模型結構準則
適用於 Rational Software Modeler、Rational Systems Developer 及 Rational Software Architect
（「傳統 RUP」導向）

白皮書

Bill Smith，模型導向開發，IBM Rational Software

7.0 版
2006/3/24

目錄

1. 簡介

適用讀者群

本文適用於 Rational Software Architect (RSA)、Rational Systems Developer (RSD) 及 Rational Software Modeler (RSM) 產品（統稱爲 RSx）的使用者，尤其，對於想要運用 Rational Unified Process (RUP) 的建模原則來使用 RSx 的使用者，更是實用。如果您是 Rational Software Modeler (RSM) 的使用者，您會覺得本文很實用，但請注意，有些章節描述的功能只有 RSA 和 RSD 才提供。

本文的重點是在 RSx 中建立一組新的模型。如果您是第一次使用 RSx，但一直都是 Rational Rose 或 Rational XDE 的使用者，而現在打算從這些產品匯入模型，本文也可以做爲將匯入的模型重新建構的指導來源。

本文假設您有 RUP 和 UML 的知識背景。也假設您熟悉 RSx 的基本概念和作業原理，請參閱 "**The New IBM Rational Design and Construction Products for Rose and XDE Users**"

目的

RUP 描述一組模型，例如使用案例模型、分析模型及設計模型，其代表在系統的問題和解決方案領域上有明確定義觀點的模型。這組模型的效用在許多實際的專案中已獲得證實。即使不採用 RUP，這些模型結構仍然值得參考。本文說明如何利用 RSx 來實現這些模型。

其他的建模方法也可列入考量，但支援的模型建構指導原則已超出本文的範圍。例如，「商業導向開發」方法可用來塑造「服務導向架構」的模型。一開始可能以商業程序模型爲起點 – 可能以 UML2 活動圖來表示 – 接著開始指定在商業程序中將作業自動化的一組服務的契約。這種方法所建議的模型結構與本文說明的結構迥然不同。

切記，本文說明的專案和模型結構只是準則，不是鐵律。是否以 RSx 來塑造特定 RUP 構件的模型，取決於您自己的開發流程。通常視每個專案而定。另外，RUP 也不是一套死板的流程規則，而是一套制訂程序定義的流程*架構*。這種程序定義可能很正式，也可能很輕率。

UML 建構的用法可能很正式，也可能很不正式。您可以將模型視爲建構時必須嚴格遵守的正式架構圖。或者，只將模型當做粗略設計的草稿，一旦專案進入實作階段之後，即可丟棄。RSx 在這些流程和建模領域上隨時支援您。本文的準則不是要束縛您的思考能力，而是幫助您了解如何利用 RSx 的特性來推動最適合您的流程。

請注意，RSx 不僅可將模型當做藍圖，也可當做規格，從規格中可自動產生重要的實作部分。這是以「模型至模型」和「模型至程式碼」轉換來達成。利用轉換來進行「模型導向開發 (MDD)」衍生出一些關於模型結構的特殊考量。您如果將以模型和轉換來進行「模型導向開發」，請至 developerWorks® 的 Rational 設計和建構區尋找 RSx MDD 的相關資源。

範圍

本文說明如何在 RSx 中表現 RUP 模型構件。同時也對這些構件的內部組織結構提出準則。本文並不

- 重申 RUP 模型構件的基本概念，也不擴大解釋
- 說明相關 RUP 構件之詳細語意或圖表內容的詳述流程或技術。

如須關於如何定義、開發及塑造 RUP 構件等非工具相關資訊的內容，請參閱 RUP。

如需 RSx 模型內容開發的特定工具技術的相關資訊，請參閱

- 產品文件（指導教學、範例、線上說明）
- RUP 配置中的工具輔助，此白皮書即爲其中一部分
- developerWorks 上的 RSx 相關資源

印刷慣例

從 IBM Rational Rose 或 XDE 移轉到 RSx 的使用者所感興趣的討論，將以資訊看板的型式呈現在淺灰色背景的滾邊文字框內：

> ***XDE/Rose***
> 已使用過 XDE 或 Rose 使用者所感興趣的討論。

內容編排

「基本概念及術語」章節提供一套實用的詞彙，也提供一些如何在 RSx 產品中實作模型的一般資訊。

接著，「RUP 模型至 RSx 模型對映」章節討論 RSx 如何支援 RUP 所定義的模型類型。

後續的幾個章節提供建構各種模型的指引。其中有些章節討論模型的不同使用方法，您可以根據在流程、建模方法及架構控制上所需的嚴謹程度來採用。

最後會討論在團隊中使用模型的相關課題。其中提到管理規模及如何讓團隊成員分享模型來降低檔案爭用和合併衝突。

2. 基本概念及術語

模型

在 RUP 中，模型代表一個問題或解決方案領域從特定觀點來看的一套完整規格。可以使用代表領域或系統上不同視景的一些模型來指定問題領域或系統。RUP 提出一套獨立的模型：

- 商業使用案例模型
- 商業分析模型
- 使用案例模型
- 分析模型（可納入「設計模型」中）
- 設計模型
- 實作模型
- 部署模型
- 資料模型

RUP 與工具無關。就 RUP 而言，模型可畫在餐巾或白板上、可能是建模工具裡的一部分，甚至只是內心的想法而已。就 RUP 的觀點而言，模型是邏輯概念。

在 RSx 環境中，我們可以用邏輯術語來討論模型，但也可以用實際術語來討論。假設您有一些團隊在開發

兩個應用程式：第一個團隊有三位分析師，負責時程表管理應用程式，第二個團隊有五位分析師，負責客服中心應用程式。兩個團隊目前都在努力收集需求，並以 RSx 進行使用案例建模。以 RUP 的用語來說，您可以說一個團隊正在建置時程表應用程式的使用案例模型，另一個團隊正在建置客服中心應用程式的使用案例模型。但如果團隊採用 RSx，則必須知道模型有實際的意義表達。這是下一節的主題。

建模檔

RSx 模型會儲存爲檔案。（以 Eclipse 術語來說，檔案就是「資源」[1]，如果您在本文或其他原始資料中看到「建模資源」這個術語，就是指「建模檔」）。RSx 大致上支援兩種建模檔：

- **實作前建模檔**在主機 OS 檔案系統中儲存爲個別檔案。副檔名是 .emx。檔案中包含
  - UML 語意元素（類別、活動、關係...等）
  - 描繪 UML 語意元素的 UML 圖（也可能以視覺化參照來描繪其他語意領域中的事物，例如 Java、C++ 或 DDL）。

- **實作建模檔**在 Eclipse 工作區儲存爲 Eclipse 專案。專案包含
  - 實作構件（Java 程式碼、網頁、XML Meta 資料檔...等）
  - 描繪且直接反映實作構件的圖檔。

  模型語意存在實作構件之內。例如，Java 實作的語意模型會經過序列化儲存爲一群 Java 原始程式檔。每一個圖型在專案內有自己的實體檔。圖檔可能有各種副檔名。最常見的是 .dnx。實作建模圖通常採用 UML 表示法，但也可能採用其他表示法（例如，IDEF1X 或「資訊工程」的資料視覺化表示法，或 Web 層設計所用的 IBM 專用表示法 ）。

本文的重點在於如何組織「實作前」模型的內部結構。**在本文的其餘部分，「建模檔」就是指「實作前建模檔」**（.emx 副檔名的檔案）。在其他資料來源中也可以找到實作專案內容的編排原則，例如 Rational Software Architect、Rational Application Developer 及 Rational Web Developer Community Edition 的線上說明。

建模檔不一定含有一個（邏輯）模型的所有資訊。事實上，建模檔通常只包含模型的一部分。在上述範例中，有一個三人團隊負責時程表應用程式的使用案例模型。該團隊可能選擇將使用案例模型實際分割成三個建模檔，如此一來，每位團隊成員可以各自負責一部分的使用案例，不必爭用同一個檔案。本文最後一個章節討論分割模型和管理建模檔的相關課題。

模型類型

在 RUP 中，模型有明確類型，例如使用案例模型、分析模型或資料模型。在 RSx 中，建模檔沒有「嚴謹類型」，但您可以依慣例使用多個「類型不嚴謹」的建模檔。如果想要採用這個慣例，您有兩種方法可以建立不嚴謹類型：

- 從「空白」建模檔開始（參見下文），根據如何命名和放入何種內容來建立類型（包括套用哪些 UML 設定檔）
- 根據預先定義代表特定模型類型的「範本模型」來建立建模檔。對於本文描述的模型類型，RSx 產品已提供一組預設的範本模型。您也可以建立自己的範本模型（請參閱產品說明及 developerWorks 論壇和其他資源）。

不論任何方法，在 RSx 中，建模檔的「類型」純粹只是有關檔案命名和內容的慣例而已。例如，此工具不禁止含有使用案例模型的檔案也包含實現使用案例的類別（在邏輯術語上，RUP 只是分析或設計模型的一部分）。

[1] 在 Eclipse 中，資源就是檔案，只是在 Eclipse 環境內還有其他內容和行爲。這裡說明的「建模檔案」被 Eclipse 視爲「資源」。

這些準則希望您將 RSx 建模檔視爲具有類型。

工作區、專案及專案類型

熟悉 Eclipse、WebSphere Studio 產品或 Rational Application Developer 的讀者應該知道檔案位於「專案」內、「專案」可以是各種類型、「專案」在「工作區」內分組和管理。就像其他檔案一樣，RSx 建模檔也在專案內。

就本討論而言，不需要詳細解釋 RSx 和 Rational Application Developer 中可用的所有專案類型。我們比較關心兩種專案：

- **UML 專案**
- **實作專案**，包括特殊化專案類型，例如企業專案、EJB 專案、Web 專案及 C++ 專案

先前說過 RSx 支援兩種建模檔：

- .emx 副檔名的檔案，包含在高於實作的抽象層次上用於建模的 UML 模型（需求、分析、設計）
- 內含實作語意的 Eclipse 專案（通常會序列化爲程式檔構件）和反映這些語意的圖型。

將模型分配到專案的規則很簡單：

a) 將「實作前」建模檔放入 UML 專案中

b) 實作模型自我管理，因爲基本上：

[實作模型] = [實作專案]

這項規則有許多例外情形。下列 UML 建模檔適合放在特定語言的實作專案內：

- 設計「草繪模型」（在稍後章節中討論）。
- 包含說明測試之序列圖的模型，而其將會配合專案中的程式碼執行。

> *XDE/Rose*
> Rose 和 XDE 作業原理包括反覆修正「設計模型」，直到達到同等的程式碼抽象層次爲止，然後利用程式碼模型同步技術，讓此模型和程式碼本身的語意保持一致。例如，在 XDE 中，實作模型在專案中不僅以程式碼和圖型的形式存在，也以「程式碼模型」檔案的格式存在，與實作構件分開保存，且在本質上是語意的備用副本。
>
> RSx 作業原理建議在高於程式碼的抽象層次上採用與平台無關的模型（例如「企業 IT 設計模型」），且利用轉換從這些模型中產生程式碼。然後，在程式碼抽象層次上，RSA、RSD 及 RAD 只是讓您以 UML 表示法來繪製程式碼語意圖，在實作抽象層次上不需要再用到另外保存的語意模型。
>
> 請注意，RSx 不禁止您在程式碼抽象層次上定義 UML 模型，然後從中產生程式碼，事實上這種作法很常見。但 RSx 不提供技術來自動保持這種模型和程式碼的同步化。

概念回顧

下圖總結先前的討論。此圖例反映稍早的情況：其中有兩個團隊，一個進行服務中心應用程式，而另一個進行時程表管理應用程式。

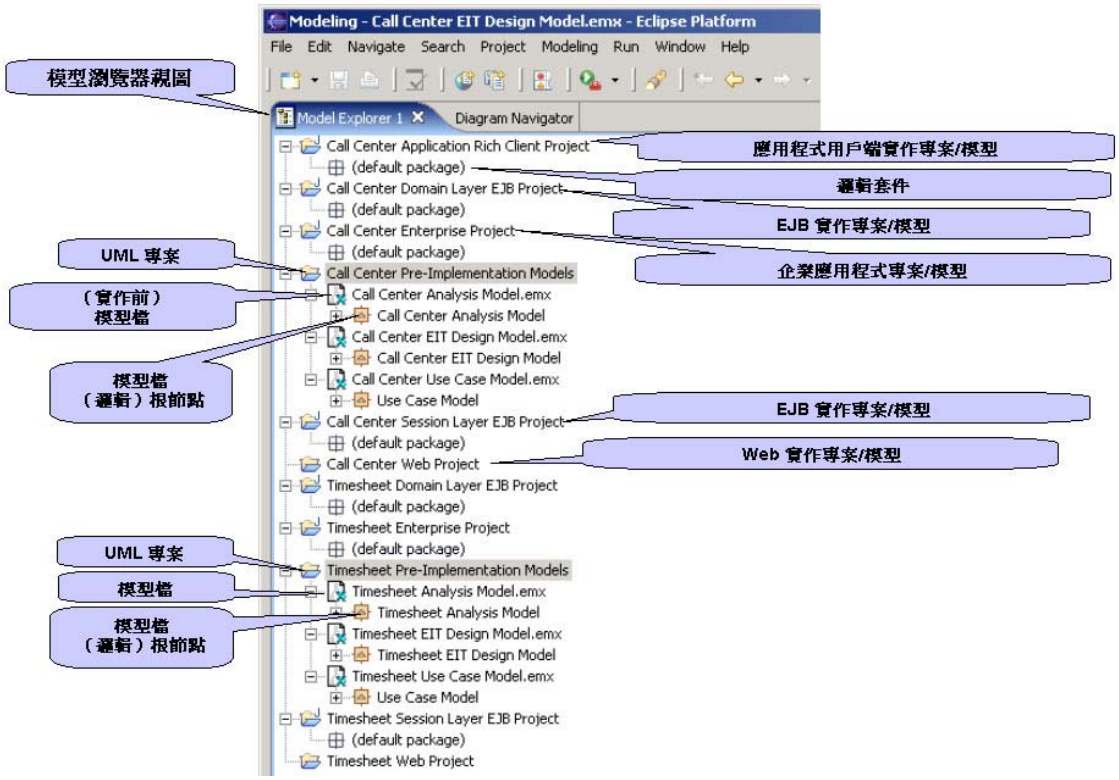


圖 2-1

RSX 提供瀏覽器視圖[2]，呈現兼具實體和邏輯觀點的模型。在瀏覽器中，可看到專案在工作區裡是最上層節點，在每一個專案內，可看到屬於此專案的資源。在**圖 2-1** 中，我們看到「模型瀏覽器」中有一群專案，對應於我們舉例的兩個應用程式。我們看到 UML 專案用在實作前模型，也看到一群適當解決方案類型的專案用於實作模型。

> *XDE/Rose*
> 相對於 RSA 模型瀏覽器，Rose 和 XDE 中的模型瀏覽器僅提供模型的邏輯觀點。請注意，「RSA 模型瀏覽器」提供的資源視圖並不是「Eclipse 導覽器」視圖提供的「純正」實體觀點。「模型瀏覽器」中可看見一些實體資源時，大多是由指出資源之*邏輯*觀點的圖示來表示。

