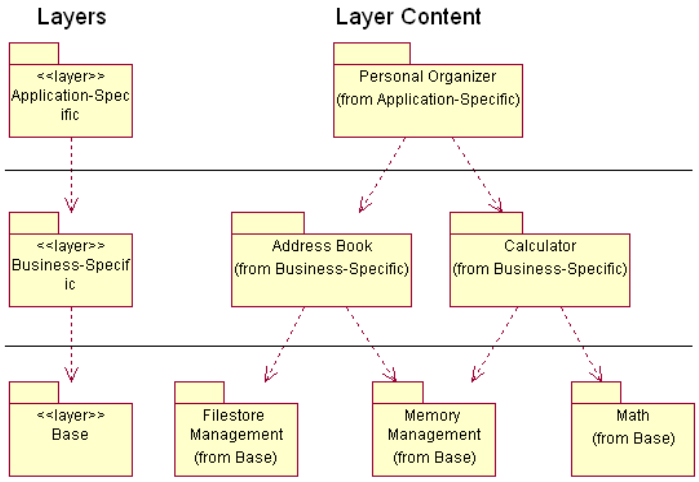

圖 10：重複使用式分層範例

由此我們可以得知，基本層的元素是最多重複使用的，而特定應用程式專用層的那些元素是特定專案專用的，因此較少重複使用。

重複使用式分層建模

重複使用策略的應用程式主要影響*設計模型*。有包含重複使用式分層的設計模型之結構是可以直接面對的，它顯示在「圖 11」，反映出「圖 10」的範例。

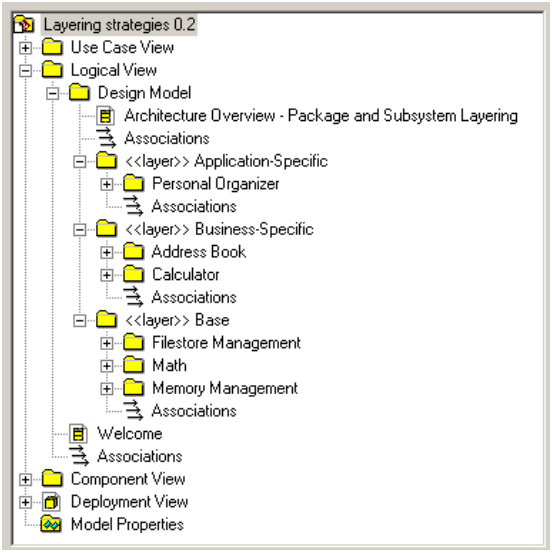

圖 11：包含重複使用式分層的設計模型

其他分層策略

本白皮書使用兩個最廣泛使用的策略作爲範例，爲已存在的不同分層策略增加一點「內容」。然而，對於確認安全、擁有權和技能集等特性的策略，也可以採用類似的方法。

多維分層

前述的策略也可以結合起來建立新的分層策略。「圖 12」的範例顯示：

- 前一個範例中的兩個重複使用式分層
  - 特定應用程式專用
  - 特定商業專用
- 三個責任式分層 (tier)
  - 呈現邏輯
  - 商業邏輯
  - 資料存取邏輯

重複使用式分層策略所呈現的相依關係通常是起因於商業邏輯層的元素之間的相依關係，如「圖 12」所暗示，我們可以在那裡看到 PersonalOrganizer 和 AddressBook 之間的相依關係。