在圖 1-2 中，我們展示「時程表」使用案例模型在內部可能如何組織成套件，代表問題領域中一些功能類似的子集。

[2] 在 6.x 版中是指「模型瀏覽器」。到了 7.0 版，模型出現在一般用途的瀏覽器中。本文的圖例以 6.0 版爲主，說明「模型瀏覽器」。

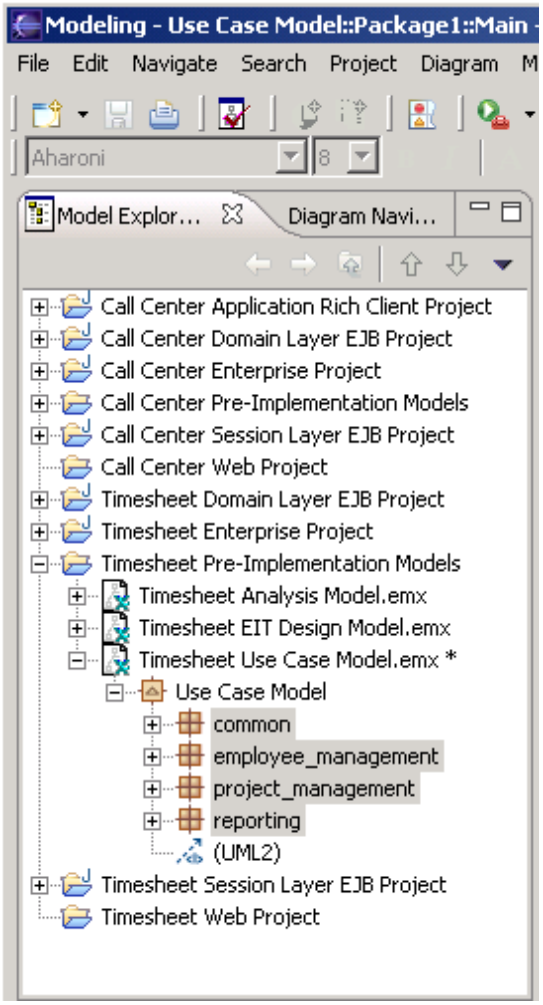


圖 2-2

在**圖 2-1 和 1-2** 中，每一個實作前模型都在單一建模檔中。在圖 1-3 中，我們展示「時程表」使用案例模型如何重構爲多個建模檔，對應於問題領域的相同子集。在構成完整使用案例模型的所有建模檔之間，爲了維持一致的名稱空間，請注意每一個建模檔的根如何命名。

3. RUP 模型至 RSx 模型對映

下表顯示最常用的 RUP 模型依慣例如何對映至 RSx 建模檔類型。對映通常很直接明確，但在以本文爲準則來使用 RUP 與 RSx 時，實爲關鍵所在。表格中提到的 RSx 建模檔類型隨後立即討論。稍後章節會提供各種建模檔類型的內部組織準則，以及使用何種專案來存放。這些討論藉由此處列出的 RSx 建模檔類型來表示。

RUP 模型	*RSx* 建模檔類型
使用案例模型	根據「使用案例模型」範本的建模檔 （替代方法：從空白建模檔開始，按照 RUP 使用案例模型指導原則來限定內容）
分析模型	分析模型 （替代方法：從空白建模檔開始，按照 RUP 分析模型指導原則來限定內容） （替代方法：在「設計」模型中使用 <<analysis>> 套件）
設計模型	N-層商業應用程式：根據「企業 IT 設計模型」範本的建模檔 _ _ （替代方法：從空白建模檔開始，按照 RUP 設計模型指導原則來限定內容） 其他應用程式類型：從空白建模檔開始，按照 RUP 設計模型指導原則來限定內容 設計「草圖」：空白建模檔 選用性補充：做爲「實作概觀模型」的其他空白建模檔
實作模型	含有實作構件和圖檔的 Eclipse 專案
部署模型	從空白建模檔開始，按照 RUP 部署模型指導原則來限定內容

RSx 建模檔類型

空白建模檔

RSx 有選項可建立「空白模型」（檔案→新建→UML 模型→空白模型）。「空白模型」是不以模型範本爲根據的建模檔。沒有套用特殊的設定檔，除了有一個「主要」圖型（開放式），沒有預設的內容。**您可以用空白建模檔來當做任何模型類型的起點。**只要選擇如何命名、定義什麼內容及套用什麼設定檔，您可以用空白建模檔來建置使用案例模型、分析模型、設計模型、部署模型或其他任何類型的 RUP 模型。

使用案例建模檔

RSx 提供選項可根據模型範本來建立「使用案例模型」檔。範本提供的預設內容如*圖 3-1* 所示。（「建置區塊」內容和搜尋字串的使用說明已超出本文的範圍。您應該會發現範本中包含的指示主要是自我闡明的。）

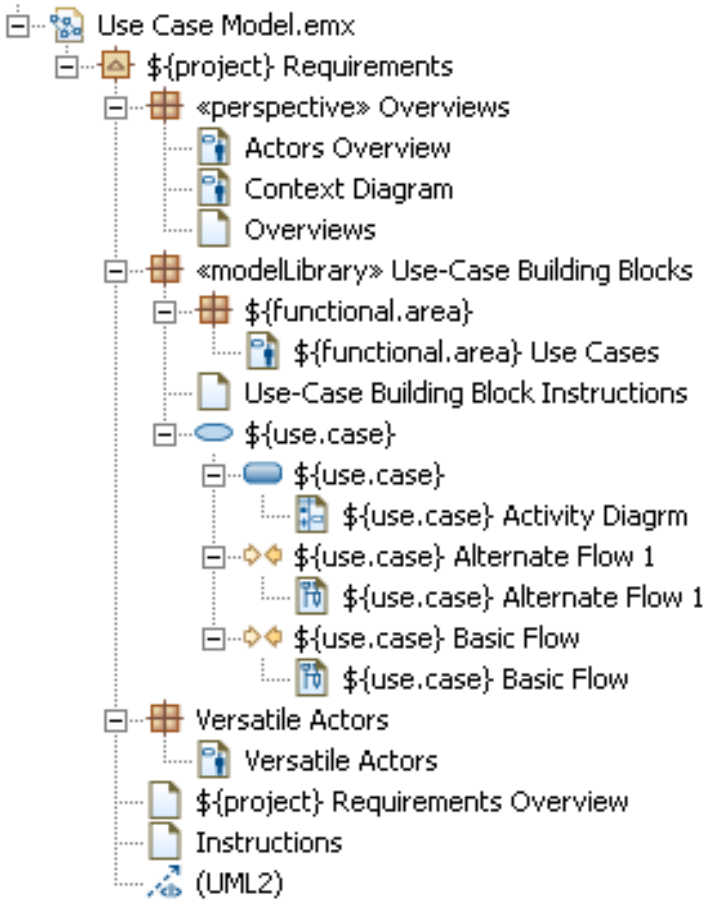


圖 3-1

分析建模檔

RSx 提供選項可根據模型範本來建立「分析模型」檔。範本提供的預設內容如*圖 3-2* 所示。此外，以這個範本建立的模型檔也會套用「分析」設定檔：

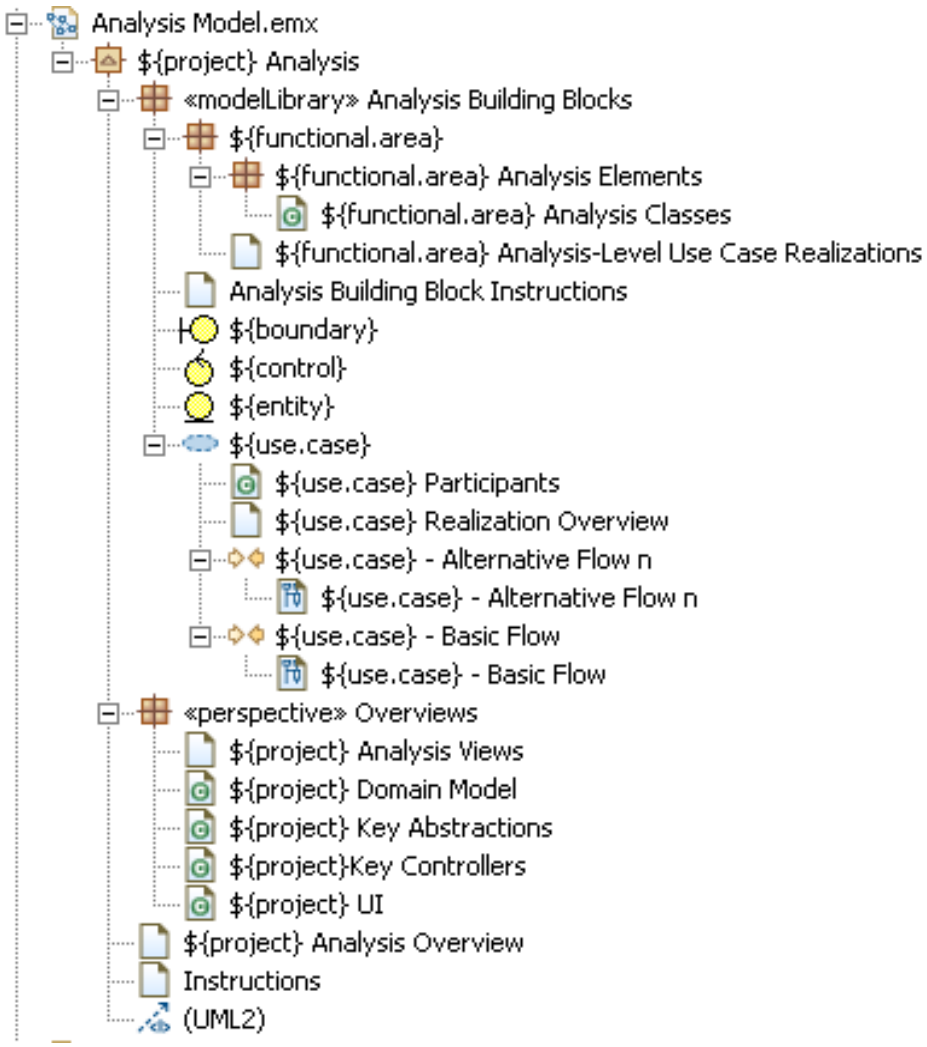


圖 3-2

企業 IT 設計建模檔

RSx 提供選項可根據模型範本來建立「企業 IT 設計模型」(EITDM) 檔。範本提供的預設內容如**圖 3-3** 所示，此外，「EJB 轉換」設定檔[3] 會套用至以這個範本建立的模型檔。以商業應用系統爲目標且利用 RSX 程式碼產生轉換來支援建立這種應用程式時，適合在設計（或分析）上使用這個範本。

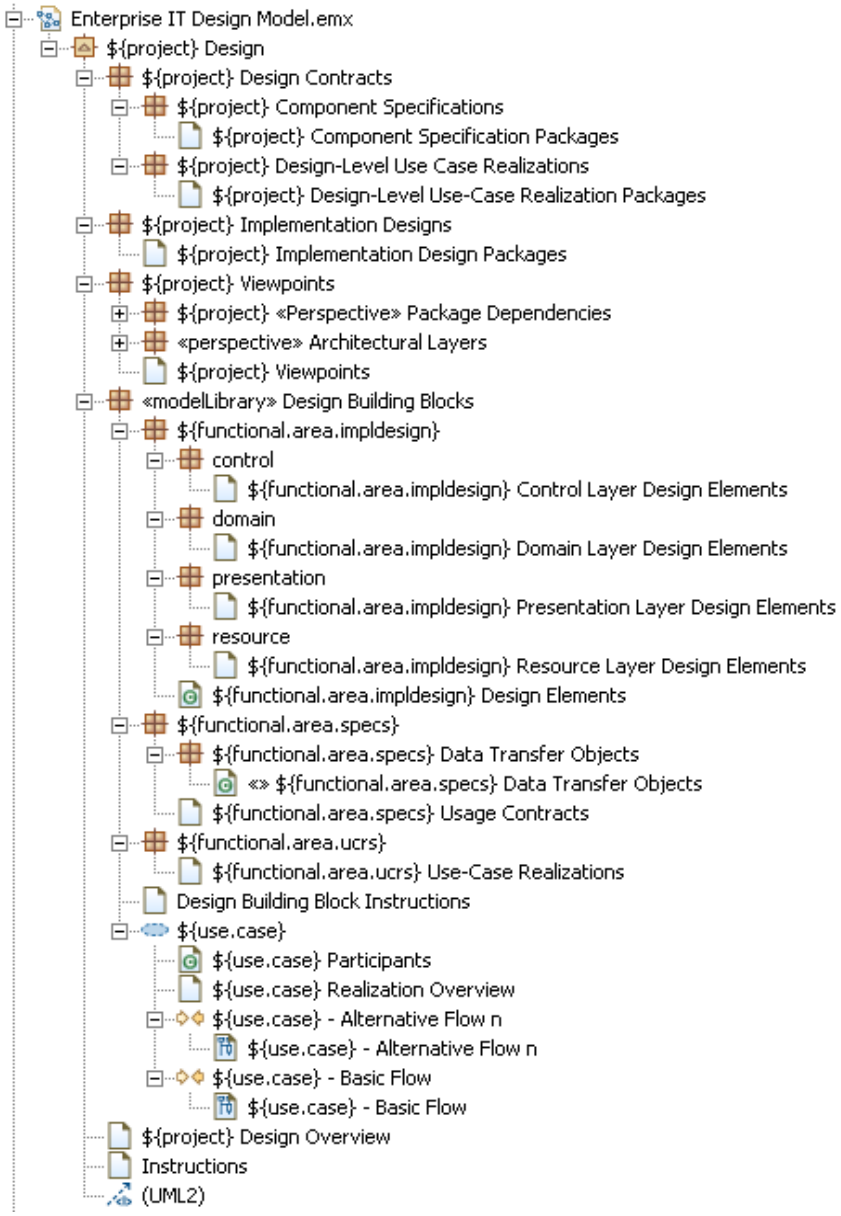


圖 3-3

[3]「EIT 設計模型」範本中的可轉換設定檔會隨著產品更新而進化。

實作概觀建模檔

在設計模型中，您可能覺得最好另外定義一個「實作概觀模型」來捕捉實作如何組織的整體觀點。「實作概觀模型」適合在設計階段初期使用 – 在產生或撰寫程式碼之前 -- 代表用來存放程式碼和相關檔案（Meta 資料、部署描述子等）的實際 RSx 或 Rational Application Developer 專案和資料夾/套件。您也可以用這個模型來顯示這些專案和套件之間預期的相依關係，有助於釐清系統建置需求。「實作概觀模型」也可以是保存解決方案架構之非正式概念性圖型的位置。

實作模型

如先前所述，在 RSx 中，實作模型由一個專案組成，內含實作構件及（選用）描繪這些構件的圖型[4]。

草繪模型

如「基本概念及術語」章節所述，您可以將設計模型視爲正式的架構圖，在整個系統生命週期內持續維護，用來支援/施行架構控制。或者，也可以視爲草稿，用來提供設計建議，協助澄清和溝通，一旦實作開始之後就丟棄。RSx 支援這兩種方法。其特性通常不針對任一方法，您選擇如何使用設計模型確實有助於決定採用哪些 RSx 特性及如何使用。在本文提出的準則中，有需要區別時，將以「草繪模型」來強調模型的「拋棄式」性質。

[4] 若要建立這些圖型，不要使用「檔案→新建→UML 模型」來建立模型，應該使用「檔案→新建→類別圖」來建立圖型，然後以 UML（或其他）表示法來撰寫程式碼的「視圖」。每一個圖型儲存爲獨立的檔案，副檔名爲 .dnx，且可以受版本控制，很像程式檔。這些圖型不含任何語意資訊，只有表示法。所有相關的語意資訊已在程式碼內。在其中一個圖型中變更時，例如變更類別名稱或作業簽章，實際上等於變更底層的程式碼。在程式碼中變更時（使用文字編輯器），變更的程式碼所在的圖型也會自動更新。

4. 模型內部結構組織的一般準則和技術

組織 UML 模型內容的主要工具是套件。UML 套件有兩個主要用途：

- 分割、組織和標示模型資訊
    - 將對應於問題或解決方案領域中特定主旨事項的元素進行分組
    - 區別不同類型的模型資訊，例如介面、實作、圖型等
    - 將元素進行分組，以便定義和控制它們對其他元素的相依關係
    - 將圖型分組，在相同模型上提出另外的觀點
- 建立名稱空間
    - 模型元素
    - 從模型元素產生的實作構件（這可能涉及模型和實作語言名稱空間之間的對映）
    - 針對重覆使用的單元

同樣地，RUP 已爲各種模型類型提出特定的封裝策略。這些策略反映在本文件內特定模型類型專用的章節中。RSx 也提供其他一些組織工具，如下所述：

使用 <<perspective>> 套件來表示觀點

如需以多種組織方式來檢視元素，您可以建立更多套件，以圖型來描繪替代的組織設計。在模型的整個封裝設計上，只要有需要表示特定的觀點或模型內容，就可以運用這種技術。RSx 在 UML「基本設定檔」中提供 <<perspective>> 套件模板來支援這項技術。大致上，您可以將 <<perspective>> 套件視爲等同於 RUP for Systems Engineering 或 IEEE 1471 - 2000 的「見解」。

請勿將語意元素（類別、套件、關聯等）放入 <<perspective>> 套件內。只需要放入圖型，根據替代的組織性課題或應用見解來描述觀點。將 <<perspective>> 模板套用至套件有幾項用途。可在視覺上指出套件代表特定的見解。也支援模型驗證規則，當 <<perspective>> 套件中被放入語意元素時會提出警告。同時也充當套件的指定元，在 RSx 轉換時必須跳過

使用主題圖建立特定事項的自我更新敘述

相較於在「一般」圖型中手動放入您要描述的元素，「主題圖」的內容是經由查詢現有的模型內容而決定。若要建立「主題圖」，請選取「主題式」模型元素，再根據其他元素和主題元素的關係類型，定義您要在圖型中顯示的其他元素。當模型的語意內容變更時，「主題圖」會隨之調整。

透過瀏覽圖來查驗模型

「瀏覽圖」不是特別用來組織模型的工具。它們的目的是協助探查和瞭解模型的內容，而不需要手動編製圖型。但在模型組織方面，最好知道這種圖，因爲可讓您儘量不必編製要保存的圖型。進一步可降低模型的大小和複雜性，組織起來更容易。

瀏覽圖有點類似「主題圖」，主要差別在於「瀏覽圖」不用保存，只是即時產生。若要產生「瀏覽圖」，請選取模型元素（從圖型或「模型瀏覽器」中），再利用快速功能表來「探索瀏覽圖」。這會產生一個圖型，將選取的元素描述成「焦點」，相關的元素會顯示在該焦點周圍的放射狀版面配置中。當然，您可以接著在此「瀏覽圖」中選取其中一個相關元素，設爲另一個「瀏覽圖」的焦點，依樣畫葫蘆繼續下去。

圖型間的導覽

在 RSx 中，有兩種在圖型之間導覽的機制：

- 可能從「模型瀏覽器」將圖型節點拖曳到一些其他的「主控」圖。然後，在主控圖上按兩下產生的圖示，即可開啓參照圖。

- 每當在模型中建立新的 UML 套件時，會自動建立「主要」圖型（開放式圖型）。根據預設值，這個「主要」圖型會建立成套件的「預設」圖型。您可以將圖型重新命名爲「主要」以外的名稱，但仍然還是「預設」圖型。您也可在套件中選取不同的圖型，並使其成爲該套件的「預設」圖型。「預設」圖型的用途：如果將套件本身放入其他「主控」圖型中，您可以按兩下套件來開啓預設圖型。

這些機制支援下列組織準則，適用於任何模型類型：

1. 編製每一個建模檔的「主要」圖型（或其他預設圖型）來描繪
    a. 建模檔案中的每個最上層套件
    b. 在建模檔的根套件內，其他任何圖型的圖型圖示（換言之，不描繪預設圖型本身的圖示）
2. 編製每一個最上層套件的「主要」圖型（或其他預設圖型）來描繪
    a. 直接包含的套件
    b. 其直接包含之任何其他圖型的圖型圖示
3. 對每一個連續更下層的套件重複這個模式

5. 使用案例模型內部組織的準則

使用案例模型高階組織

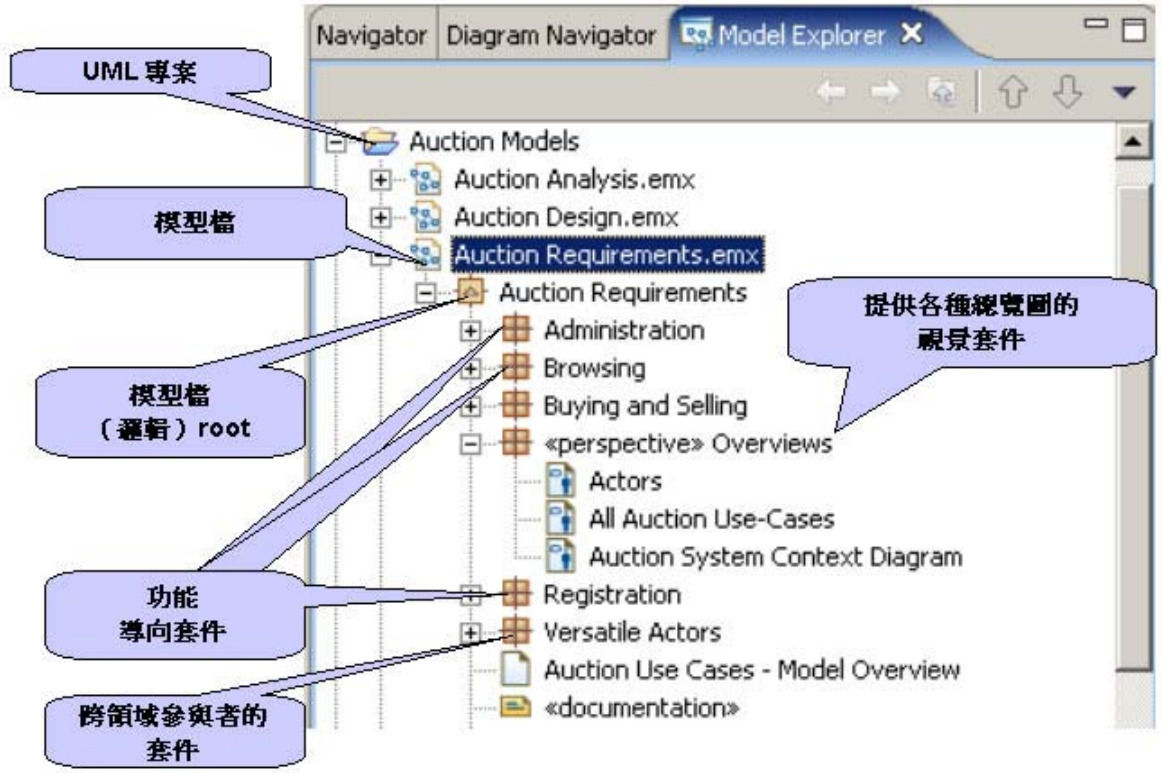


圖 5-1

圖 5-1 說明下列的使用案例模型建構準則：

1. 使用最上層套件建立功能導向的分組。理由：

    - 當一群人負責「使用案例模型」時，這通常可適當反映人力分配的問題。如果後來檔案爭用逐漸成爲問題（您可能只爲最上層套件建立個別的建模檔），而決定將使用案例模型分割成多個建模檔，也不會有問題。

    - 與其他組織方法比較，這通常較適用於對映至最終實作的組織。如果要以轉換來植入每一個連續更低的抽象層次，這就很重要。尤其，如果要根據使用案例模型來產生分析模型的種子內容，您一定希望使用案例模型的封裝結構可以適當地對映至目標分析模型中適合的封裝結構。輪流地，您也希望分析模型的封裝結構可以適當地對映至設計模型，然後設計模型的封裝結構也適當地對映至將包含實作的專案集。這些對映愈簡單，轉換抽象層次所需的配置工作就愈輕鬆。

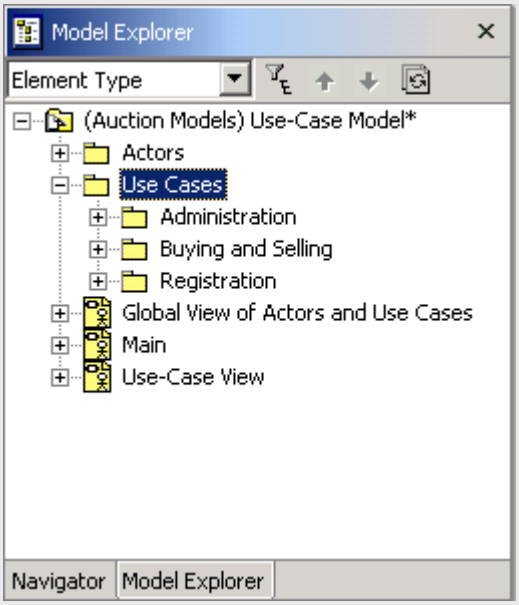
2. 使用其他最上層套件來擷取「廣泛授權」或「多用途」的參與者。

3. 利用 <<perspective>> 套件中的圖型來擷取高階、整體的「使用案例」觀點。理由：

    - 保持將模型的語意元素組織爲功能導向的分組時，提供整體觀點和「重要架構」使用案例觀點。

XDE/Rose

RSX 指導原則稍微修正使用案例模型傳統的高階組織原則，在傳統原則上，「參與者」會建立一個套件，「使用案例」也會建立一個套件。然後，隨著模型的大小和複雜性的要求，您將採用較低階的套件來進行功能導向的分組，如下列 XDE 型範例所示：



使用案例模型內容

詳細的指導教學已超出本文的範圍，本文無法詳述如何撰寫好的使用案例，或良好的使用案例建模應該做什麼或不該做什麼。不過，這裡稍微討論在使用案例模型中除了「參與者」和「使用案例」之外，還可以加入什麼。

- **建議：**在模型根上建立「主要」圖型來描繪模型的其他套件，並支援往下探查這些套件及各自的「主要」圖型。
- **建議：**在每一個使用案例套件中，加入圖型來描繪套件使用案例、任何關係及參與其中的「參與者」。（如果案例數量太大，可適量使用多個圖型）
- **建議：**在「文件」欄位中描述每一個使用案例的主要和備用流程[5]（請參閱**圖 5-2** ）

[5]使用案例說明範例中的格式是先利用 RTF 編輯器來建立使用案例說明的文字「範本」，再將範本剪貼至使用案例說明欄位。

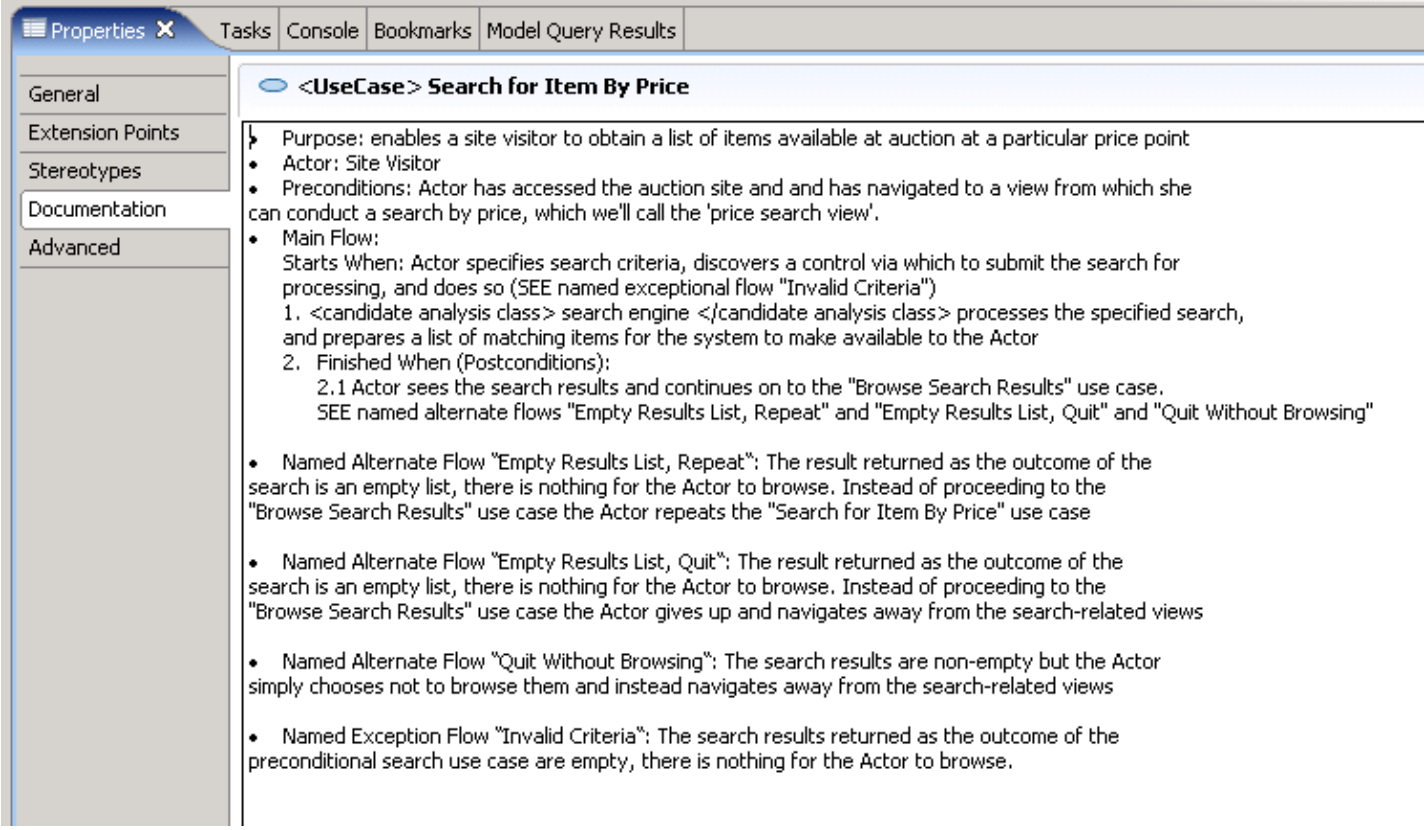


圖 5-2

- **選用：**當使用案例的複雜性不可避免時，請新增「活動圖」來編製，反映使用案例的整體活動流程。（請參閱**圖 5-3**）**理由：**有助於指出對應於每一個（主要及替代/異常）流程的條件，也有助於確定所有不同的流程最後會再次會合。（在 RSx 中新增「活動圖」會自動將一個「活動」新增至使用案例，圖型會放在「活動」下方）。

- **選用：**爲使用案例的每一種指定的（主要、替代及異常）流程塑造「黑箱」實現化：將合作事件新增至使用案例；加入對應於使用案例主要流程的互動實例，以及每一個指名的替代和異常流程的互動實例；爲每一個互動實例編製序列圖（或通訊圖）。請勿混淆這些使用案例合作實例和分析層次的使用案例實現化（如「分析模型」所述）或設計層次的使用案例實現化（如「設計模型」所述）。這些是使用案例的「白箱」實現化，描述一個解決方案內部各元素之間的互動。這裡爲「使用案例模型」所提出的合作事件完全是參與者和系統之間的「黑箱」互動。（請參閱**圖 5-3**）**理由：**讓非技術性的關係人可以從整體的角度來觀察系統使用者和系統如何互動。也有助於您指出實作時所需的各種視圖（螢幕或頁面）。同時也正式命名使用案例的各種流程（情境），將這些名稱指定給語意模型元素（亦即給合作事件）。