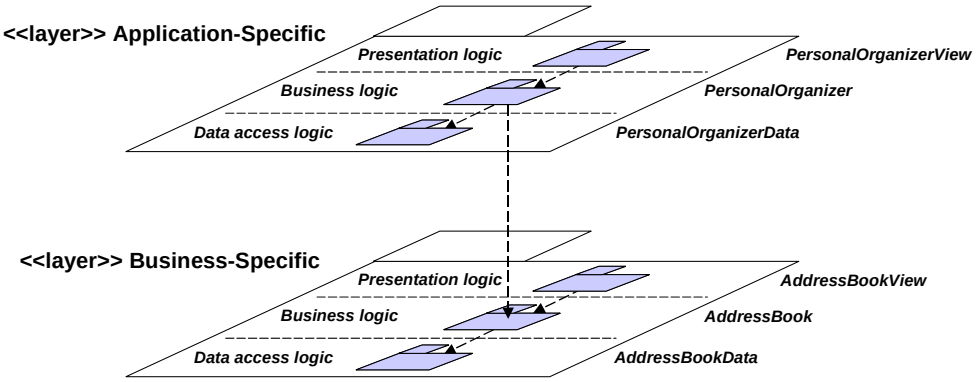

圖 12：多維分層

多維分層建模

這裡我們考慮二維*設計模型*內的分層的多維層面的表示法。我們也考慮藉此包含*商業元件*概念的一種結構。

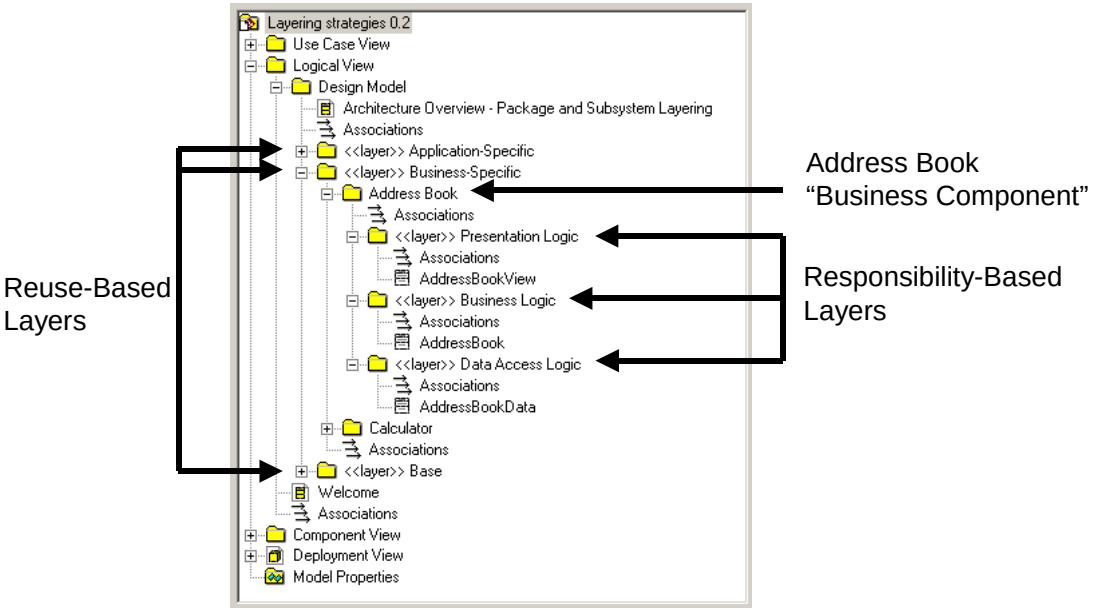

圖 13：包含多維分層的設計模型

採用多維分層策略需要先識別主要策略。在我們的範例中，主要分層策略是基於重複使用。設計模型先依據此策略組織，提供*特定應用程式專用層*、*特定商業專用層*和*基本層*。這每一層再由其中的元素進一步組織；例如，「圖 13」顯示特定商業專用層，它包含*通訊錄*和*計算機*。這每一個元素再依據第二策略進一步組織：責任式分層。例如，*通訊錄*套件包含三層：*呈現邏輯*、*商業邏輯*和*資料存取邏輯*。

每一層再包含此層裡的元素：

- *呈現邏輯層*包含 AddressBookView 類別
- *商業邏輯層*包含 AddressBook 類別
- *資料存取邏輯層*包含 AddressBookData 類別

結論

架構師必須做的最重要決定之一是選擇適當的分層策略，因爲它對所產生的模型之結構有重大影響。然而，更重要的是這個事實：商業利益（例如可維護性和重複使用）可以直接由分層策略支援。例如，透過採用責任式分層策略，如果系統的不同責任與另一個系統是隔離的，則可開發更多可維護的系統。同時，可重複使用的系統元素可使用重複使用式分層策略來明確識別。

感謝辭

作者想感謝 Kelli Houston、Wojtek Kozaczynski、Philippe Kruchten、Bran Selic 和 Catherine Southwood 等人（全部都與 Rational Software 有關）對本白皮書初稿提出的寶貴意見。

參考書目

[Buschmann] Buschmann, Frank, et al. *A System of Patterns*. 1996. New York: John Wiley & Sons. ISBN 0-471-95869-7.

[Edwards] Edwards, Jeri. *3-Tier Client/Server at Work*. 1999. New York: John Wiley & Sons. ISBN 0-471-31502-8.

[Eeles] Eeles, Peter, and Oliver Sims. *Building Business Objects*. 1998. New York: John Wiley & Sons. ISBN 0-471-19176-0.

[Herzum] Herzum, Peter, and Oliver Sims. *The Business Component Factory*. 2000. New York: John Wiley & Sons.

[Jacobson] Jacobson, Ivar, et al. *Software Reuse*. 1997. Reading, Massachusetts: Addison-Wesley. ISBN 0-201-92476-5.

[PLoP2] Vlissides, John, James Coplien, and Norman Kerth. *Pattern Languages of Program Design 2*. 1996. Reading, Massachusetts: Addison-Wesley. ISBN 0-201-89527-7.

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