> *XDE/Rose*
> 在 UML 1.x 中，您在這方面是用「合作實例」，而非「合作事件」。

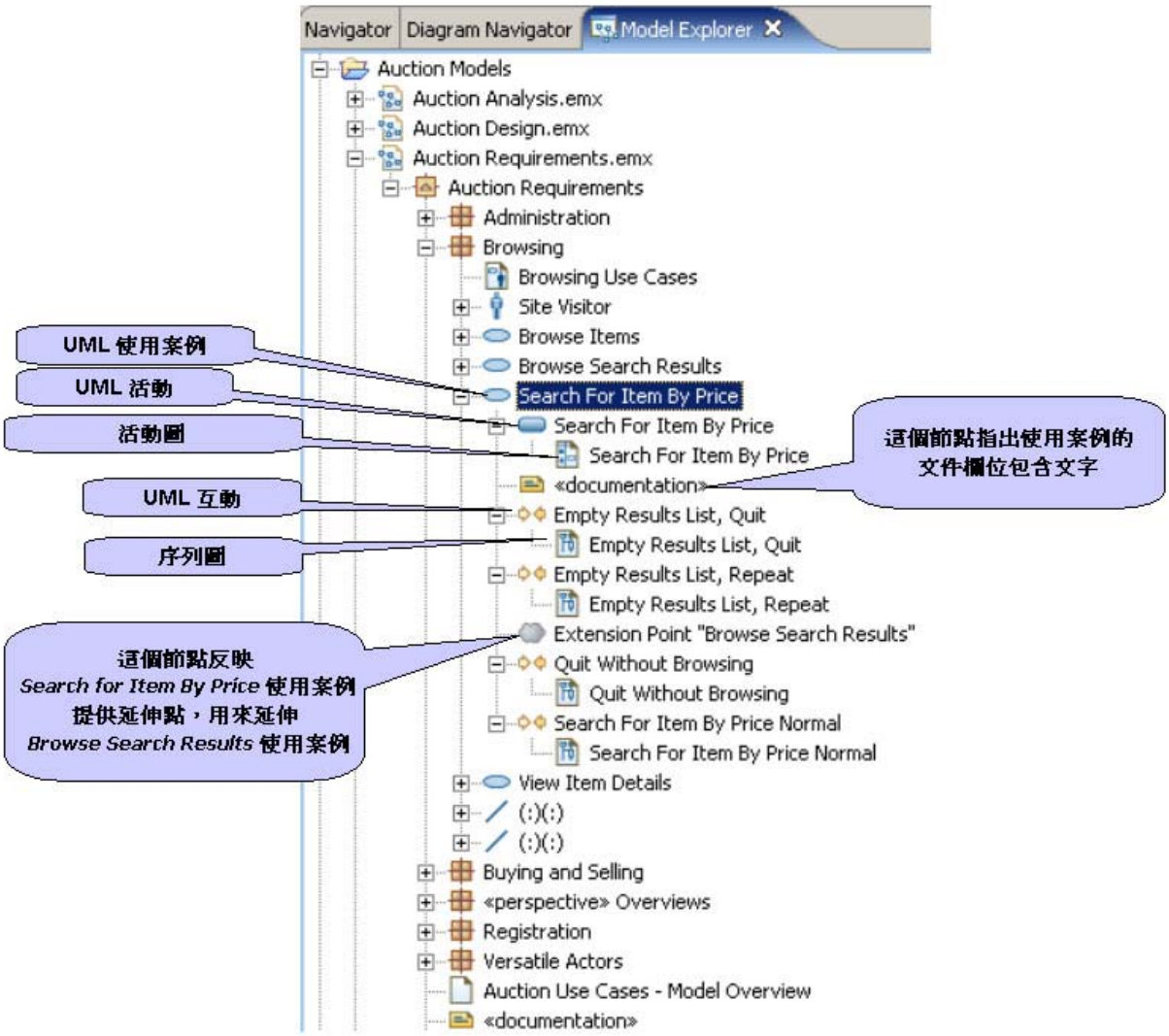


圖 5-3

- **選用：**如果您遵守 RUP 指導原則來找出架構的「重大架構性」觀點，尤其在需要維護「軟體架構文件」時，請新增一個最上層 <<perspective>> 套件來包含使用案例圖，描繪重大架構性使用案例。您可能會將此套件命名爲「架構的使用案例觀點」。

6. 分析模型內部組織的準則

「分析模型」代表解決方案展開的第一層面。是從需求進入最終設計的一塊墊腳石，著重在捕捉商業領域的相關資訊，並以最接近商業的高階抽象層次來顯示候選的解決方案元素。也是「分析類別」和分析層次「使用案例實現化」出現的地方。經由塑造使用案例實現化的過程（主要是使用序列圖），您將開始*探索*解決方案所需的類別 – 尤其，這些類別將對應至您認爲在序列圖中需要的生命線。也可以運用一些經驗法則，根據使用案例模型的內容來建議分析模型內容。本節稍後會提到。

在 RUP 中，「分析模型」和「設計模型」是否分開維護視各專案而定，根據您是否相信分開維護「分析模型」的價值是值得投入的時間來做決定。如果建立獨立的「分析模型」，但不維，則「分析類別」將移至設計模型來修正。或者，分析模型可能逐漸進化成設計模型[6]。在產品專用的術語中，這裡是一些您可考量的選項：

1. 建立分析模型，放在建模檔中（或根據「分析模型」範本產生的一組檔案）。然後，透過手動流程或自動化轉換，根據「企業 IT 設計模型」範本，在第二個模型檔中（或一組檔案）建立分析元素的修正版，最後刪除分析建模檔。這樣可讓您選擇繼續維護獨立的分析模型，或選擇放棄。
2. 根據「企業 IT 設計模型」範本在建模檔案（或一組檔案）中執行套用「分析設定檔」的分析層次建模。如此一來，您可以開始使用此分析類別來塑造使用案例實現化，隨著時間逐步修正，由設計介面來執行行爲。
3. 第二種和第三種選項混合的作法是將分析模型的一部分放在與設計模型相同的建模檔內一起維護。這需要將分析內容打散成套件，在套件上套用關鍵字 <<analysis>>。這樣就有機會在更精鍊的設計層次構件所在的建模檔中保留原有的分析層次構件。

在使用 RSx 轉換來產生實作時，請注意，這些轉換大多可接受分析層次元素做爲輸入，省去不少將這些元素修正成設計元素的手動步驟。以這種方式使用 RSx 時，最好採取上述的第 2 種或第 3 種作法。封裝成 RSx 一部分的標準程式碼產生轉換會略過有 <<analysis>> 關鍵字的模型套件。

[6] 事實上，RUP 會選擇在設計模型中建立分析類別和分析層次使用案例實現化，然後從這裡直接進化成設計形式。在該方法之下，「探查」到設計模型時您可沿用保留部份「純正的分析」視景的方法來建立套件。

分析模型高階組織

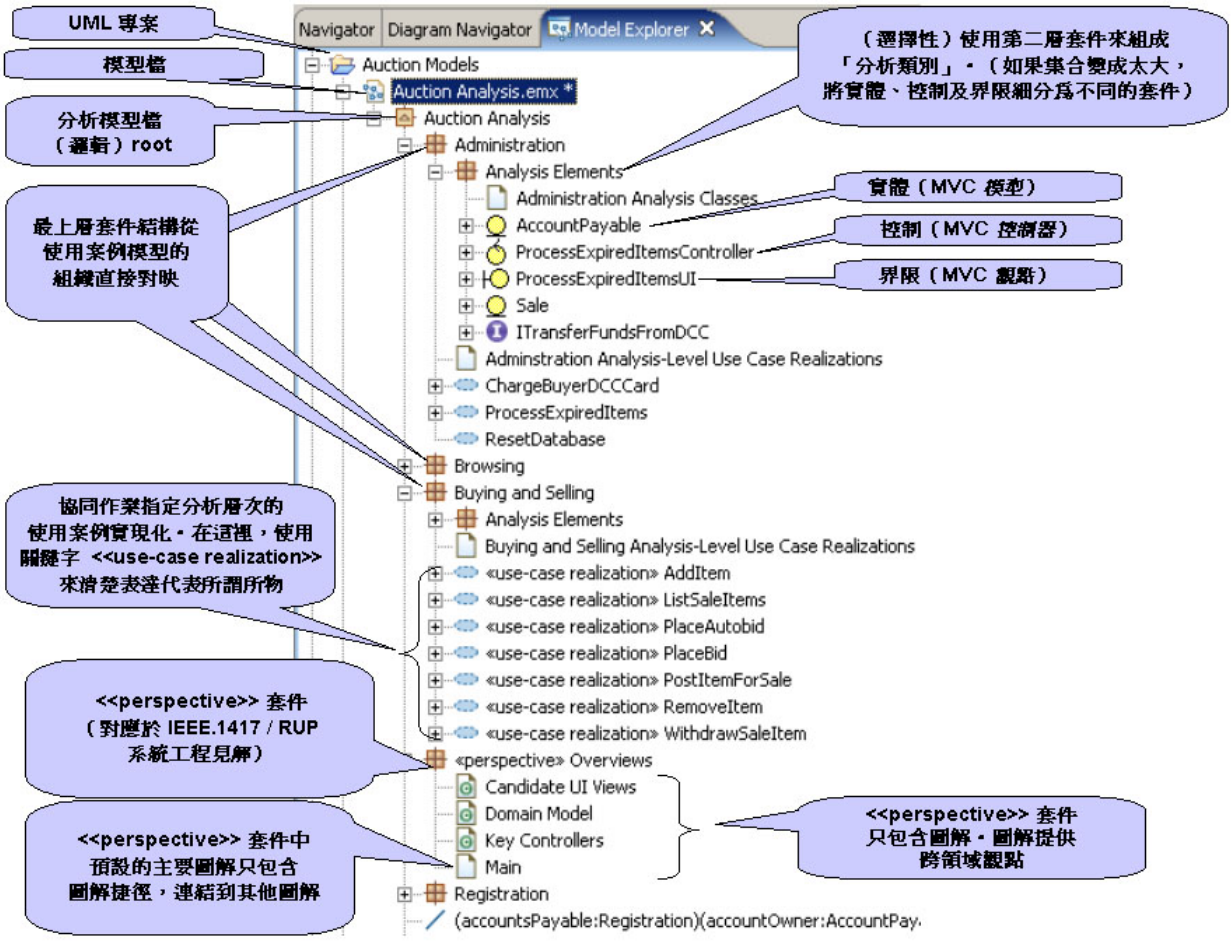


圖 6-1

圖 6-1 說明下列的分析模型建構準則：

1. 利用最上層套件對分析類別進行功能導向分組。理由：理由同於使用案例模型。

2. （選用）在最上層套件內，利用子套件來收集和組織分析類別。

3. 利用 <<perspective>> 套件中的圖型來捕捉分析元素的替代、高階或整體觀點。理由：讓不同的關係人有不同的看法，同時保持以功能導向分組來編排模型的語意元素。

這種方式可以稍微變化，請參閱下列圖 6-2 我們看到圖中利用最上層套件來隔開使用案例實現化和分析類別。在這個最上層套件內，有一組功能導向的子套件符合一組最上層套件。以這種方式來隔離使用案例實現化，可以重新分解含有分析類別的套件結構，但未必會影響使用案例實現化的組織。（尤其，如果分析模型將*就地*進化成設計，則類別的套件組織也很可能會進化，以致於不符合使用案例原先的組織。）

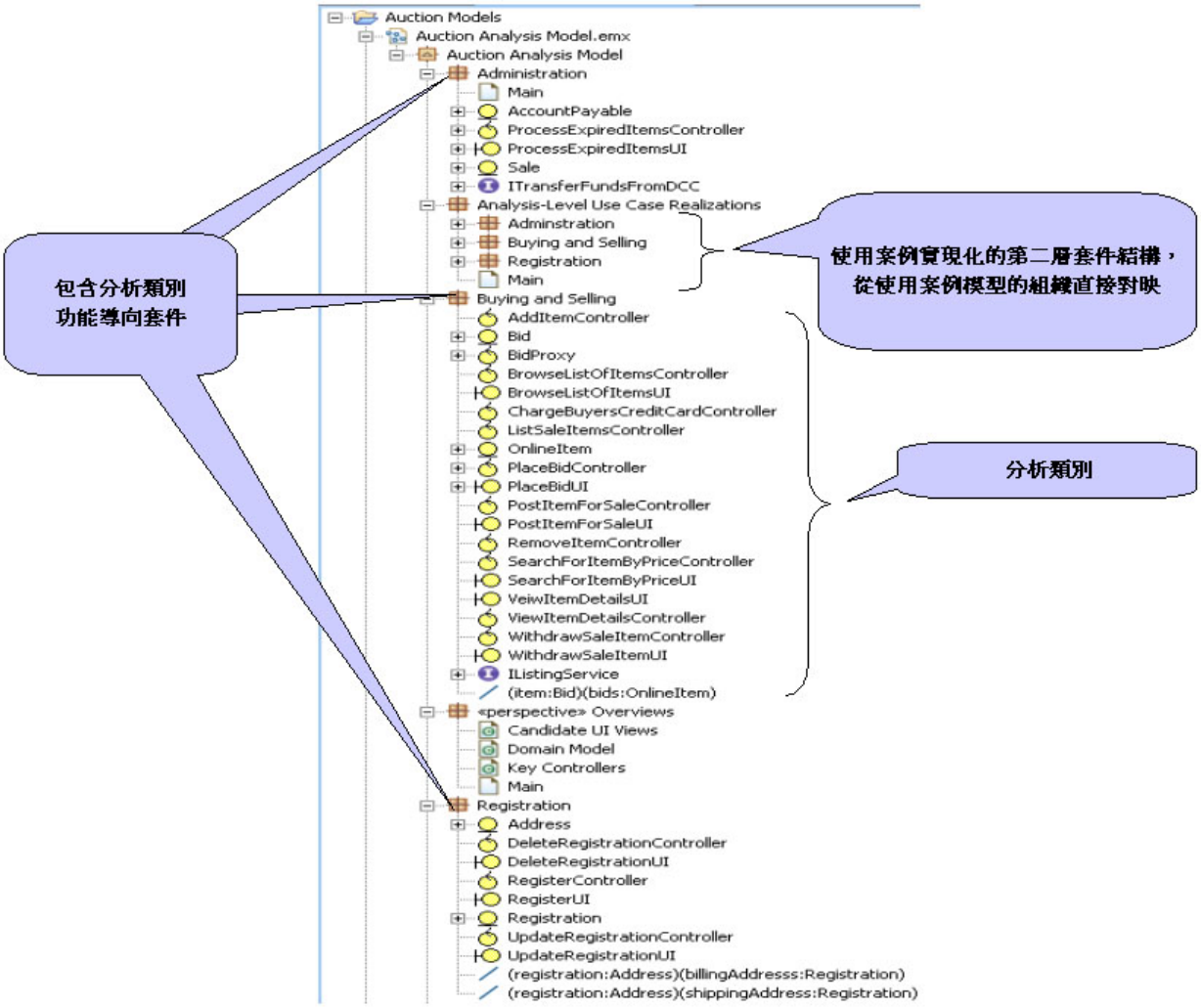


圖 6-2

視情況而定，顧及可能需要合併和重複使用多個獨立團體所建立的模型內容，甚至包括不同（夥伴）企業中的團體，有時需要導入命名慣例來預做準備。有此顧慮時，請採用顛倒式網際網路網域名稱空間慣例，如圖 6-3 所示。請注意，就分析模型本身而言，這或許不重要，但如果選擇讓分析模型*就地進化*成設計模型，且預料在設計層次上有重複使用和商業整合的情形出現，您可能需要做好事前規劃。採取這種方式可能有另一項優點：由於可適當對映至從分析/設計中產生的程式碼組織，因此可以簡化後續爲程式碼產生轉換所做的配置工作。

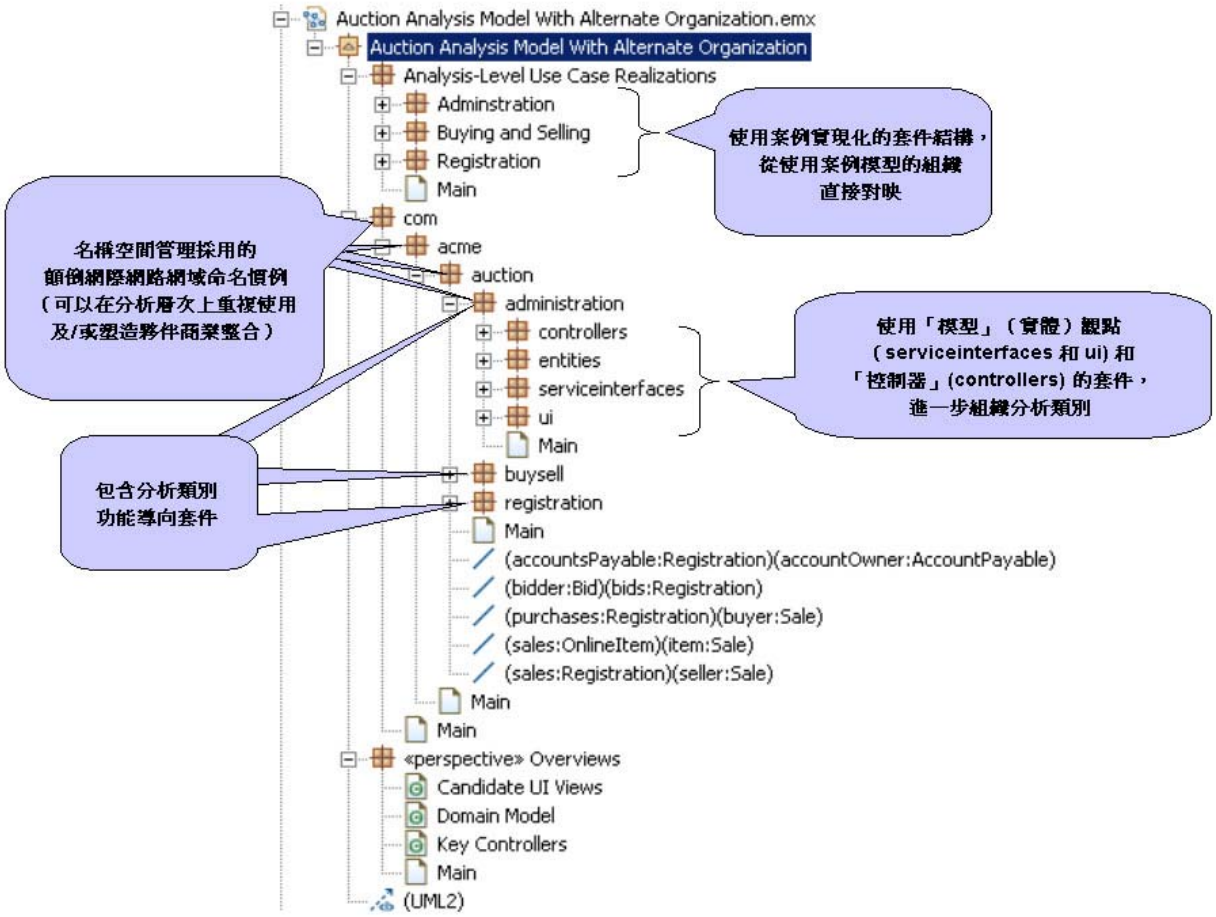


圖 6-3

分析模型內容

有數個方法可探查何謂分析類別。其中一種方法是開始繪製序列圖來建議使用案例實現化。在過程中，您會找到所需的生命線，且每一條生命線通常對應於一個可能的分析類別。依此來探索類別時，您可能在分析模型的實現化套件中建立類別，但請勿將類別留在這裡。應該「重新分解」模型，將分析類別移至功能導向的套件，如稍早在分析模型的高階組織準則中所述（請參閱**圖 6-1**）

另一種探索分析類別的實用方法：根據這些經驗法則在分析模型中「植入」類別：

- 對於每一個使用案例（在使用案例模型中），在分析模型中新增一個 <<control>> 類別。<<control>> 類別代表使用案例相關的商業邏輯。（稍後在設計時，這種類別也會對映至階段作業管理等事項。）
- 對於每一個參與者/使用案例關係（在使用案例模型中），在分析模型中新增一個 <<boundary>> 類別。<<boundary>> 類別代表解決方案和真人參與者之間的介面，或解決方案和一些外部系統之間的介面。對應於人類參與者的 <<boundary>> 類別最後可能會對映到設計及實作中的一或多個使用者介面構件。對應於外部系統的 <<boundary>> 類別最後可能對映至設計和實作中的一些銜接層。
- 透過 CRC 卡片分析或使用案例說明的字彙分析，找出其他 <<control>> 類別（動詞）和 <<entity>> 類別（名詞）

採用這種植入方法來尋找分析類別時，您可以將類別直接放入功能導向的套件中，如稍早在分析模型的高階組織準則中所述（請參閱**圖 6-1** ）

無論如何探索分析類別，最初的功能性套件組織方式註定要改變。

選用：在分析類別套件內利用第二層套件，進一步組織這些套件的內容（請參閱**圖 6-4** ）

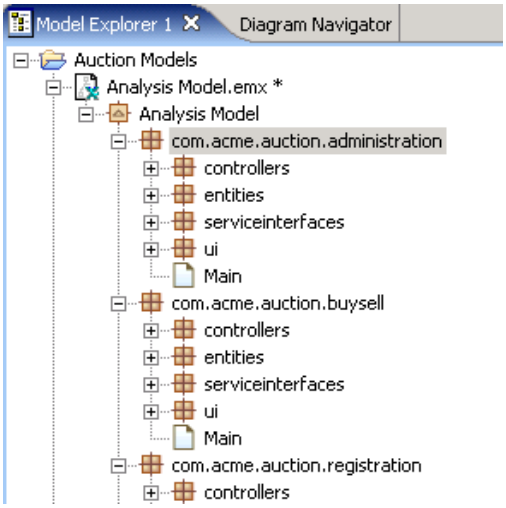


圖 6-4

建議：分析模型應該包含分析層次使用案例實現化，以分析類別來描述如何執行使用案例。每一個分析使用案例實現化（以「UML 合作」表示）可在使用案例模型中實現使用案例，且名稱同於該使用案例。請參閱**圖 6-5**。對於每一個指名的使用案例流程[7] 您覺得應建立爲分析層次實現化的模型，新增序列圖（會自動新增擁有端互動）。**圖 6-6** 顯示建立序列圖時將新增至模型的語意內容類型。（請注意，您可從「模型瀏覽器」視圖過濾任何 UML 元素類型，以便如**圖 6-6** 所述隱藏許多內容）

[7] 先前在「使用案例模型」中建立

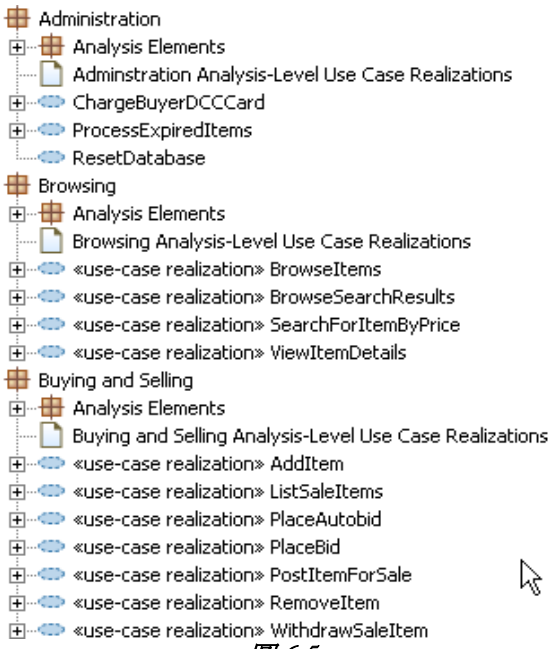


圖 6-5

選用：爲使用案例流程建立序列圖之後，您可以在「模型瀏覽器」中選取其本身的「UML 互動」，並加入「通訊圖」。新的「通訊圖」會自動移入有參與序列圖的分析類別實例。

建議：從使用案例模型的每一個使用案例實現化（UML 合作）和相對應的使用案例中建立「實現化」相依關係（請參閱**圖 6-6**）。因爲可以利用「主題圖」和「追蹤性分析」等特性來了解模型中的追蹤關係，實際上並不需要保留永久的圖型來描繪追蹤關係，建議您利用一些「拋棄式」圖型來建立關係，例如：

- 在「合作」中新增開放式圖型
- 將「合作」拖入
- 將使用案例拖入
- 繪製相依關係
- 最後（在「模型瀏覽器」中）從「合作」中刪除圖型

建議：對於每一個使用案例實現化，請新增一個「參與者」圖型，以顯示參與實現化的分析類別（亦即，在互動圖上出現實例來描述使用案例實現化的分析類別）及這些類別之間爲了支援互動圖所描述的合作而存在的關係。請參閱**圖 6-6**

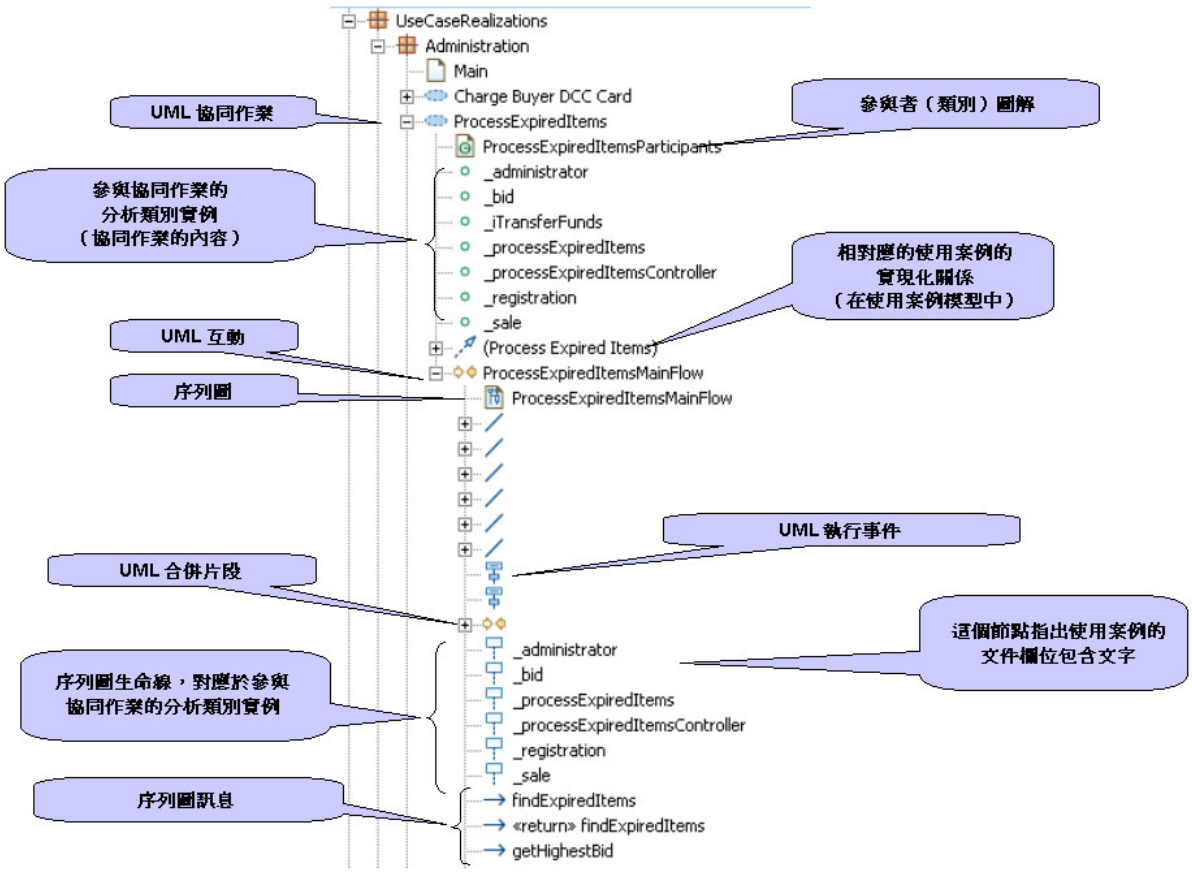
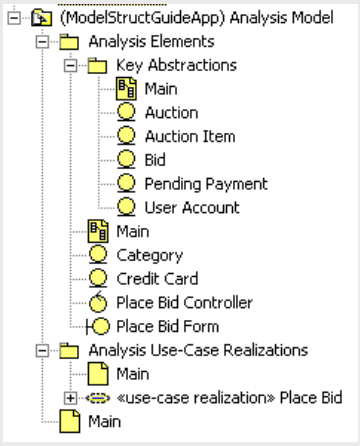


圖 6-6

XDE/Rose

爲了讓 RSx 專注於分析類別的功能導向式套件組織，傳統上對「分析模型」所建議的結構（如下所示）已做修改。同時請注意，在 <<perspective>> 套件中已使用一或多個「關鍵抽象」圖取代「關鍵抽象」套件（它危及了其他功能導向套裝方法）。



7. 設計模型的內部組織準則

設計模型高階組織

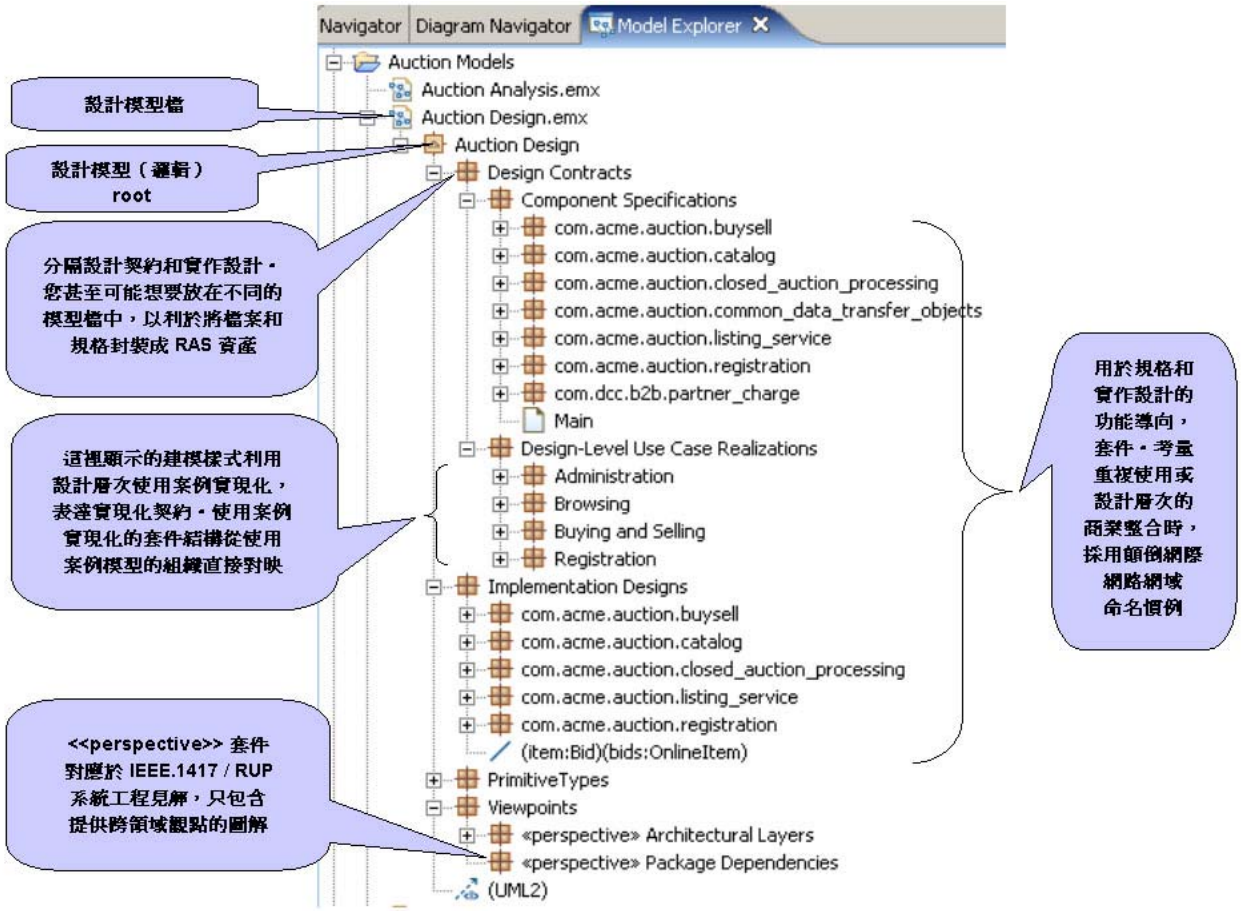


圖 7-1

圖 7-1 說明下列的設計模型建構準則：

1. 分隔實作設計中的規格。此圖例顯示最上層「設計合約」的使用及用於達成上述作業的「實作設計」套件。

2. 利用更下層的套件來進行功能導向分組。例如，以分析期間所用的組織開始，隨著決定分析類別如何對映至實際的設計類別、元件及服務而自然進化。（任何初步的組織方法可能都會在設計期間進化 -- 請參閱以下進一步的討論）。

   在此需要討論一下「子系統」。在 UML 第 2 版以前，子系統是一種特殊化的套件。到了 UML 2，子系統變成一種特殊化的元件，「元件」中可能含有套件。因此，在 UML 2 中，<<subsystem>>「元件」是除了套件以外可行的組織/名稱空間，但對於子系統或套件的適當用法，UML2 仍然沒有明確的交代。建議：在某種分解程度上（例如，特定應用程式的設計子系統）使用「套件」，在整體企業的架構觀點

上，保留以「子系統」來代表整個應用程式（例如 CRM 或 SCM)。

> *XDE/Rose*
> *本文在撰寫時原本預期 Rose 和 XDE 模型匯入工具會提供選項，以爲只要套用 <<subsystem>> 關鍵字，就可將 UML 1.x 子系統對映至 UML2「子系統」或套件。*

3. 設計元素的組織和系統的使用案例（在使用案例模型中，如果維護獨立的分析模型，也可能在分析模型中）如何組織很可能各自分開進化。利用套件將設計契約進一步細分爲設計元素規格（使用契約）及設計層次使用案例實現化（實現化契約），並維護使用案例實現化的套件子結構，繼續反映使用案例本身的組織。

4. *請考慮*使用架構層做爲第二層組織結構的基礎，構成功能區的規格和實作設計的元素（請參閱以下進一步的討論）

5. 在語意模型元素的 UML 分組元件和套件內，放入圖型來提供此群組特有的觀點。這項指導原則指出分組是否根據商業領域的功能導向子集、架構層等，諸如此類。請以套件或元件本身的名稱來命名「預設」圖型，加以編製來展現套件內容的概觀。這樣可讓一些圖型更貼近所要闡述的意義，也更容易導覽和了解模型。

6. 您可能會在設計模型中導入顛倒式網際網路網域名稱空間。理由：
   - 基本上，如此做的相同原因對於語言特定的實作是很重要的：
     a. 涉及到有多個模型驅動應用程式牽涉在內之整合運作的情況（特別是與合作夥伴）
     b. 重覆使用情況
   - 這可能會簡化要實作之轉換的後續配置（來源至目的位置及名稱的對映）。

7. *請考慮*在目標實作平台上使用有效的套件名稱，以避免名稱空間對映的麻煩和可能發生的混淆。（大致上，這只不過意謂著「在名稱中，除了底線，不要使用空格或標點符號」。）

8. 使用小寫套件名稱，以利於區別套件名稱和套件中的類別名稱。

9. *請考慮*在「介面」和「元件」或實現用的「類別」上使用不同的名稱。請針對介面及實作名稱分別使用 ILoan 及 Loan，或 Loan 及 LoanImpl。這在模型中並非絕對需要，但在產生的程式碼中卻是不錯的作法，這樣您又可省去一些後續的轉換配置工作。

10. 在下列情境中，任何不打算產生程式碼的分析層次內容，應該另外放在以 <<analysis>> 爲模板的套件內[8]。
    - A) 您已選擇不採用獨立的分析模型，決定在設計模型中填入分析層次內容，並在分析抽象層次上維護此內容，同時在相同模型中建立設計層次內容；
    - B) 您將從「EIT 設計模型」中進行模型至程式碼轉換

11. 利用 <<perspective>> 套件中的圖型來擷取高階、整體的設計元素觀點。理由：提供整體觀點、「重大架構性」內容的觀點及吸引各類關係人的觀點，同時保持以功能導向分組來編排模型的語意元素。

[8] 轉換時會略過這些套件。

請注意，設計模型的封裝結構會與時俱進。組織方式最終應該反映出您如何將架構建構成元件和服務。這種設計的*最終*策略組織方法，通常最有利於封裝可重複使用的資產，也最可能以最直接的方式從設計對映至專案和資料夾（將存放從設計產生的實作構件，包括程式碼、Meta 資料、文件）。

然而，*初步*組織多少會直接對應至您在「使用案例」模型中採用的組織方法，然後在分析期間修正[9]。事實上（如先前在「分析模型的內部組織準則」中所述），您可以選擇讓分析模型就地進化成設計。換句話說，設計的初步組織趨向於將相似但鬆散耦合的商業問題聚在一起，並隔離通用或可重複使用的元素。這種達成初步組織的方式很有效率，因爲

- 如果希望從分析或使用案例模型內容中使用會產生設計模型內容的轉換，則來源套件至目的套件的對映會很簡單又直接
- 根據功能面相似但關係鬆散的套件所形成的初步組織方法，無疑地最有利於對映至最終的元件導向組織，意謂著在設計過程中可以減少重新分解的工作。
- 套件的鬆散性有利於改善團隊工作流程，如果設計分解成多個建模檔，也有助於重複使用

當然有可能的替代方法，而且在一些情況下建議使用*端點策略*組織：

- 如果以 J2EE 型 Web 應用程式爲目標，包括 EJB，則設計的組織可能需要爲 RSA 和 Rational Application Developer 在 J2EE 專案方面的慣例預做準備。[10] 尤其，您可能選擇定義對應於架構層的最上層設計套件（呈現層和商業層，商業層細分爲階段作業和領域）。這當然不是一種與平台無關的方法，只有當您確定所設計的解決方案不會在 J2EE 以外的平台上實作時才可行。
- 更常見的情況，N-層應用程式通常會將專業開發人員和人力部門對應至呈現層和商業層，因此，同樣地，您可能選擇使用對應於這些架構層的最上層套件。但請小心安排打算支援特定*商業功能*的類別，以支援特定的*架構*。它會使商業功能及架構更難以變更。
- 如果決定採用非元件/服務/子系統導向的組織方式，只要另外再投入時間來配置程式碼產生轉換，您還是可以將設計的組織對映至一組目標專案和資料夾。有一種特殊類型的隨附模型稱爲「對映模型」，可用於定義特別複雜的對映。

設計模型內容

設計模型中應該放入什麼並無嚴格的規定，但下列建議或許有用。

[9] 分析類別的封裝在找到時通常會儘量重新分解，目的是爲了支援重複使用性和超出預料的功能面需求。

[10] 大略而言：每系統或應用程式的「企業專案」或大型的子系統，及針對每個「企業專案」來說呈現層級的專案 Web 專案，及 EJB 專案一般對應到元件或次要子系統的多個 EJB 專案，及一般個別 EJB 專案用於每元件或子系統的階段作業（階段作業 EJB）和領域（實體 EJB）的位置。如需相關資訊，請參閱本文件的第 9 節。

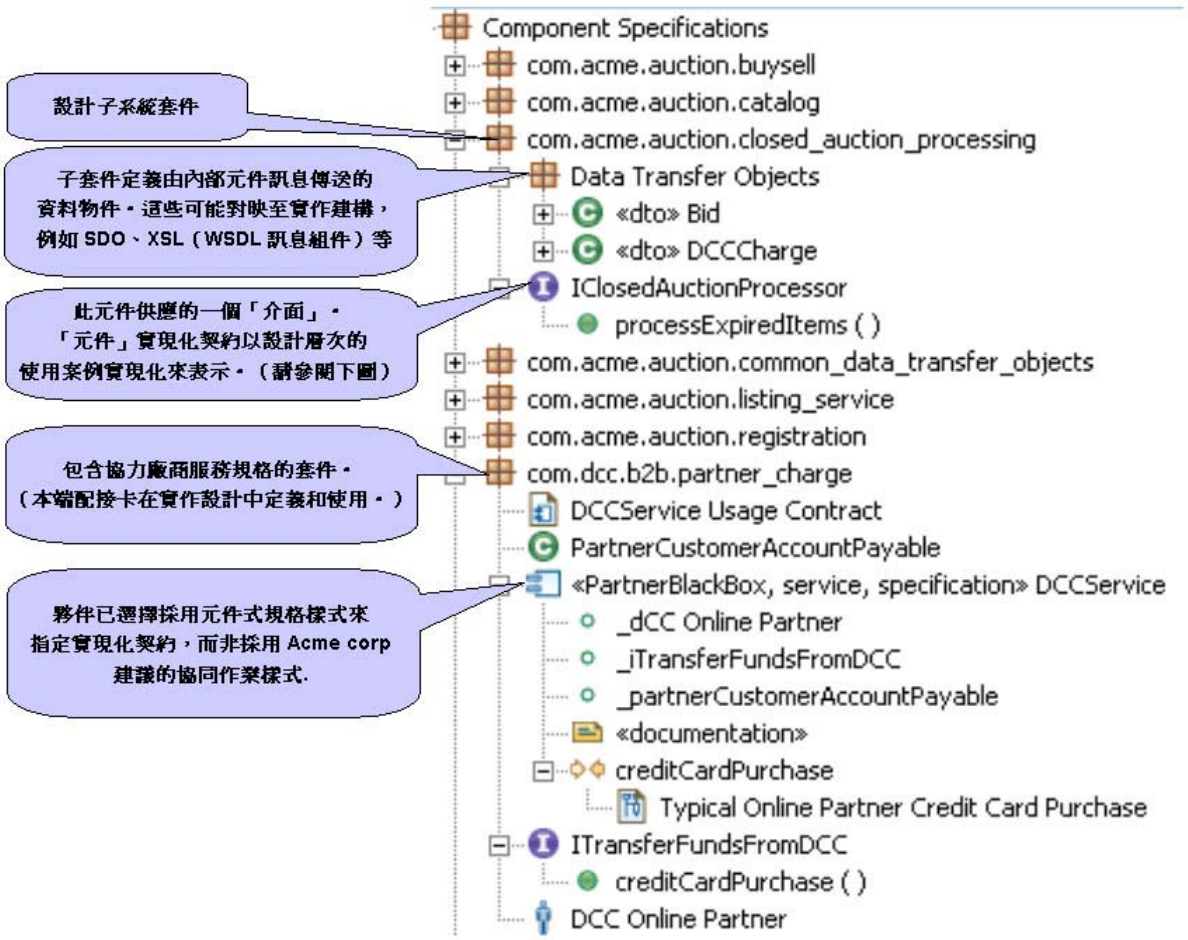

設計子系統套件
子套件定義由內部元件訊息傳送的資料物件。這些可能對映至實作建構，例如 SDO、XSL（WSDL 訊息組件）等
此元件供應的一個「介面」。「元件」實現化契約以設計層次的使用案例實現化來表示。（請參閱下圖）
包含協力廠商服務規格的套件。（本端配接卡在實作設計中定義和使用。）
夥伴已選擇採用元件式規格樣式來指定實現化契約，而非採用 Acme corp 建議的協同作業樣式.
Component Specifications
com.acme.auction.buysell
com.acme.auction.catalog
com.acme.auction.closed_auction_processing
Data Transfer Objects
«dto» Bid
«dto» DCCCharge
IClosedAuctionProcessor
processExpiredItems ( )
com.acme.auction.common_data_transfer_objects
com.acme.auction.listing_service
com.acme.auction.registration
com.dcc.b2b.partner_charge
DCCService Usage Contract
PartnerCustomerAccountPayable
«PartnerBlackBox, service, specification» DCCService
_dCC Online Partner
_iTransferFundsFromDCC
_partnerCustomerAccountPayable
«documentation»
creditCardPurchase
Typical Online Partner Credit Card Purchase
ITransferFundsFromDCC
creditCardPurchase ( )
DCC Online Partner

圖 7-2

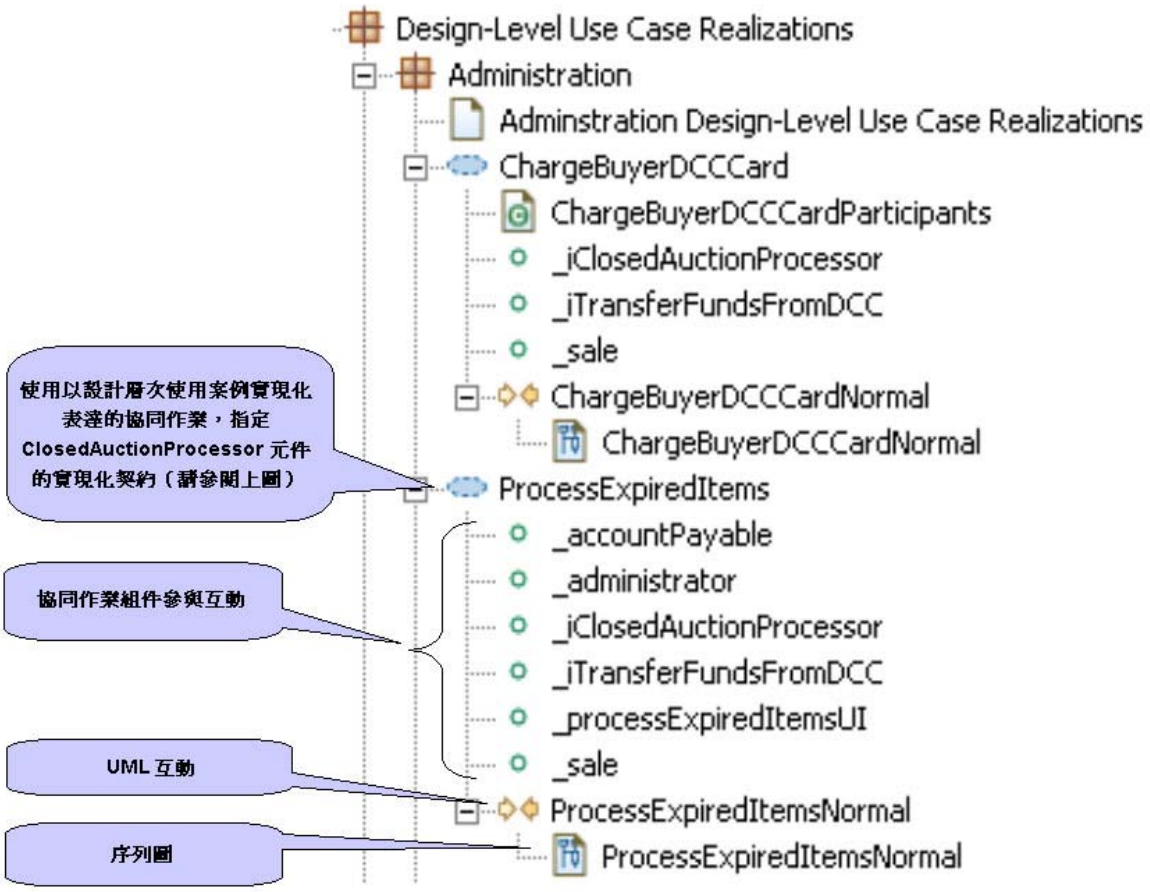


圖 7-3

圖 7-2 和**圖 7-3** 接續**圖 7-1** 的組織結構，並描述如何指定設計合約。

- "ClosedAuctionProcessor" 元件的使用契約以單一「介面」表示[11]（**圖 7-2**）。相對應的實現化契約是由以「合作」表示的單一設計層次使用案例實現化來指定[12]（**圖 7-3**）請注意，分析層次使用案例實現化呈現分析類別之間的合作，而設計層次實現化呈現較不抽象的設計元素之間的合作[13]。如果設計模型的規格子集和實作設計子集必須分開封裝，則設計層次使用案例實現化在角色中只能使用分析或設計規格元素 – 絕不能使用實作設計元素。
- 協力廠商 "DCCService" 的使用契約和實現化契約全部都在一個套件中[14]。同樣地，我們看到使用契約只有一個「介面」，但在此例中，實現化契約以 <<specification>>「元件」來表示（**圖 7-2** ）。（否則，實現化契約的規格也會剛好一樣，也是以行爲來表示 – 在此例中是指 CreditCardPurchase 的互動）。有另一個使用「元件」來代替「合作」的例子，請參閱**圖 7-4**

[11] 元件當然可以有多個供應介面，只是在這個範例中剛好只有一個

[12] 其他元件可能參與多個系統使用案例，所以實現化契約可能存在多個使用案例實現化中。在此類案例中，您也可在具有元件介面的相同套件中納入名稱爲「{元件名稱}使用位置」的圖型，在這個圖型上放置組成這些使用案例實現化之各種圖型的鏈結。

[13] 另一項可能的差異：設計層次實現化中的一些參與者圖型可能是描繪元件佈線的「元件圖」，而不是（或除了）特定的分析層次使用案例實現化適用的「類別圖」。

[14] 請注意，這裡的假設狀況是 DCC 公司提供 UML 規格給 Acme 公司，供 Acme 加入到其設計模型中。這是顛倒式網際網路網域空間可派上用場的一種情況。

- 作業在介面中定義，可由 <<specification>> 元件（如果使用的話）來實現，或由「實作設計」中實作介面的分類器來實現。

- 資料傳送物件（做爲供應作業的參數類型，可能對映至實作結構，例如 XML 綱目或 SDO）的規格也可以納入使用契約中。對於不打算分送的元件，您可以決定是否將資料傳送物件當做作業參數所用的類型規格。對於可分送的服務（例如 Web 服務），已規定服務的作業不可參照本端位址空間中的物件，所以必須使用 DTO。

> *XDE/Rose*
> 在之前的 UML 版本中，使用案例實現化的指導原則是使用每個使用案例合作實例，對於實現化的每個重要流程則使用互動及序列圖。
>
> 在 RSx 中，您通常可以只使用一個互動和圖型，因爲 UML2 序列圖現在已支援替代執行路徑的表示法。
>
> 同時，在 UML 2 中不再有「合作實例」。只有以「合作」爲類型的「合作使用」。在 RSx 中，請以「合作」來表示使用案例實現化。

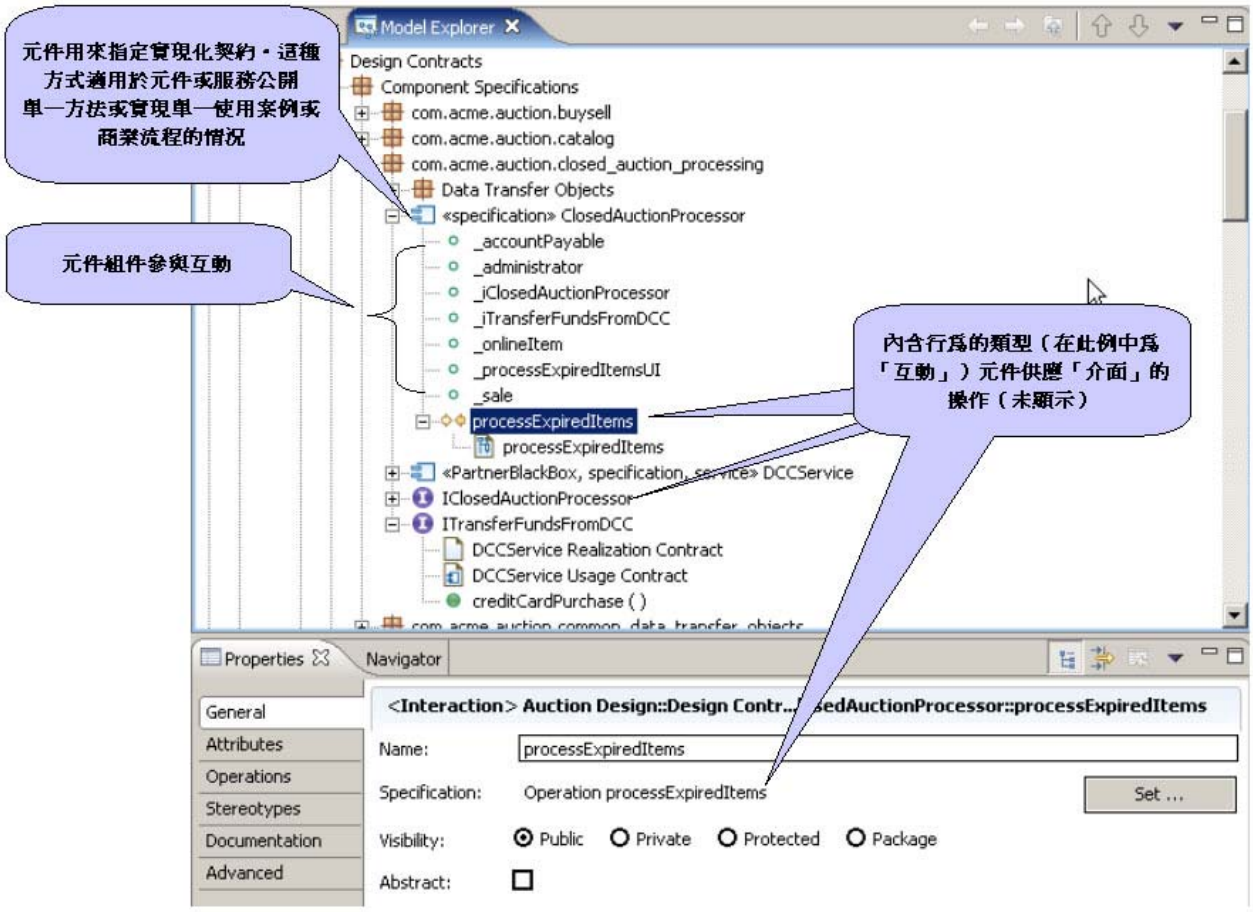


圖 7-4

- 關於指定實作設計的可能方式，請參閱**圖 7-5**。實作結構是以含有作業的簡單類別來定義。這種方式常見於以 UML 1.x 所建立的設計模型中。有第二種可能更符合 UML2 目標的方式，請參閱**圖 7-6**。在此，沒有「類別」，我們只看到使用「元件」，「元件」沒有「作業」，只有行爲（在此例中是指「互動」）。

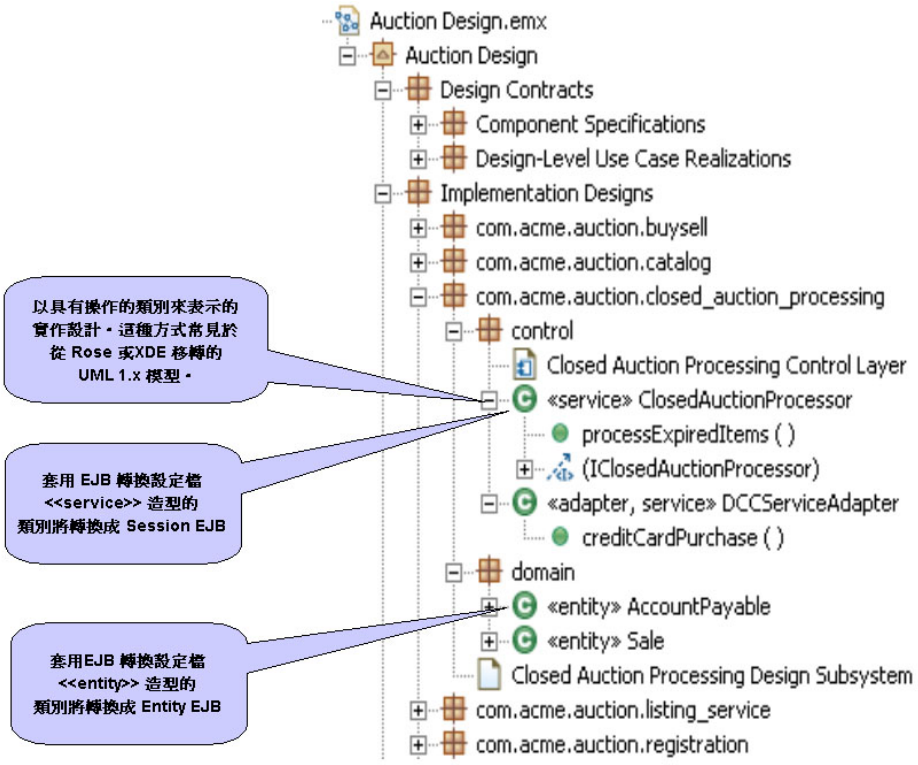

Auction Design.emx
Auction Design
Design Contracts
Component Specifications
Design-Level Use Case Realizations
Implementation Designs
com.acme.auction.buysell
com.acme.auction.catalog
com.acme.auction.closed_auction_processing
control
Closed Auction Processing Control Layer
«service» ClosedAuctionProcessor
processExpiredItems ( )
(IClosedAuctionProcessor)
«adapter, service» DCCServiceAdapter
creditCardPurchase ( )
domain
«entity» AccountPayable
«entity» Sale
Closed Auction Processing Design Subsystem
com.acme.auction.listing_service
com.acme.auction.registration
以具有操作的類別來表示的實作設計。這種方式常見於從 Rose 或XDE 移轉的 UML 1.x 模型。
套用 EJB 轉換設定檔 <<service>> 造型的類別將轉換成 Session EJB
套用EJB 轉換設定檔 <<entity>> 造型的類別將轉換成 Entity EJB

圖 7-5

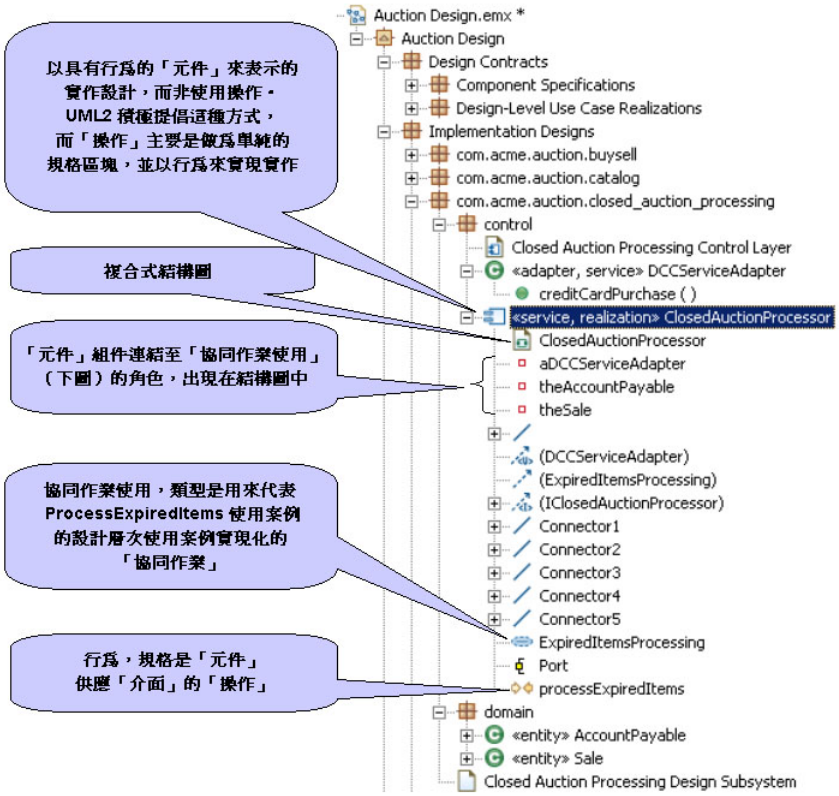

Auction Design.emx *
Auction Design
Design Contracts
Component Specifications
Design-Level Use Case Realizations
Implementation Designs
com.acme.auction.buysell
com.acme.auction.catalog
com.acme.auction.closed_auction_processing
control
Closed Auction Processing Control Layer
«adapter, service» DCCServiceAdapter
creditCardPurchase ( )
«service, realization» ClosedAuctionProcessor
ClosedAuctionProcessor
aDCCServiceAdapter
theAccountPayable
theSale
(DCCServiceAdapter)
(ExpiredItemsProcessing)
(IClosedAuctionProcessor)
Connector1
Connector2
Connector3
Connector4
Connector5
ExpiredItemsProcessing
Port
processExpiredItems
domain
«entity» AccountPayable
«entity» Sale
Closed Auction Processing Design Subsystem
以具有行為的「元件」來表示的實作設計，而非使用操作。UML2 積極提倡這種方式，而「操作」主要是做為單純的規格區塊，並以行為來實現實作
複合式結構圖
「元件」組件連結至「協同作業使用」（下圖）的角色，出現在結構圖中
協同作業使用，類型是用來代表 ProcessExpiredItems 使用案例的設計層次使用案例實現化的「協同作業」
行為，規格是「元件」供應「介面」的「操作」

圖 7-6

8. 實作概觀模型的內部組織準則

> *XDE/Rose*
> 在「XDE 模型結構準則」中，建議以「實作概觀」模型做爲裝置來提供實作的子系統層次概觀。然後在實作子系統的專案的程式碼模型中，指定每一個子系統的詳細資料。

嚴格來說，在 RSx 中不需要使用「實作概觀」模型。只要遵守設計模型組織準則，自然可從元件（包括笨重的 <<subsystem>> 和較易分送的各種 <<service>> ）形成設計模型的（最終）組織。之後，透過轉換，設計的套件可對映至專案。例如，在 J2EE 實作中，套件將對映至各種 Java、EJB、Web、J2EE 應用程式及其他開發實作的專案。（如本文的「基本概念及術語」章節所述，這些專案實際上代表解決方案的實作模型。）

如果從由下而上的角度來思考，您應該思索如何以專案和資料夾來組織實作，並分解成設計模型的組織，最後達成直接的轉換對映。或者，如果以由上而下來看待，則設計模型的組織（由於已在設計過程中進化）應該決定所需的實作模型集（專案）。不論何種方式，設計模型的最終組織應該自然就解決專案和子專案的概觀需求，亦即，在設計模型中描繪套件的圖型就等同於專案和子資料夾的概觀。

不過，您仍可能喜歡在早期就擬好專案結構，或只是喜歡以更明確的手法來刻畫專案結構 – 例如，代表專案和資料夾的構件明確地以 <<project>> 和 <<folder>> 爲關鍵字，甚至以 <<EJB Project>> 和 <<Web Project>> 爲關鍵字。另外一項考量是設計模型不適合描寫太細部的實作構件（例如 JAR）（這在 Rational Software Architect 作業理論中很容易變成與平台無關）。但該類構件卻可在「實作概觀」模型中完全接受。因此，您有時可能需要使用「實作概觀」模型。**圖 8-1** 顯示一個「實作概觀」模型範例

最後要思考的是「實作概觀」模型可能是擷取解決方案不同層面之非正式圖型的好地方。**圖 8-2** 顯示拍賣系統的非正式整體概念圖，本文的大多數範例以此爲基礎。

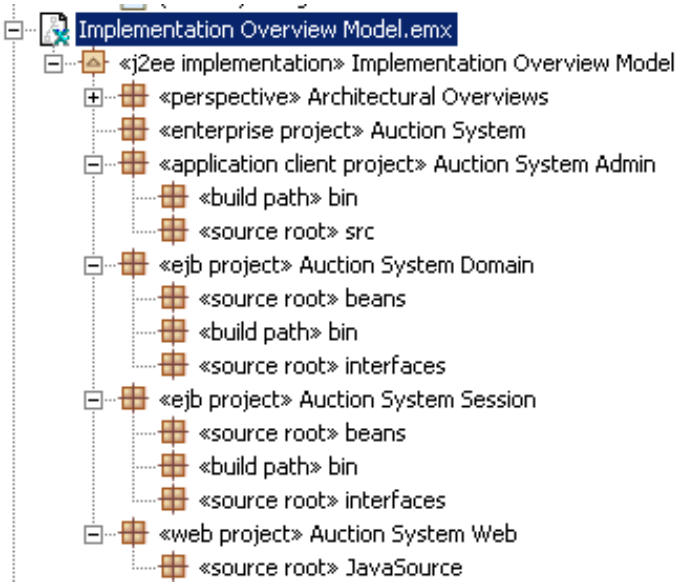

Implementation Overview Model.emx
«j2ee implementation» Implementation Overview Model
«perspective» Architectural Overviews
«enterprise project» Auction System
«application client project» Auction System Admin
«build path» bin
«source root» src
«ejb project» Auction System Domain
«source root» beans
«build path» bin
«source root» interfaces
«ejb project» Auction System Session
«source root» beans
«build path» bin
«source root» interfaces
«web project» Auction System Web
«source root» JavaSource

圖 8-1

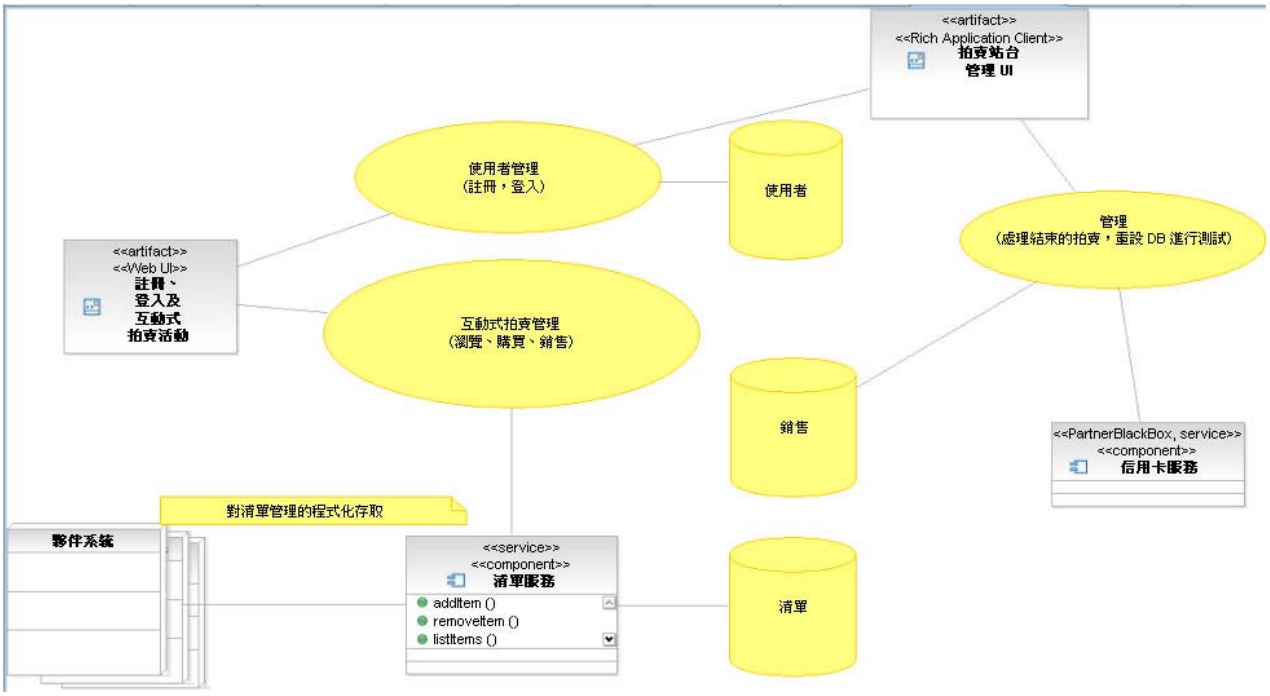

<<artifact>>
<<Rich Application Client>>
拍賣站台
管理 UI
使用者管理
(註冊，登入)
使用者
管理
(處理結束的拍賣，重設 DB 進行測試)
<<artifact>>
<<Web UI>>
註冊、
登入及
互動式
拍賣活動
互動式拍賣管理
(瀏覽、購買、銷售)
銷售
<<PartnerBlackBox, service>>
<<component>>
信用卡服務
對清單管理的程式化存取
零件系統
<<service>>
<<component>>
清單服務
addItem ()
removeItem ()
listItems ()
清單

圖 8-2

9. 部署模型的內部組織準則

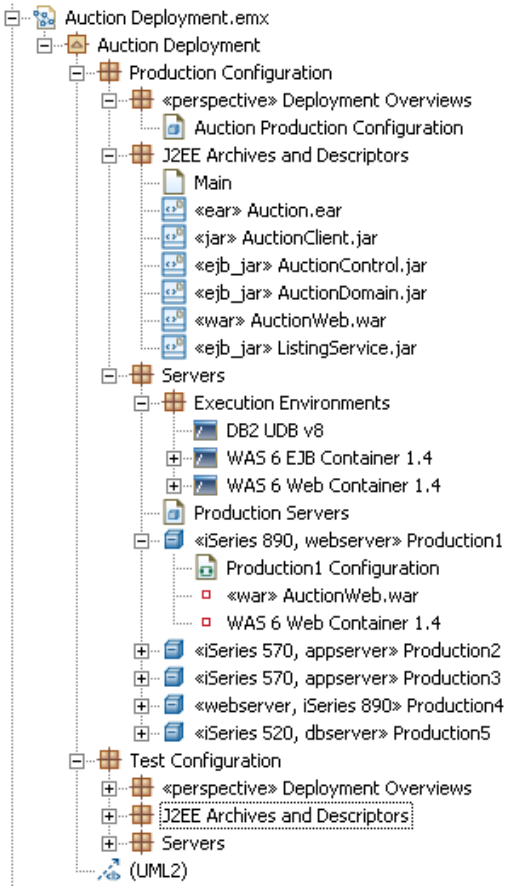


圖 9-1

部署模型需要解釋的部分可能少於本文所描述的其他任何模型。部署建模組織和內容選擇通常沒有太多含意，只要順理成章即可。儘管如此 – 爲了幫助您思索 – 還是提出可能的策略和些許典型的內容，請參閱**圖 9-1**。此範例只有幾個要點：

1. 正式配置和測試配置的規格已分開。

2. 概觀（例如，叢集、資料中心或企業）放在 <<perspective>> 套件中維護

3. 採用簡易的方式將節點和構件特殊化及分類：封裝和關鍵字混用。更準確的方法是開發特殊化的 UML 設定檔，用來定義特殊化的模板和內容，以適當說明和記載在您自己的環境中所使用的類型和資源。

10. 使用建模檔來代表「軟體架構文件」

給定用來組織模型的工具，例如圖型鏈結，並以跨模型參照支援多個模型檔，如此建立有效代表 RUP 軟體架構文件及「架構的 4+1 視圖」變成只是平凡的瑣事。

您至少可以按照**圖 10-1** 來做些動作。建立建模檔案並使用對應於 4+1 視圖的簡式套件集合移入資料。（範例中沒有「流程觀點」的套件，因爲範例中的系統未展現太多的並行性。）

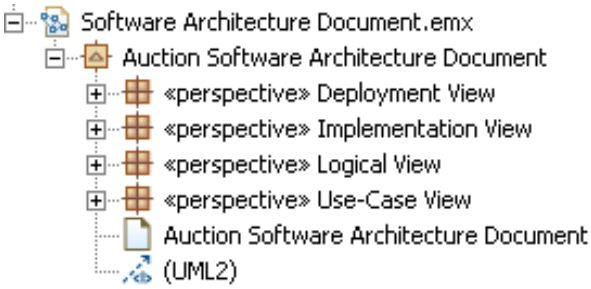


圖 10-1

接著，多半按照**圖 10-2** 的建議來編製預設圖型。您也可以在此圖型中加入其他附註或文字

圖 10-2

然後，只要使用下列方法在「軟體架構文件」建模檔案中建立圖型：

- 利用其他建模檔中的 UML 語意元素來建立圖型，描繪在這些建模檔中缺乏但在架構文件中有需要的新觀點。

- 建立由幾何圖形和/或位於「軟體架構文件」建模檔案中的「特別」UML 元素編製的圖型。（這些 UML 元素僅止於說明或澄清，對於所描述的解決方案的實際實作沒有語意上的重要性）

- 建立只包含對其他建模檔案中現有圖型之鏈結的圖型。（如果架構文件建模檔案與其他建模檔案一起分送供讀者消費，此技術將可正常運作。如果架構文件要改在 Web 上發行，請改採其他方式）

11. 團隊開發和模型管理注意事項

本節說明將模型分割成多個建模檔的時機和原因。關於這些課題的完整論述，請參閱 RSx 線上說明。在此假設讀者已稍微熟悉並行開發的概念，也具有將多個構件副本的並行變更合併起來的觀念。

首先快速復習一下：在「基本概念及術語」章節，我們討論 RUP 認可的各種模型類型，例如使用案例、分析及設計。提供的範例在 RSx 中指出…

- 如果建置多個應用程式，則一個類型可能會有多個模型（例如，多個使用案例模型、多個分析模型等）

- 一個模型（在邏輯上）可能儲存爲一或多個建模檔，例如，應用程式 'X' 的設計模型可能儲存爲單一建模檔或一群建模檔集合。

分割模型

關於將模型分割成多個建模檔的技術，請參閱 RSx 線上說明，這裡不討論。在此我們只關心何時及爲何要分割。在兩種情況下，您可能選擇將一個特定的模型分成多個建模檔來維護：

1. 模型過大，難以掌控，或封裝結構太深，難以管理[15]

2. 一個建模檔開始出現太多並行變更，以致於需要合併 2 個以上的要素項[16]。（如果您不明白這段陳述，請繼續閱讀。）

在團隊中建模

當您的配置管理原則允許並行開發模型時[17]，檔案會發生不對等的變更（事實上是變更邏輯模型或檔案所代表的模型子集）。到了一定程度時必須合併變更。合併變更時若發生衝突，則需要**非任意性**合併，因爲有人必須從衝突的變更中決定何者應該「獲勝」。（「任意」合併表示不對等的變更沒有衝突，模型合併引擎可以在不須人爲介入下進行合併。）
非任意性合併做起來可能很困難。如何儘量避免呢？

有兩種基本的手段可以避免衝突和最後導致的非任意性合併。其中之一是「健全的架構」。另一種是「強大的所有權」。兩者形影相隨：健全的架構「賦予」強大的所有權。

手段 #1：健全的架構

「健全的架構」在此主要是指分解。這裡引述的架構分解*原則*就是推動「物件導向開發」、「元件式設計」及「服務導向架構」的原則：

---

[15] 大多是因爲檔案太大，使用者群組的機器無法負荷時，才需要分割。例如，在 1GB RAM 機器上，每天很難處理磁碟上達到 30MB 的模型。在這樣的機器上，一定很想分割模型，希望 RAM 中隨時只保留 5 至 10MB 的模型。另一種辦法是升級 RAM（相當廉價的解決方案，但也是非常好的方案 -- 在 2GB RAM、沒有交換檔的機器上，可以更快速執行幾乎每一個 Eclipse 作業，同樣地，非常大的模型執行起來也相當良好。）

[16] RSx 模型合併工具頂多支援合併 3 個要素項：兩個「變更」要素項，一個「基本」要素項（又稱爲共同上代）。需要處理 2 個以上的「變更」要素項時，合併會開始串聯，從這點開始，其中一個「變更」要素項成爲階段作業中先前已完成的合併的結果集。

[17] 「並行」開發原則的例子：

・建模檔可能以非獨佔存取方式移出

・開發流程中有多位工作者並行使用一個建模檔

- 努力將商業功能完全拆解

- 將必須緊密接合的事物聚在一起。將各組分開隔離

- 如果分解結果出現大量細目，視人員配置模型而定（切記，健全的架構和強大的所有權形影相隨），您可以開始將這些細目分組成密切關係的聚集（以建模術語來說就是「UML 套件」）

- *總是*會有一些事物必須由許多（有時是全部）分解單位來接觸。請將這些事物放在「通用」套件中，然後規劃每一次的開發反覆，在反覆初期加上少許的「瀑布式」形式，以穩定「通用」項目爲重點。

- 另外還有一個時間元素。隨著從最初很抽象的想法開始，一直到對解決方案有更具體的了解之後，您將逐漸領悟到架構（和模型）的最佳組織。您應該規劃每一個階段轉換時的模型重構（重組）活動（商業分析、需求、應用分析、高階設計...）。

如果查看解決方案之後發現所有事物都糾纏不清，則表示架構需要修改，或問題領域的本質有疑點，確實無法拆解問題。不論何種情況，您有幾項作法：

- 決定將專案指派給一個超小型團隊，此團隊共用一個實際場所，且彼此很積極溝通任何可能影響其他構件的變更。
- 準備執行大量全面性合併

XDE/Rose

如果您是 Rose 或 XDE 使用者，則可能曾經做過全面性模型合併。幸好，在 RSx 中，全面性合併不再困難。有一項主要的差異在於 RSx 處理錯綜復雜的相關建模檔，不是層級分明的小型檔或次單元。 這項重大差異表示在 RSx 中，我們對實體層次上的高階邏輯分解有更好的支援。

我們也以更好的工具來支援比較合併 – 事實上是一個絕佳的合併過程。

- 在 RSx 中，後端合併引擎比 XDE 後端快上 10,000 倍（沒錯，一萬倍）。
- 也提供許多技術，例如「複合 Delta」（依圖型來將變更分類的一種分組機制，可將大量圖解動作組合在一起執行群組及/或不可分割的作業）、「模型整合性保護」、「不可分割性」及「串聯式 Delta 處理」。經由像直接編輯檔案一樣地合併，可以儘量避免檔案損毀。
- 現在有一個真正視覺化的圖型合併 GUI，可以解決排版/修飾的衝突

手段 #2：強大的擁有權

建立健全的架構分解之後，架構元件的「強大」所有權就可以很直接地對映至個別的工作者或小團隊（姑且不論專業技能）。當模型中的每一個邏輯套件（或分支）可以由一位工作者獨佔使用時，該模型所執行的合併最瑣碎（不論模型儲存成單一建模檔或多個建模檔）。如果每一個分支可以由一個經常私底下溝通彼此工作的小組獨佔使用，也會有同樣的情形。

將模型分成多個建模檔，可以避免全面性合併嗎？一句話：不行。**架構相互依賴性是邏輯現象，不是物理現象**。將一個模型分割成多個建模檔時，元素相互依賴性會直接變成跨檔案參照，不再是檔案內參照。對解決衝突沒有幫助（事實上變得更困難）。此外，引進跨檔案參照之後，也埋下可能的破損點（請參閱資訊看板）。

資訊看板：跨建模檔參照

當兩個建模元素位於不同的建模檔時，如果建立兩者之間的關係，就等於建立「跨建模檔參照」。因爲建模檔（.emx 檔）在主機 OS 檔案系統上，可以移動、重新命名或在 Eclipse 環境外修改，這些參照象徵著可能的破損點。不過，只要建模檔一律透過 Eclipse 環境來修改和管理變更，且您也遵守下列準則，應該就不會遇到破損。

每當使用（編輯）具有「封閉性」的建模檔時（亦即，許多彼此參照的建模檔），工作區應該會顯示這一整袋的所有建模檔。這並不絕對表示一整袋的所有建模檔必須在相同專案內，只是單一專案通常可以保證所有模型一定存在，因爲在一般的 CM 工作流程中，所有模型檔在相同的專案內會一起移動。

總結如下：

- 如果缺乏健全的架構，或雖有健全的架構，但缺少強大的擁有權，您將經常面臨全面性合併，無論如何分割也無濟於事。

- 如果有健全的架構和強大的擁有權，將可大幅減少（但無法排除）全面性合併的次數。無法排除是因爲元件之間總是存在相互依賴性。前述的「通用」元素就是一個例子，但不是唯一的例子。

- 與其將模型分割成多個建模檔，倒不如在邏輯上好好建構模型，讓多位工作者在並行使用一個建模檔時不會引起衝突的變更。

- 幸好，RSx 處理模型合併的速度和效率遠超過其他任何可用的建模工具。

何時應該分割模型呢？儘量避免，除非模型的總大小開始考驗硬體的能耐，且您建立的建模檔通常可以獨佔使用（亦即，一個檔案在任何時候只供一位團隊成員移出），也可以隔離使用（亦即，檔案的大部分變更不必存取含有相關模型元素的其他檔案）。

後記：RSx 6.x 版和 7.x 版之間的變更

RSx 最初發行時（6.0 版）並不像 Rose 和 XDE 一樣支援「次單元」的概念（次單元是含有模型子集的建模檔，因爲在模型瀏覽器視圖的模型邏輯觀點中看不到，所以有「透明」性質）。相反地，模型分割僅限於定義多個最上層模型（「最上層」表示在模型瀏覽器視圖中，每一個建模檔是一個獨立的最上層項目）。

RSx 也可讓您選取現有模型的 UML「套件」，根據此套件來「製作模型」。也可以說成「裁切」套件來建立新的建模檔。結果，在模型瀏覽器視圖中，「套件」會變成新的最上層邏輯模型。原始的階層式容納結構會消失，但每當開啓如此「裁切」套件的建模檔時，也會開啓所有「裁切」的模型。這樣會導致不必要的移出，也會造成模型瀏覽器視圖壅塞，編輯器窗格中也會出現大量標籤，最後難以導覽。

當 RSx 7.0 版增加次單元支援之後，您可以將模型分割成多個建模檔，也不會讓階層式結構消失，加上在編輯器窗格中已去除「模型編輯器」標籤，也克服其他導覽問題。

不過，舊版 6.x 中有一項模型建構技術至今仍然有用，值得學習。如上所述，開啓已裁切「套件」的建模檔也會開啓所有裁切的模型。只要事前規劃好分割策略，不要使用「製作模型」特性，即可避免。

採用「製作模型」特性的人通常從單一建模檔開始，然後建立「套件」來代表解決方案架構的組織。例如，解決方案的每一個主要元件或子系統可能有一個「套件」（「元件」和「子系統」只是反映昔日運算時代的非正式用語，不是正式的 UML 語意定義）。”common”、”utility” 或 “framework” 元件也可能會有「套件」。隨著模型成長，對應至應用程式專用元件的「套件」可能會「裁切」成獨立的建模檔，於是，建置這些元件的團隊（在理論上）可以各自使用這些模型檔。但事實上，開啓任何元件專用的模型檔會導致開啓原始的「主控」模型（含有「通用」部分），進而開啓其他所有應用程式元件專用的模型。如前所述，最後會造成不必要的移出和導覽困難。

最好事先規劃，爲每一個應用程式專用元件定義單獨的模型，並爲「通用」元件定義一個模型（或多個模型，定義連續幾層的通用/可重複使用的元件，亦即，反映實作架構的抽象層）。然後，任何應用程式專用元件模型可以由負責的團隊開啓，也只有應用程式專用模型所依賴的「通用」模型才需要開啓。